

# DARTS Lab Documentation (Developer)

DARTS Lab  
October 06 2023, 06:14:01

Jet Propulsion laboratory, California Institute of Technology

The research was carried out at the Jet Propulsion Laboratory, California Institute of Technology, under a contract with the National Aeronautics and Space Administration (80NM0018D0004).

## Table of Contents

### *DARTS/Dshell Framework*

1. Dshell++
  - 1.1. Background
    - 1.1.1. Reference & Source material
  - 1.2. Design
    - 1.2.1. Dshell Model Design (aka Models Style Guide Draft)
    - 1.2.2. Dshell Simulation object
    - 1.2.3. Dshell Assemblies
    - 1.2.4. Dshell Models
    - 1.2.5. Linearization
  - 1.3. Usage
  - 1.4. Software
  - 1.5. Raw documentation
    - 1.5.1. DshellCommon: Split Assembly into BaseAssembly and Assembly
    - 1.5.2. Proposal: New state propagation design
    - 1.5.3. Working with FSMs
    - 1.5.4. Notes from issue on the new time stepping scheme.
  - 1.6. Sphinx documentation
    - 1.6.1. Dshell Simulation
    - 1.6.2. Dshell Models
    - 1.6.3. Dshell Signals
    - 1.6.4. Dshell Assemblies
    - 1.6.5. Dshell Parameter Classes
    - 1.6.6. Dshell++ Events
    - 1.6.7. MultiRate EndIOStep Execution
  - 1.7. Regression Tests
    - 1.7.1. Dshell++ Regression Tests
2. DshellCommon
  - 2.1. Background
    - 2.1.1. Reference & Source material
  - 2.2. Design
    - 2.2.1. InputDict: replacement for anonymous dictionaries
    - 2.2.2. DAssembly
  - 2.3. Usage
    - 2.3.1. Module Usage
  - 2.4. Software
  - 2.5. Raw documentation
    - 2.5.1. DshellCommon: Add an optional reference frame path/uuid field to NodeParam class
    - 2.5.2. DshellCommon: Create MarsAssembly etc assemblies specific to known planetary bodies
    - 2.5.3. DshellCommon: New Object-Oriented way to handle Assemblies (design requirements)
    - 2.5.4. DshellCommon: Replacing anonymous dictionaries with explicit classes
    - 2.5.5. DshellCommon: Use of InputDict with Assembly classes notes
    - 2.5.6. DshellCommon: Validation with InputDict
  - 2.6. Sphinx documentation
    - 2.6.1. Getting Started
    - 2.6.2. Reference
3. DIntegrator
  - 3.1. Background
    - 3.1.1. Reference & Source material
  - 3.2. Design
  - 3.3. Usage
  - 3.4. Software
  - 3.5. Raw documents
  - 3.6. Sphinx documentation
    - 3.6.1. Reference
4. CNode
  - 4.1. Background
    - 4.1.1. Reference & Source material
  - 4.2. Design
  - 4.3. Usage
  - 4.4. Software
  - 4.5. Raw documents
    - 4.5.1. Sphinx documentation
5. DshellEnv
  - 5.1. Background
    - 5.1.1. Reference & Source material

- 5.2. Design
  - 5.2.1. Dclick option handling
- 5.3. Usage
  - 5.3.1. Dclick Usage
  - 5.3.2. Typing
- 5.4. Software
- 5.5. Raw documentation
  - 5.5.1. Handling different versions of typing and typing\_extension Python modules
- 5.6. Sphinx documentation
  - 5.6.1. Introduction
  - 5.6.2. DshellEnv Classes
  - 5.6.3. DshellEnv Reference
- 6. Ndarts
  - 6.1. Background
    - 6.1.1. The multibody dynamics problem
    - 6.1.2. Reference & Source material
  - 6.2. Design
    - 6.2.1. Architecture overview
    - 6.2.2. Ndarts Prescribing and Masking Joints
    - 6.2.3. Multibody Modeling with Ndarts - Design
    - 6.2.4. Algorithms available
    - 6.2.5. Additional concepts
  - 6.3. Usage
    - 6.3.1. Creating a multibody model
    - 6.3.2. Using the model
    - 6.3.3. Changing model configuration
    - 6.3.4. Troubleshooting and FAQ
  - 6.4. Software
    - 6.4.1. DartsMbody
    - 6.4.2. DartsBody
    - 6.4.3. DartsHinge
    - 6.4.4. DartsSubinge
    - 6.4.5. DartsNode
    - 6.4.6. DartsSubGraph
    - 6.4.7. DartsTreeDynamicsSolver
    - 6.4.8. CM frame
    - 6.4.9. General Functions
  - 6.5. Raw documentation
    - 6.5.1. Ndarts: Switch DartsBody.setBodyParams() to use createPartGeometries() instead of load() for geometries
  - 6.6. Sphinx documentation
    - 6.6.1. DARTS Multibody Modelings Basics
    - 6.6.2. Ndarts Hinges
    - 6.6.3. Ndarts Prescribing and Masking Joints
    - 6.6.4. Geometric Shapes for Bodies
    - 6.6.5. Multibody Modeling with Ndarts - Design
    - 6.6.6. Ndarts Primary Object Classes API Reference
- 7. DFrame
  - 7.1. Background
    - 7.1.1. Reference & Source material
  - 7.2. Design
    - 7.2.1. DartsFacadeScene
  - 7.3. Usage
    - 7.3.1. DartsFacadeScene
  - 7.4. Software
  - 7.5. Raw documentation
    - 7.5.1. DFrame: Add method to look up frame from a partial frame names path
    - 7.5.2. Dshell++: Support calling updateSceneFrameTransform() at the granularity required by individual models
  - 7.6. Sphinx documentation
    - 7.6.1. Darts/Dshell Frames
    - 7.6.2. Regression tests
- 8. DVar
  - 8.1. Background
    - 8.1.1. Reference & Source material
  - 8.2. Design
  - 8.3. Usage
  - 8.4. Software
  - 8.5. Raw documents
  - 8.6. Sphinx documentation

## 8.6.1. Darts/Dshell Data Inspection and Updating with Dvar

### 9. DataRecorder

#### 9.1. Background

##### 9.1.1. Reference & Source material

#### 9.2. Design

##### 9.2.1. PlotJugglerRecorder

#### 9.3. Usage

##### 9.3.1. PlotJugglerRecorder

#### 9.4. Software

#### 9.5. Raw Documents

#### 9.6. Sphinx Documentation

##### 9.6.1. DataRecorder Logging

##### 9.6.2. DataRecorder Logging Example

##### 9.6.3. DataRecorder Regression Tests

### 10. Dshell++Scripts

#### 10.1. Background

##### 10.1.1. Reference & Source material

#### 10.2. Design

#### 10.3. Usage

#### 10.4. Software

#### 10.5. Raw documents

#### 10.6. Sphinx documentation

##### 10.6.1. Logging with DebugLog

##### 10.6.2. Regression tests

### 11. SOA

#### 11.1. Background

##### 11.1.1. Reference & Source material

#### 11.2. Design

##### 11.2.1. SOAVector vector class

##### 11.2.2. SOAMatrix matrix class

##### 11.2.3. SOAQuaternion unit quaternion class

##### 11.2.4. SOAHomTran homogeneous transform class

##### 11.2.5. SOASpatialInertia spatial inertia class

#### 11.3. Usage

#### 11.4. Software

##### 11.4.1. SOARodriguesParam

#### 11.5. Raw documentation

##### 11.5.1. SOA: Implement default object value printouts using pydump()

#### 11.6. Sphinx documentation

##### 11.6.1. SOAVector vector class

##### 11.6.2. SOAMatrix matrix class

##### 11.6.3. SOAQuaternion unit quaternion class

##### 11.6.4. SOAHomTran homogeneous transform class

##### 11.6.5. SOASpatialInertia spatial inertia class

##### 11.6.6. Rotational Conventions

##### 11.6.7. Inertia Tensor

#### 11.7. Inertia Tensor and Products of Inertia Integral Sense

##### 11.7.1. Inertia Tensor

##### 11.7.2. Products of Inertia - Negative Integral Sense

##### 11.7.3. Products of Inertia - Positive Integral Sense

##### 11.7.4. Simulations

##### 11.7.5. SOASpatialInertia

##### 11.7.6. Additional Information

##### 11.7.7. Footnotes

### 12. Spice

#### 12.1. Background

##### 12.1.1. Bodies and Frames in NAIF

##### 12.1.2. SpiceFrameContainer

##### 12.1.3. SpiceFrame

##### 12.1.4. SpiceFrame2Frame

##### 12.1.5. Reference & Source material

#### 12.2. Design

##### 12.2.1. Spice Kernel Loading

#### 12.3. Usage

##### 12.3.1. Using SpiceFrames in your simulation

##### 12.3.2. Spice Utils

##### 12.3.3. spkcheck

#### 12.4. Software

- 12.5. Raw documents
- 13. DMesh
  - 13.1. Background
    - 13.1.1. Reference & Source material
  - 13.2. Design
    - 13.2.1. Mesh class design
    - 13.2.2. Mesh Utilities
    - 13.2.3. DMeshObject
  - 13.3. Usage
    - 13.3.1. DMeshObject
  - 13.4. Software
  - 13.5. Raw documents
- 14. CORE
  - 14.1. Background
    - 14.1.1. Reference & Source material
  - 14.2. Design
    - 14.2.1. CORE objects
  - 14.3. Usage
  - 14.4. Software
  - 14.5. Raw documents
- 15. SimScapeBasic
  - 15.1. Background
    - 15.1.1. Reference & Source material
  - 15.2. Design
  - 15.3. Usage
    - 15.3.1. Importing GeoTiff files
  - 15.4. Software
  - 15.5. Raw documents
- 16. DScene
  - 16.1. Background
    - 16.1.1. Reference & Source material
  - 16.2. Design
    - 16.2.1. SceneObject flags
    - 16.2.2. CallbackRegistry
  - 16.3. Usage
    - 16.3.1. CallbackRegisry
  - 16.4. Software
  - 16.5. Raw documentation
    - 16.5.1. DScene: support of Hapke and Principled material through BodyDParam and general improvements
    - 16.5.2. DScene: Add better support for client scenes that do not support scene graph constructs
- 17. FacadeScene
  - 17.1. Background
    - 17.1.1. Reference & Source material
  - 17.2. Design
    - 17.2.1. Registering client scenes
    - 17.2.2. Various object types
  - 17.3. Usage
  - 17.4. Software
  - 17.5. Raw documents
- 18. OptixScene
  - 18.1. Background
    - 18.1.1. Reference & Source material
  - 18.2. Design
    - 18.2.1. Key components and their relationships
    - 18.2.2. Proposed restructuring of sensor/viewpoint classes
    - 18.2.3. Renderables
    - 18.2.4. Sensor Frames
  - 18.3. Usage
    - 18.3.1. Assigning materials to partGeometries
    - 18.3.2. Assigning materials to partGeometries
    - 18.3.3. Advanced topics
  - 18.4. Software
    - 18.4.1. Event queue implementation
    - 18.4.2. Upgrading Optix and CUDA
  - 18.5. Raw documentation
    - 18.5.1. OptixScene: Stop using scene graph approach for updating transforms
    - 18.5.2. Cleanup and Redeisgn of OptixScene 2023
- 19. DMeshScene

- 19.1. Background
  - 19.1.1. Reference & Source material
- 19.2. Design
- 19.3. Usage
- 19.4. Software
- 19.5. Raw documentation
  - 19.5.1. DMeshScene: DMesh collision detection capability using embree's rtcCollide method
- 20. DBullet
  - 20.1. Background
    - 20.1.1. Reference & Source material
  - 20.2. Design
    - 20.2.1. MeshToBullet2
    - 20.2.2. MeshToSDF
  - 20.3. Usage
    - 20.3.1. Collision geometry
    - 20.3.2. MeshToBullet2
    - 20.3.3. MeshToSDF
  - 20.4. Software
  - 20.5. Raw documents
- 21. NdartsConstraint
  - 21.1. Background
    - 21.1.1. Reference & Source material
  - 21.2. Design
  - 21.3. Usage
  - 21.4. Software
    - 21.4.1. General concepts
  - 21.5. Raw documents
- 22. NdartsContact
  - 22.1. Background
    - 22.1.1. Reference & Source material
  - 22.2. Design
    - 22.2.1. Unilateral constraints
  - 22.3. Background
  - 22.4. Software
    - 22.4.1. CollisionNdarts
  - 22.5. Raw documents
- 23. NdartsFlex
  - 23.1. Background
    - 23.1.1. Reference & Source material
  - 23.2. Design
    - 23.2.1. Frame conventions
  - 23.3. Usage
    - 23.3.1. Flexible body constraints
  - 23.4. Software
    - 23.4.1. Flex ATBI recursion structure
  - 23.5. Raw documents
- 24. FModal
  - 24.1. Background
    - 24.1.1. Reference & Source material
  - 24.2. Design
    - 24.2.1. NASTRAN-to-DARTS Pipeline
    - 24.2.2. Modal Analysis
    - 24.2.3. Geometric stiffening due to inertial loads
  - 24.3. Usage
    - 24.3.1. NASTRAN-to-DARTS Pipeline
  - 24.4. Software
  - 24.5. Raw documents
- 25. Dtest
  - 25.1. Background
    - 25.1.1. Reference & Source material
  - 25.2. Design
  - 25.3. Usage
  - 25.4. Software
  - 25.5. Raw documents
  - 25.6. Sphinx documentation
    - 25.6.1. Introduction
    - 25.6.2. JPL Darts/Dshell Testing Framework
    - 25.6.3. Using the Dtest Utilities

#### 25.6.4. DUnit Testing Framework

### 26. SiteDefs

#### 26.1. Background

##### 26.1.1. Reference & Source material

#### 26.2. Design

#### 26.3. Usage

#### 26.4. Software

#### 26.5. Raw documents

#### 26.6. Sphinx documentation

##### 26.6.1. Makefiles

### 27. pyam

#### 27.1. Background

##### 27.1.1. Reference & Source material

#### 27.2. Design

#### 27.3. Usage

##### 27.3.1. Using baseline package releases

##### 27.3.2. Recipes for using `git` based `pyam` modules

#### 27.4. Software

#### 27.5. Raw documents

### *Model modules*

### 28. DshellCommonModels Dshell model library

#### 28.1. DshellCommonModels::Accelerometer Sensor Model

#### 28.2. DshellCommonModels::BallEncoder Encoder Model

#### 28.3. DshellCommonModels::BallMasking Actuator Model

#### 28.4. DshellCommonModels::CmFrameStateSensor Sensor Model

#### 28.5. DshellCommonModels::GimbalAngleEncoder Encoder Model

#### 28.6. DshellCommonModels::NodePosVelAccelSensor Sensor Model

#### 28.7. DshellCommonModels::NoisyIMU Sensor Model

#### 28.8. DshellCommonModels::NoisyNodePosVelAccelSensor Sensor Model

#### 28.9. DshellCommonModels::RateGyro Sensor Model

#### 28.10. DshellCommonModels::ReactionWheel Motor Model

#### 28.11. DshellCommonModels::SpiceFramePCIBodySync Actuator Model

#### 28.12. DshellCommonModels::SpiceFramePCRBBodySync Actuator Model

#### 28.13. DshellCommonModels::UserClock Sensor Model

### 29. GeneralSGModels Dshell model library

#### 29.1. GeneralSGModels::BallJointSpringDamper Motor Model

#### 29.2. GeneralSGModels::BearingAngle Sensor Model

#### 29.3. GeneralSGModels::DCMotorVin Motor Model

#### 29.4. GeneralSGModels::DCMotorVin2 Motor Model

#### 29.5. GeneralSGModels::ExternalSingleDofDisturbance Actuator Model

#### 29.6. GeneralSGModels::GearedPinAccel Motor Model

#### 29.7. GeneralSGModels::GearedPinAngle Motor Model

#### 29.8. GeneralSGModels::GeneralAccelGimbal Motor Model

#### 29.9. GeneralSGModels::GeneralAccelUjoint Motor Model

#### 29.10. GeneralSGModels::GeneralForceFullDofs Motor Model

#### 29.11. GeneralSGModels::GeneralForceGimbal Motor Model

#### 29.12. GeneralSGModels::GeneralForceUjoint Motor Model

#### 29.13. GeneralSGModels::GeneralizedSpringDamperMotor Motor Model

#### 29.14. GeneralSGModels::GimbalEncoder Encoder Model

#### 29.15. GeneralSGModels::JointForceTorqueSensor Sensor Model

#### 29.16. GeneralSGModels::NodeFrame2FrameUuid Sensor Model

#### 29.17. GeneralSGModels::NoisyAttitude Sensor Model

#### 29.18. GeneralSGModels::PinRate Encoder Model

#### 29.19. GeneralSGModels::PrescribedUjoint Motor Model

#### 29.20. GeneralSGModels::SecondOrderResponse Sensor Model

#### 29.21. GeneralSGModels::SignalMux Sensor Model

#### 29.22. GeneralSGModels::SingleTrapezoidalProfile Motor Model

#### 29.23. GeneralSGModels::SoftJointStop Motor Model

#### 29.24. GeneralSGModels::SpringDamper Actuator Model

#### 29.25. GeneralSGModels::SpringDamperMotor Motor Model

#### 29.26. GeneralSGModels::SpringDamperMotor6dof Motor Model

#### 29.27. GeneralSGModels::TiltVector Sensor Model

#### 29.28. GeneralSGModels::UjointSpringDamper Motor Model

### 30. GravitySGModels Dshell model library

#### 30.1. GravitySGModels::SphericalHarmonicGravity Actuator Model

### 31. RoverNavModels Dshell model library

#### 31.1. RoverNavModels::ArcTraj Flow Model

- 31.2. RoverNavModels::ControlStatus Flow Model
- 31.3. RoverNavModels::DriveTrain4x4 Sensor Model
- 31.4. RoverNavModels::DriveTrainAccel Sensor Model
- 31.5. RoverNavModels::DriveTrainSteering Flow Model
- 31.6. RoverNavModels::Locomotion Flow Model
- 31.7. RoverNavModels::NavOdometry Actuator Model
- 31.8. RoverNavModels::NavOdometry2W Actuator Model
- 31.9. RoverNavModels::RoverNavDyn Flow Model
- 31.10. RoverNavModels::RoverPosNavigation Flow Model
- 31.11. RoverNavModels::RoverPosNavigationFsm Flow Model
- 31.12. RoverNavModels::RoverTurningRadiusWayPointsNav Flow Model
- 31.13. RoverNavModels::RoverVelNavigation Flow Model
- 31.14. RoverNavModels::RoverVelNavigationFsm Flow Model
- 31.15. RoverNavModels::SimpleArcPlanner Flow Model
- 31.16. RoverNavModels::SimpleArcPlannerFsm Flow Model
- 31.17. RoverNavModels::Steering2W Actuator Model
- 31.18. RoverNavModels::Steering2WFlow Flow Model
- 31.19. RoverNavModels::Steering4W Actuator Model
- 31.20. RoverNavModels::Steering4WFlow Flow Model
- 31.21. RoverNavModels::Steering6W Actuator Model
- 31.22. RoverNavModels::Steering6WFlow Flow Model
- 31.23. RoverNavModels::SteeringStatus Flow Model
- 31.24. RoverNavModels::SwitchExample Flow Model
- 31.25. RoverNavModels::SwitchMSMExample Flow Model
- 31.26. RoverNavModels::SwitchMSMFunctorExample Flow Model
- 31.27. RoverNavModels::WheelDriveMotion Flow Model
- 31.28. RoverNavModels::WheelDriveVelocity Flow Model
- 31.29. RoverNavModels::WheelSteerDriveMotion Flow Model
- 31.30. RoverNavModels::WheelSteerDriveVelocity Flow Model
- 32. SurfaceContactModels Dshell model library
  - 32.1. SurfaceContactModels::BekkerWheelSoilContact Actuator Model
  - 32.2. SurfaceContactModels::BekkerWheelVariableSoilContact Actuator Model
  - 32.3. SurfaceContactModels::CompliantContact Actuator Model
  - 32.4. SurfaceContactModels::CompliantContact2 Actuator Model
  - 32.5. SurfaceContactModels::CompliantTerzaghi Actuator Model
  - 32.6. SurfaceContactModels::CompliantTerzaghiBaseAlt Actuator Model
  - 32.7. SurfaceContactModels::CompliantTerzaghiBaseAltVariableSoil Actuator Model
  - 32.8. SurfaceContactModels::CompliantTerzaghiCylindricalWheelAlt Actuator Model
  - 32.9. SurfaceContactModels::CompliantTerzaghiCylindricalWheelAltVariableSoil Actuator Model
  - 32.10. SurfaceContactModels::CompliantTerzaghiPad Actuator Model
  - 32.11. SurfaceContactModels::CompliantTerzaghiPadAlt Actuator Model
  - 32.12. SurfaceContactModels::CompliantTerzaghiRotatingPad Actuator Model
  - 32.13. SurfaceContactModels::ComputePlanePenetration Sensor Model
  - 32.14. SurfaceContactModels::FialaTire Actuator Model
  - 32.15. SurfaceContactModels::MagicFormulaTireHMMWV Actuator Model
  - 32.16. SurfaceContactModels::MagicFormulaTireMRZR Actuator Model
  - 32.17. SurfaceContactModels::ScmDartsModel Actuator Model
  - 32.18. SurfaceContactModels::TerrainPenAnalytic Sensor Model
  - 32.19. SurfaceContactModels::TerrainPenAnalyticBaseAlt Sensor Model
  - 32.20. SurfaceContactModels::TerrainPenAnalyticCylindricalWheelAlt Sensor Model
  - 32.21. SurfaceContactModels::TerrainPenAnalyticPad Sensor Model
  - 32.22. SurfaceContactModels::TerrainPenAnalyticPadAlt Sensor Model
  - 32.23. SurfaceContactModels::TyreContact Actuator Model
- 33. VehicleModels Dshell model library
  - 33.1. VehicleModels::FixedThruster Actuator Model
  - 33.2. VehicleModels::FuelManifold Actuator Model
  - 33.3. VehicleModels::FuelTank Actuator Model
  - 33.4. VehicleModels::FuelTankDryMassWithTableLookup Actuator Model
  - 33.5. VehicleModels::FuelTankWithTableLookup Actuator Model
  - 33.6. VehicleModels::PulsedThruster Actuator Model
  - 33.7. VehicleModels::PulsedThrusterBlowdown Actuator Model
  - 33.8. VehicleModels::SimpleTurn Actuator Model
  - 33.9. VehicleModels::ThrottledProfileThruster Actuator Model
  - 33.10. VehicleModels::ThrottledProfileThrusterWithBackPressure Actuator Model
  - 33.11. VehicleModels::ThrottledThruster Actuator Model
  - 33.12. VehicleModels::ThrottledThrusterMinimal Actuator Model
  - 33.13. VehicleModels::ThrottledThrusterWithBackPressure Actuator Model
  - 33.14. VehicleModels::ThrusterBase Actuator Model





image: ./extras/darts\_logo\_100x100.png[]



# 1. Dshell++

## 1.1. Background

### 1.1.1. Reference & Source material

- [Dshell++ Doxygen docs](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/Dshell++/html/) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/Dshell++/html/)
- [Team talks](https://dlabportal.jpl.nasa.gov/resources/darts-lab-talks/) (https://dlabportal.jpl.nasa.gov/resources/darts-lab-talks/)
- [2019, 2020 course slides](https://dlabportal.jpl.nasa.gov/documentation/coursetutorial-links/) (https://dlabportal.jpl.nasa.gov/documentation/coursetutorial-links/)
- [Sphinx docs](https://dartslab.jpl.nasa.gov/dlabdocs/) (https://dartslab.jpl.nasa.gov/dlabdocs/)
- [Jupyter notebooks](https://dlabnotebooks.jpl.nasa.gov/hub/login) (https://dlabnotebooks.jpl.nasa.gov/hub/login)
- [DARTS Lab QA site](https://dartslab.jpl.nasa.gov/qa/) (https://dartslab.jpl.nasa.gov/qa/)
- [Model/assemblies style guide](https://dartslab.jpl.nasa.gov/technotes/Modeling/DshellModeling.pdf) (https://dartslab.jpl.nasa.gov/technotes/Modeling/DshellModeling.pdf)
- Release notes appendices
- [DSENDS manual](https://dartslab.jpl.nasa.gov/technotes/Modeling/2010-dsendsManual.pdf) (https://dartslab.jpl.nasa.gov/technotes/Modeling/2010-dsendsManual.pdf)
- [DSENDS paper](https://dartslab/References/pdf/2016-aiaa-dsends.pdf) (https://dartslab/References/pdf/2016-aiaa-dsends.pdf)

## 1.2. Design

### 1.2.1. Dshell Model Design (aka Models Style Guide Draft)

#### 1.2.1.1. Dshell Model Design and Implementation (Draft)

##### 1.2.1.1.1. Introduction



TBD: Review/scrub use of ContinuousModel vs Model vs Modelthroughout to ensure proper uses.



TBD: Review/scrub use of 'flow ins' vs 'flowIns', etc, throughout.

The Dshell simulation framework is for the development of physics and dynamics models of articulated, multi-degree of freedom vehicles including actuator/sensor devices and interactions with the environment primary for control and autonomy applications such as shown in Figure Vehicle simulations. Figures [DSENDS simulations, ROAMS simulations, and Rotorcraft simulations] illustrate the key sub-systems involved in the simulation of such systems for aerospace and ground vehicle applications. An important goal of the Dshell architecture is to:

- provide support for key simulation physics models needed for realistic physics modeling
- allow the reuse of component models from one simulation to another
- manage the complexity and variety of simulation models
- provide high-performance computational dynamics engines for high-fidelity dynamics.
- build in range of simulation services (eg. data logging, introspection) that are typically needed for simulation use
- ...

Examples of vehicle dynamics simulations using Dshell.

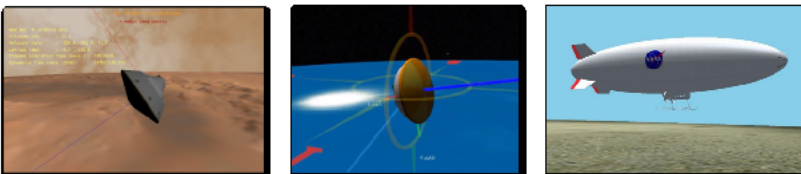
**Rovers**



**Spacecraft**



**Aerial**



TBD: Add here discussion of why not Simulink, and need for DARTS.\*

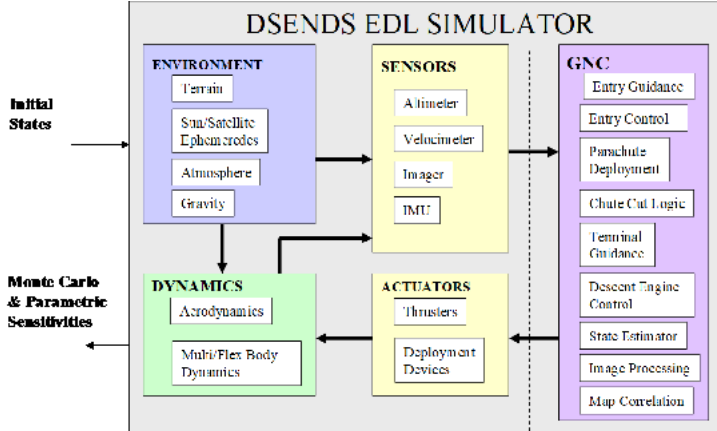


TBD: Add brief discussion of why use DARTS instead of ADAMS.\*

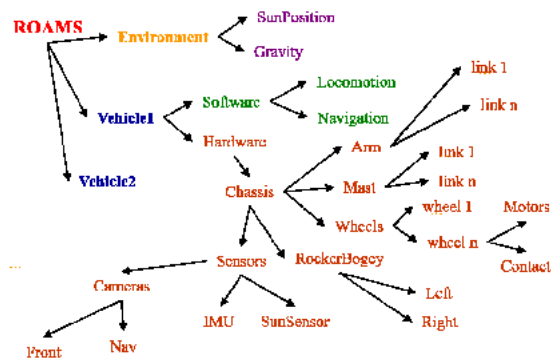
A key design difference from other simulation tools (eg. Simulink) is the emphasis and focus on simulating vehicle multibody dynamics models within the architecture. Due to the complexity of multibody dynamics goals, our goal is to build in a generic multibody dynamics engine within the architecture so that it can be used for variety of application projects and scenarios. The more traditional route is to leave it to the user to create the multibody dynamics component block on their own and integrate it in as a block within the dataflow. While simplifying the dataflow implementation, this approach has significant limitations:

- it puts the burden of developing the complex multibody vehicle equations of motion solver entirely on the user (eg. rigid/flex, tree/closed-chain, collision/contact dynamics)
- it is not possible to reuse dynamics model from one simulation to another
- it makes it difficult to handle run-time structural changes (eg. body attach/detach) changes in the dynamics model.
- Manually handle copious number of interfaces between the device models and the multibody dynamics model.
- ..

Example of lander system and its components simulated by Dshell.

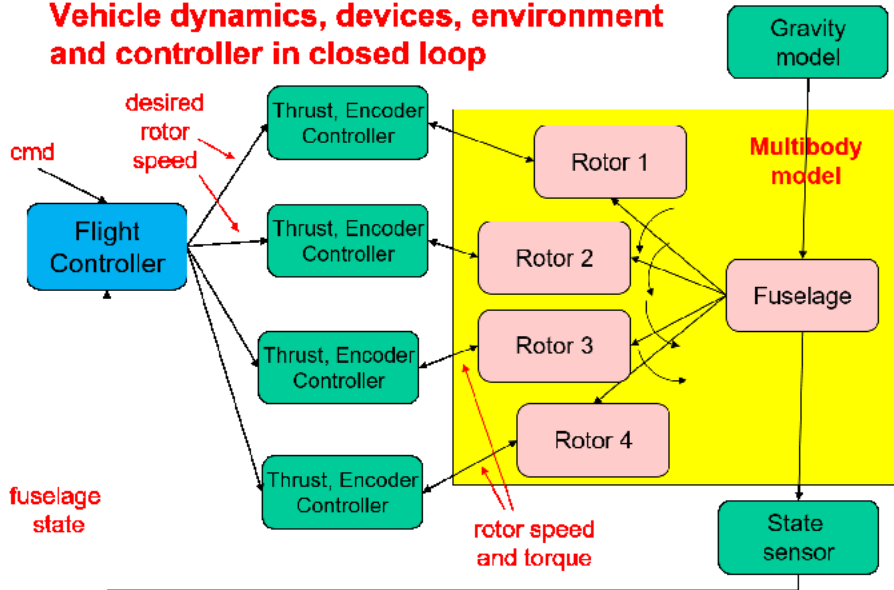


Example of ground vehicle system and its components simulated by Dshell



Example rotorcraft system and its components simulated by Dshell

## Vehicle dynamics, devices, environment and controller in closed loop



This document describes design details for the Dshell simulation framework that includes a description of the underlying mathematical problem and the important architectural elements related to model development and integration. While this document does touch upon aspects of the software, it is not meant to serve as software documentation. We recommend that the full document be read for readers unfamiliar with the Dshell framework. For readers who have been using the Dshell software, we recommend the Simulation time stepping section which describes various topics pertinent to simulation time-stepping. The Dshell Simulation Models section is especially relevant to model developers. The Model implementation guidelines section describes recommended guidelines for the overall simulation development as well as model implementation. The Dataflow model calling order and sorting section is devoted to the specific topic of component model sorting and calling order and handling of loops within model dataflows.

### 1.2.1.1.1.1. Simulating dynamical systems

For a model of a dynamical system, the continuous states are governed by a differential equation, and their time evolution is carried out using numerical integrators. The general form of the dynamical system equations is:

$$\begin{aligned}
 Y(t) &= G(X(t), \tilde{U}(t), t) && \text{outputs} \\
 X_d(t + \Delta_t) &= H(X(t), \tilde{U}(t), t, \Delta_t) && \text{discrete state update} \\
 \dot{X}_c(t) &= F(X(t), \tilde{U}(t), t) && \text{continuous state derivative}
 \end{aligned}$$

The following represents the `output` equation part within the overall system dynamics:

$$Y(t) = G(X(t), \tilde{U}(t), t) \quad \text{outputs}$$

The following represents the `discrete state` dynamics equation part within the overall system dynamics:

$$X_d(t + \Delta_t) = H(X(t), \tilde{U}(t), t, \Delta_t) \quad \text{discrete state update}$$

The following represents the `continuous state` dynamics equation part within the overall system dynamics:

$$\dot{X}_c(t) = F(X(t), \tilde{U}(t), t) \quad \text{continuous state derivative}$$



**TBD:** Make a more rigorous list explaining what each term is? Should we explain what  $\Delta_t$  is?\*

The overall states  $X(t)$  of the model are a combination of the continuous and discrete states of the system, i.e.,  $X(t) = [X_c(t), X_d(t)]$ . Some points to note are:

- A dynamical system is said to not have **feed-thru** if the  $Y(t)$  outputs are not directly affected by the  $\tilde{U}(t)$  inputs, i.e. if  $G(X(t), \tilde{U}(t), t)$  is in fact just  $G(X(t), t)$ . Otherwise the system is said to have feed-thru.
- The  $Y(t)$  output is not shown as being a function of  $\dot{X}_c(t)$ , i.e.  $\dot{X}_c(t)$  is not an argument of  $G(\cdot)$ . Such a dependency can be accommodated by redefining the  $G(\cdot)$  function to include  $F(\cdot)$  for the state derivative portion, so that the new  $G(\cdot)$  remains a function of just the state, the inputs and the current time. Note however that this might introduce feed-thru into the system.



**TBD:** Add comment here that in most typical physical systems  $\dot{X}_c(t)$  is not a function of  $G(\cdot)$ ???



**TBD:** Add a section (near) here to overview the model types, model the inheritance tree, etc, to give users a bit more background.

### 1.2.1.1.1.2. Simulation framework requirements

The Dshell simulation framework is meant to be general, and to support a wide range of use cases and models. This means that a simple timing system, where the simulation is updated using a constant time step is often inadequate.



**TBD:** Do we really need to refer to constant step size? If so, then why is it inadequate? Beyond state propagation, there are several additional capabilities that are required in real-life simulations. These are described below.

#### 1.2.1.1.2.1. Decomposition into component models

Instead of implementing monolithic blocks to implement the  $F(\cdot)$  etc functions, simulations (including Dshell) often use a collection of inter-connected component dynamical system models that individually manage slices of the overall state vector, and implement parts of the  $F(\cdot)$ ,  $G(\cdot)$  etc methods to implement the system dynamics in Eq 2. Such a decomposition is only possible when there is loose dynamical coupling across components of the system, and can have the benefit of increasing the simulation modularity and the implementation into reusable components. Use of such decompositions however requires care in the proper sequencing of the computations so that the dependencies and coupling between the components is handled correctly for replicating the desired dynamics.



**TBD:** Should we add a sentence telling that the user never needs to construct the full state themselves. It is done automatically behind the scenes so that stock integrators can be used.



**TBD:** Add a figure to show the decomposition into a dataflow with component models.

#### 1.2.1.1.2.2. Multi-rate simulations

We need to support models that *require* a specific timing: for instance if we model an IMU that runs at 100Hz, then this model *must* run at exactly 100Hz, regardless of the normal simulation timestep. We support this case with **multi-rate models** (see the Multi-rate models section).

#### 1.2.1.1.2.3. Timed event handling

The user might want to get control at a specific point in time to (for instance) log data or peek/poke the simulation. These time points may not align with our default simulation timestep. We support this case with **timed events** (see the Events handling section).

#### 1.2.1.1.2.4. Zero-crossing detection

The user might want to get control of the simulation when something happens in the simulated world (and we don't beforehand know the exact time when this happens). Example: we have a falling body and we want to deploy a parachute when we get to some predetermined height. This time-point is unlikely to end up exactly at a normal simulation step boundary. We support this case with **zero-crossing callbacks** (see the {Dshellpp\_sgstepvalidation\_section\_uri[Zero-crossing events] section).

#### 1.2.1.1.2.5. Masking selected continuous coordinates



**TBD:** Add description\*

#### 1.2.1.1.2.6. Handling structural changes to the model



**TBD:** Add description - multibody changes, to the dataflow\*

#### 1.2.1.1.2.7. Support multiple numerical integration schemes



**TBD:** Add description - support for different types of integrators, fixed/variable step, different schemas, implicit etc

#### 1.2.1.1.3. Organization and goals

The purpose of this note to clearly define the rationale and guidelines for implementing and sequencing the DARTS and ContinuousModel methods so that the overall system dynamics is modeled correctly in Dshell simulations. The specific drivers are:

- To rigorously define ContinuousModel design principles in order to avoid incorrect implementation.
- Scrub existing Models to bring them into compliance.
- Tighten up the ContinuousModel method calling sequence to avoid one call delays for updating flow inputs when there are topological loops.
- Get rid of the need for the `step(0)` calls.



**TBD:** Should we explain `step(0)`?

- Reduce the need for ContinuousModel breaks by requiring them only for algebraic loops, and not all topological loops.

The topics addressed here include:



**TBD:** We may not have covered enough background for readers to understand these issues.

- the notion of continuous and discrete dynamical states
- the role and content of the individual Model methods
- the decomposition of the overall dynamics into component dynamics models
- the sequencing of the DARTS and Model calls
- the concept of granularity for tailoring Model flow out computations
- the notion of topological and algebraic loops and feed-thru for DARTS and Models models
- the use of model breaks for handling algebraic loops
- integrating in logic via an FSM



**TBD:** FSM is used before it is explained. We should at least spell out the acronym.

- style guide for implementing ContinuousModels
- plan for bringing Model implementations into compliance

### 1.2.1.1.2. Simulation time stepping

#### 1.2.1.1.2.1. Multi-rate models

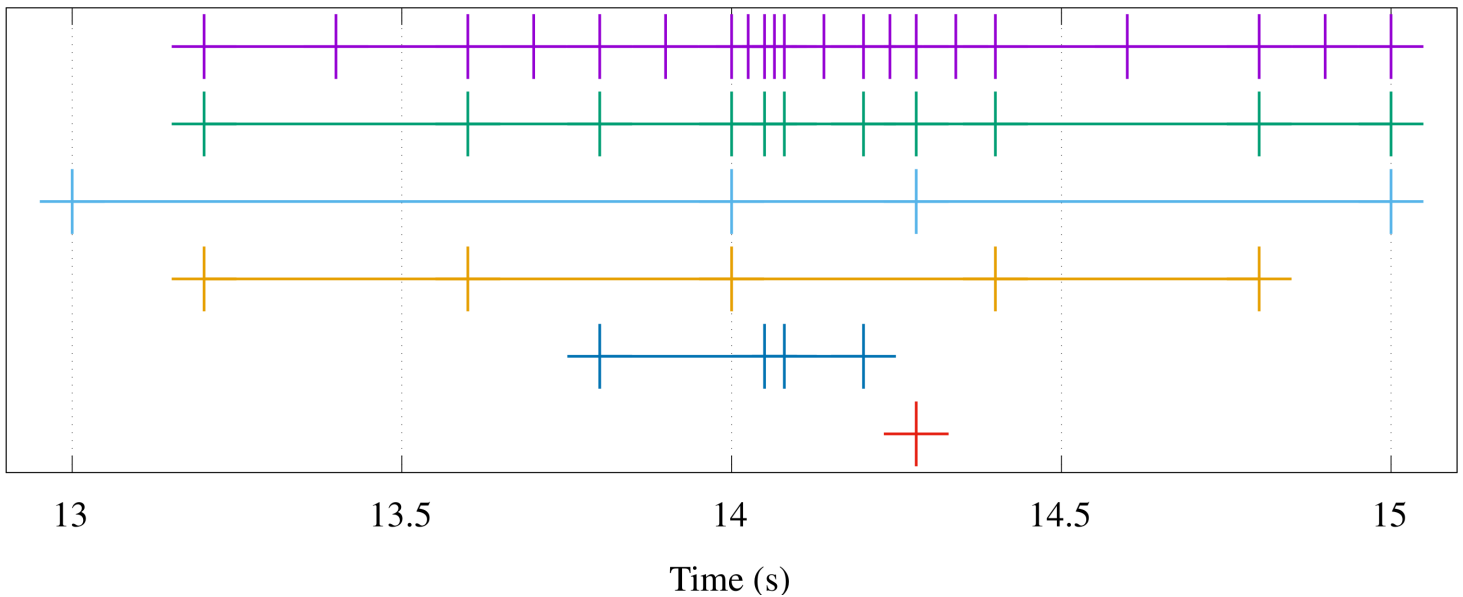
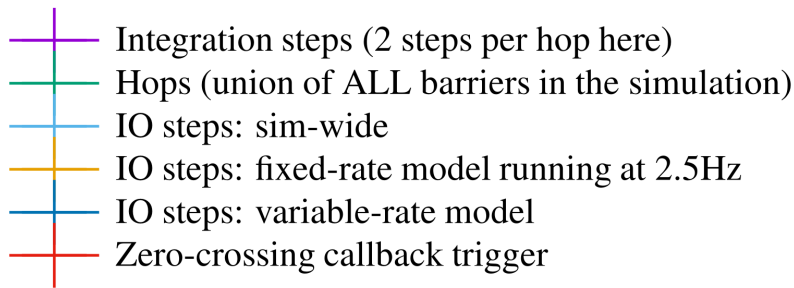
In practice, most models do not have specific calling frequency requirements, and for convenience Dshell uses the simulation's default model period for such a model's I/O period (a.k.a, the model's step size). Within Dshell however each Model in principle is a **multi-rate** model, i.e. each model can specify its own individual I/O step size value based on its needs. Common examples of Models with custom time-stepping needs are sensor hardware models (eg. cameras, IMUs) that operate at fixed rates, or models that encapsulate control software running at a specific frame rate. Often the I/O step value corresponds to a fixed rate, but in general the I/O step boundary can vary arbitrarily from step to step for each model. A model takes charge of specifying custom I/O step sizes by registering an I/O step callback method with the simulation to determine the step size value for the model. See the Registering Model multi-rate callbacks section for implementation details on using multi-rate capability.

#### 1.2.1.1.2.2. Simulation hops

As discussed above, each model defines its own I/O step-size. The simulation inserts the start and end I/O step events for every model on a common timeline. In addition, timed events can be registered that are inserted along the same time common timeline. The model I/O steps and timed events act as **barriers** such that the simulation proceeds in time by propagating the full system state from barrier to barrier and processing the pertinent functions on reaching a barrier. The time interval between successive barriers is referred to as a simulation **hop**. The simulation takes a hop to advance the state to the next barrier; calls methods associated with the barrier; and then proceeds to the next hop and so on. Note that the hop time interval size is determined by the spacing between the barriers on the simulation time-line and can vary from hop to hop. **We emphasize that hops are defined simulation-wide and thus all models see the same hop.**

The following example illustrates the various timing intervals. In this example, the simulation contains a variety of multi-rate models; some use the sim's default model period, another is registered with a different fixed rate, and another uses a multi-rate callback to create a non-uniform IO period for the associated model.

*The barriers inserted by different timing event types.*



**TBD:** This figure needs a caption and is hard to follow. We also need to get rid of the simulation-level I/O steps in this figure. The line widths and colors need to be adjusted so that they are more readable.

After a simulation hop has been completed, the simulation calls the end I/O step actions for all the models whose I/O step ends at that simulation time instant. The registered I/O step callback is evaluated for each of these models and their next I/O step time instants are added to the common timeline. Any events scheduled for this time instant are also processed. Once this is done, the start I/O step method is called for this set of models, and the next simulation hop is initiated.

#### 1.2.1.1.2.3. Simulation time propagation

To advance the simulation time, the simulation moves from hop to hop. It does this until it hits a special callback called a stop callback. When one of these stop callbacks returns true, then the simulation stops advancing. There are many convenience methods on the simulation object such as `advanceTimeBy`, `advanceTimeTo`, and `step` that will register one of these callbacks for the user based on the given arguments and then advance the simulation to the specified time.

#### 1.2.1.1.2.4. Simulation integration step



The propagation of the *continuous states* in the system across a hop time interval is carried out by a numerical integrator. For convenience of use with fixed-step numerical integrators, the Dshell simulation object provides the option of specifying the number of integration sub-steps that the integrator should take during the hop. Each integration sub-step is referred to as an **integration step**. When the number of integration sub-steps is 1, the integration sub-step size is the same as the hop size. An *integration step* thus is the smallest time quantum in the simulation. The only thing that happens within an integration step is state derivative evaluations carried out by the numerical integrator: for instance RK4 samples the state derivative at the beginning, the middle and the end of an integration step. These are mostly an internal detail, as far as the models and users are concerned, since users can only intervene at the barrier events, i.e. in between hops.



TBD: Do something with the material below.

An *IO-step* is the smallest step in time a *specific* model cares about. **This varies from model to model**. A model's IO step consists of some number of hops, the exact number and size of hops being set by other parts of the simulation.

By default, a model inherits the sim's default model period as its preferred IO-step. However, one can modify this and register a different preferred IO-step timing. We make no assumptions about these. They can be periodic or sporadic; these can recur infinitely, or end at some point in time.

Registering a different preferred IO-step happens in two ways:

- A *fixed-rate* model is a kind of *multi-rate* model that is triggered periodically at some arbitrary fixed rate, recurring forever.
- A *variable-rate* model is another kind of *multi-rate* model. These are triggered sporadically, at arbitrary model-specified points in time.

#### 1.2.1.1.2.5. Events handling



TBD: Add description of timed events - logging, fsms, visualization updates, strip charts etc

#### 1.2.1.1.2.6. Zero-crossing events

Unlike Dshell events described in the Events handling section that have explicitly defined trigger times, we do not look at events whose trigger times are implicitly defined.



TBD: Review the last sentence.

The barriers in the above example time-line include additional non-model specific simulation events explicitly registered by the user, and one from a **zero-crossing** event. A zero-crossing event is an event (or barrier) whose timing is not explicitly known, but instead implicitly defined when a specified condition on the the system state is satisfied. An example of a zero-crossing event is that of a lander spacecraft coming into contact with the ground, i.e., when the altitude state value reaches zero. The exact time of the touchdown is not known a-priori. The Dshell simulation allows the user to register multiple such zero-crossing condition callbacks that the simulation then uses to stop at the exact times when a zero-crossing condition is satisfied. In practice, when a simulation hop over-steps a zero-crossing condition, the hop is repeated and shrunk in an iterative loop to terminate precisely at the time instant when the zero-crossing condition is satisfied. In these situations, the duration of a hop can be smaller than the hops defined by the explicit events time-line.

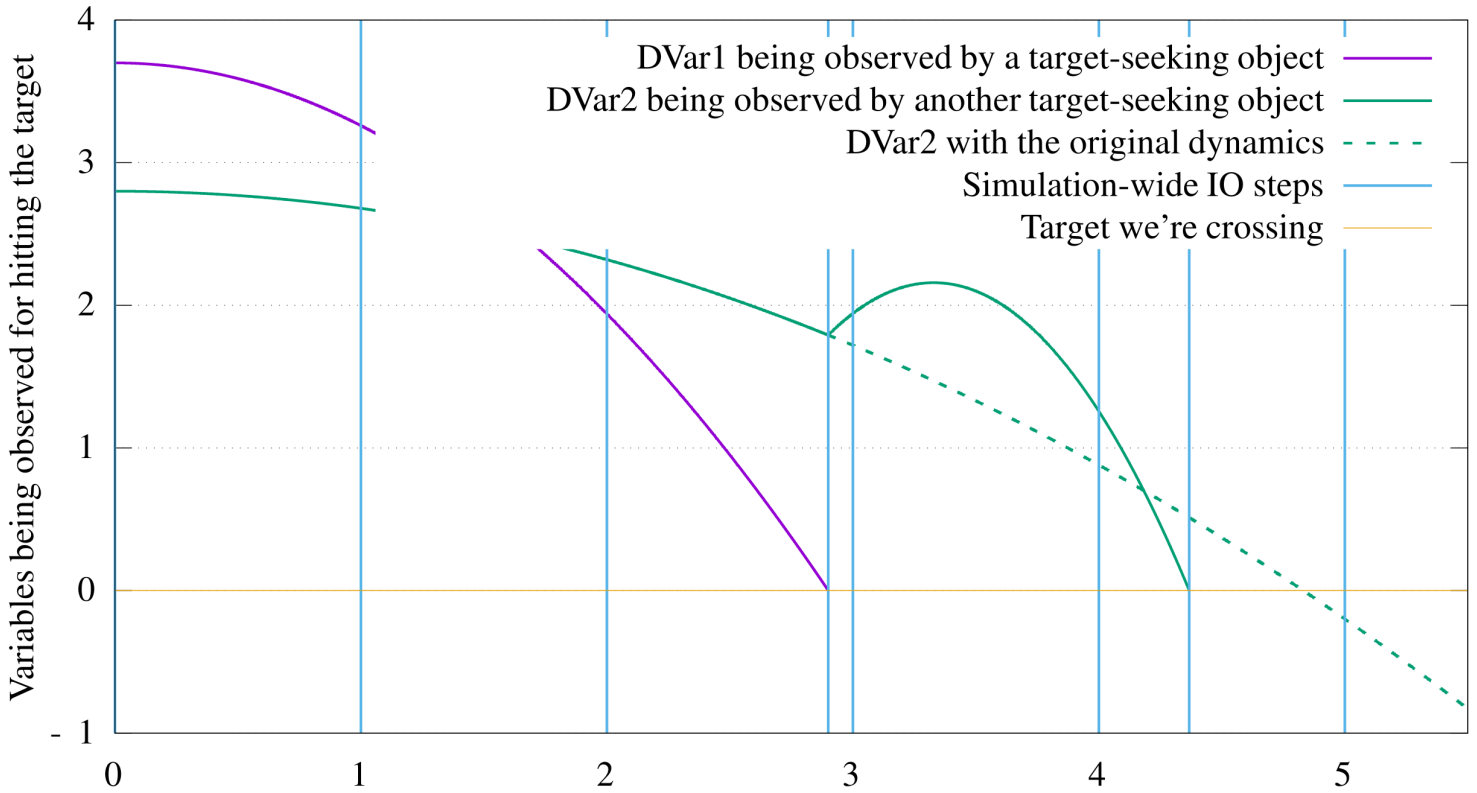
One can register callbacks corresponding to *zero-crossing conditions* with the simulation. A zero-crossing callback has the behavior of a comparison function with return values indicating where the the condition has not been met, has been met, or has been overstepped. While executing a simulation hop, the Dshell simulation monitors the status of all the registered zero-crossing conditions. None of the conditions will be satisfied at the beginning of a hop. After propagating the simulation state across the hop, the simulation checks each of the conditions. If none of the condition has yet been met, the new state is accepted and the hop is terminated.



TBD: Explain why no conditions will be satisfied at beginning of a hop?

However, if the simulation has over-stepped one or more of the conditions, the simulation rejects the new step, and embarks on an iterative procedure to find the smallest hop interval where at least one of the conditions has been met, and none have been overstepped. Once the hop interval is determined, the zero-crossing condition that has been satisfied is removed from the list of zero-crossing conditions, the new simulation step is accepted and the hop is terminated at this time instant. Zero-crossing conditions and roll-back implementation section contains more details on the specific software implementation of the zero-crossing capability. Example:

*Trajectory from a Dshell simulation involving zero-crossing detection.*



Let's say we have a simple simulation with these signal connections:



Figure 1. A simple model data flow

The simulation-step loop would then look like this:

- Loop
  1. Flow model 0 `startHop()`
  2. State propagator flow model `startHop()` to invoke the integrator
  3. Flow model 1 `startHop()`
  4. Evaluate all the step-validation callbacks
  5. if(*none* of the step-validation callback has overshot)
    - We're done! Break out of the loop
  6. else
    - restore *all* previous state: continuous and discrete
    - Adjust step size
    - Loop to retry the step

Note that the zero-crossing callbacks are completely generic, i.e. they aren't directly associated with any particular model. As far as the simulation is concerned, it's just calling a function; the simulation does not know or care about what data this function looks at.

In summary, the simulation time barriers determining the hop intervals are defined via



TBD: Complete/fix the previous sentence!

- Model-specific (i.e. multi-rate) IO-step times
- Registered timed events
- Registered zero-crossing conditions

#### 1.2.1.1.2.7. Overview of methods used to step the simulation

The following diagram gives a brief overview of the methods in the `Simulation` class that are used to move the simulation forward in time.

```

/'
Define a macro that stores the PlantUML theme.

To use, in your asciidoc wiki page write:

include::_auto_global/plantUmlTheme.asciidoc[]

For example,
```plantuml
include::_auto_global/plantUmlTheme.asciidoc[]
@startuml
...
@enduml
```
'/

!theme spacelab
skinparam ArrowFontColor #446e9b
@startuml
[*] --> _run

state _run {
  _advanceOverExplicitHopInterval : Calls timed events.
  _advanceOverExplicitHopInterval : Calls appropriate model events.
  _advanceOverExplicitHopInterval : Calculates next explicit hop time.

  state _advanceOverExplicitHopInterval {
    state _advanceHop {
      state _advanceState {
        state _advanceContinuousStates {
        }
        _advanceContinuousStates: Calls appropriate model events
        _advanceContinuousStates: Calls the integrator

        [*] --> [*] : No continuous states
        [*] --> _advanceContinuousStates : Has continuous states
        _advanceContinuousStates --> [*]
      }
    }
    _advanceHop: Calls appropriate model events.
    _advanceHop: Handles step validation callbacks.

  }
}
_run --> [*] : Stop callback triggered
_run --> _run : No stop callback triggered

@enduml

```

#### 1.2.1.1.2.8. References

- See [this](https://dartsgitlab-internal.jpl.nasa.gov/darts-lab/jpl-internal/all-access/development/simulation-framework/dshellpp/-/issues/23) (https://dartsgitlab-internal.jpl.nasa.gov/darts-lab/jpl-internal/all-access/development/simulation-framework/dshellpp/-/issues/23) GitLab issue for the most recent simulation time step refactor.

#### 1.2.1.1.3. Dshell Simulation Models

Within Dshell, we decompose the vehicle dynamical model described by Eq 2 into a DARTS multibody dynamics model, and a dataflow of component Models that interact with the multibody dynamics and each other through flow input/output connections.

##### 1.2.1.1.3.1. DARTS multibody dynamics model

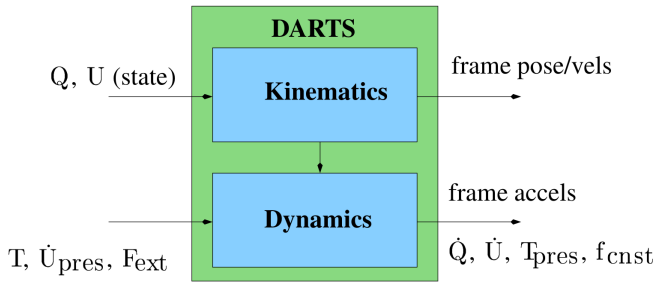
The state vector for the DARTS multibody model consists of the  $[Q(t), U(t)]$  configuration and the corresponding velocity coordinates. The DARTS model has no discrete states. As illustrated in Figure DARTS components the DARTS model includes two computational sub-models:

- **Kinematics:** This layer computes the pose and velocities of all frames including bodies, hinges and nodes in the mechanism based on the input state  $[Q(t), U(t)]$ . Since the state is normally managed by the integrator and there are no other inputs, this component normally does not have feed-thru. The only time this component has feed-thru is when some of elements of the  $[Q(t), U(t)]$  arrays are **masked**, i.e. when these values are set by a DartsModels instead of by the the integrator.
- **Dynamics:** This layer solves the multibody dynamics equations of motion to compute accelerations, i.e. the  $\dot{Q}(t), \dot{U}(t)$  state derivatives. The *dynamics inputs* are normally the generalized forces  $T(t)$  and external forces  $F_{ext}(t)$ . The equations of motion take the form:

$$\begin{aligned} \dot{Q}(t) &= G_M(Q(t), U(t)) && \text{generalized coordinate derivatives} \\ \dot{U}(t) &= F_M(Q(t), U(t), T(t), F_{ext}(t), t) && \text{accelerations computations} \end{aligned}$$

There are situations when part of the DARTS state derivative  $\dot{U}(t)$  may be **prescribed**, i.e. be externally specified by the DARTS inputs  $y_m(t)$ . All that this means is that these elements of the state derivative are directly defined by the inputs and this in of itself does not imply feed-thru for DARTS.

*The kinematics and dynamics components of the DARTS multibody dynamics model.*



There are no pre-defined outputs from DARTS. Models typically use the DARTS (or frames) API to obtain DARTS state dependent information and accelerations. Generally speaking, the DARTS model *does not have feed-thru* except for the following special cases:

- When there are Models whose inputs from DARTS (denoted below as  $u_m(t)$ ) depend on the DARTS multibody accelerations  $\dot{U}(t)$ , such as for accelerometer and IMU sensor models, and for friction models that depend on inter-body constraint forces. When such ContinuousModels exist in the simulation, the outputs of DARTS are directly dependent on the accelerations which in turn are directly dependent on DARTS inputs, and thus the DARTS model has feed-thru. We emphasize that the acceleration dependency, and consequent feed-thru nature of DARTS, is a use-case context dependent property, rather than an intrinsic property of DARTS. Inter-body constraint forces are also computed by DARTS, and if these are used by other models then DARTS potentially has feed thru because such forces directly depend on the input prescribed generalized accelerations and forces.
- There are situations when portions of the  $[Q(t), U(t)]$  multibody coordinates may be defined directly by inputs and not the dynamics (eg. for trim aerodynamics, or high speed engine gimbal control) and thus be **masked** from the integrator. Since these masked coordinates directly affect the DARTS outputs, the use of such masked coordinates introduces feed-thru into the DARTS model.
- While coordinates change with the simulation, parameters defining the geometry, kinematics and mass properties of the DARTS model bodies in most cases remain constant during a simulation. However there are situations where these parameters may change during a simulation, such as mass properties of a body may change due to fuel consumption, gas in/out flow for balloons. Such updates DARTS model parameters also introduce feed thru into the DARTS model.

#### 1.2.1.1.3.2. Dshell Models

As illustrated in Figure Inter-connected models, component Dshell models are connected to each other in the dataflow via Signals.

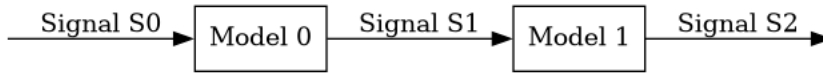
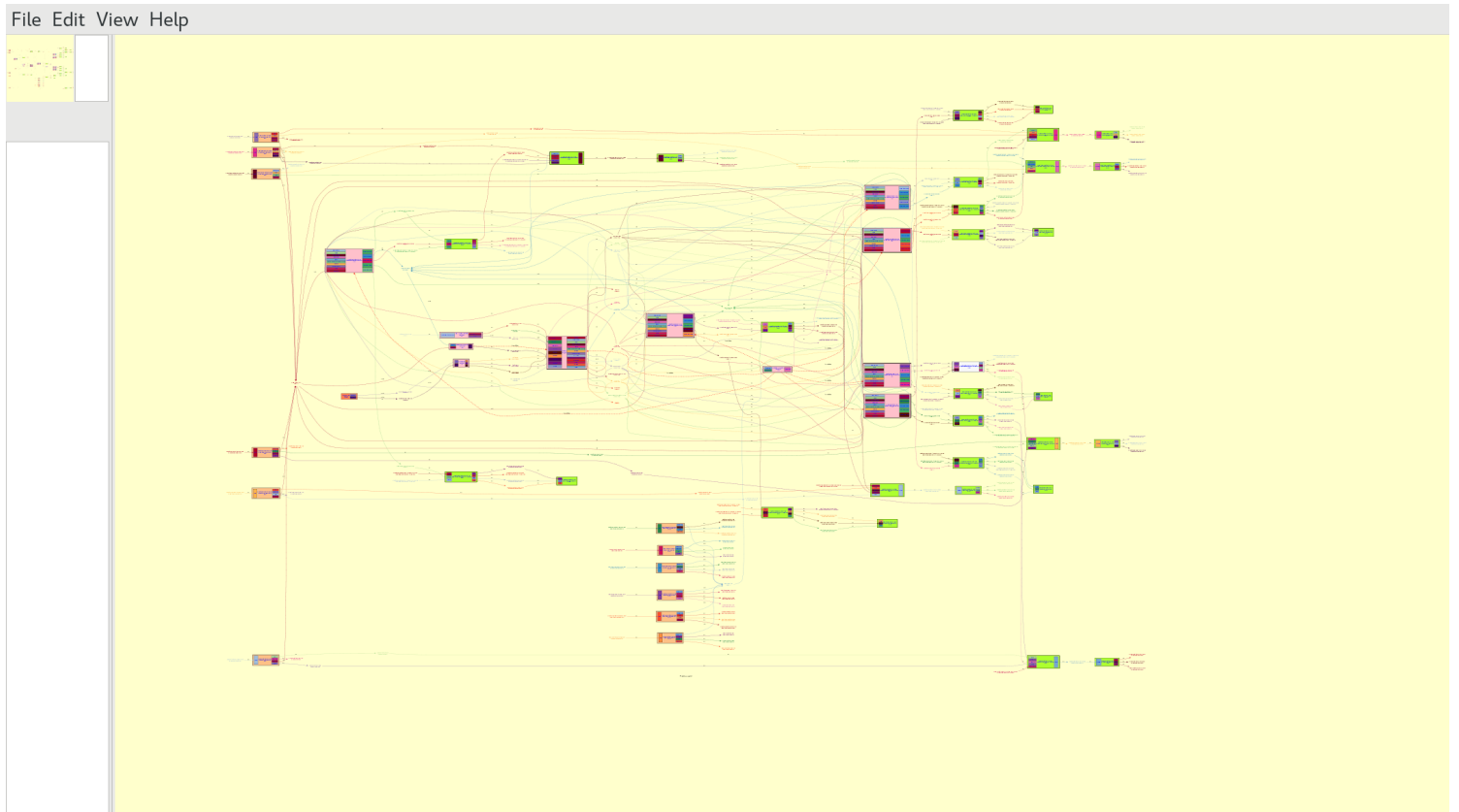


Figure 2. Dataflow with models connected via signals

These signals are *memory-less* (have no history), and their values depend *only* on the *current* values of the source model's state and input signals. Furthermore, the memory for the inputs  $\bar{U}$  and outputs  $Y$  is shared, so the signal update *must* happen first in order to update the upstream signals before the downstream ones. Figure Dataflow

#### A typical dataflow from a full up vehicle simulation]

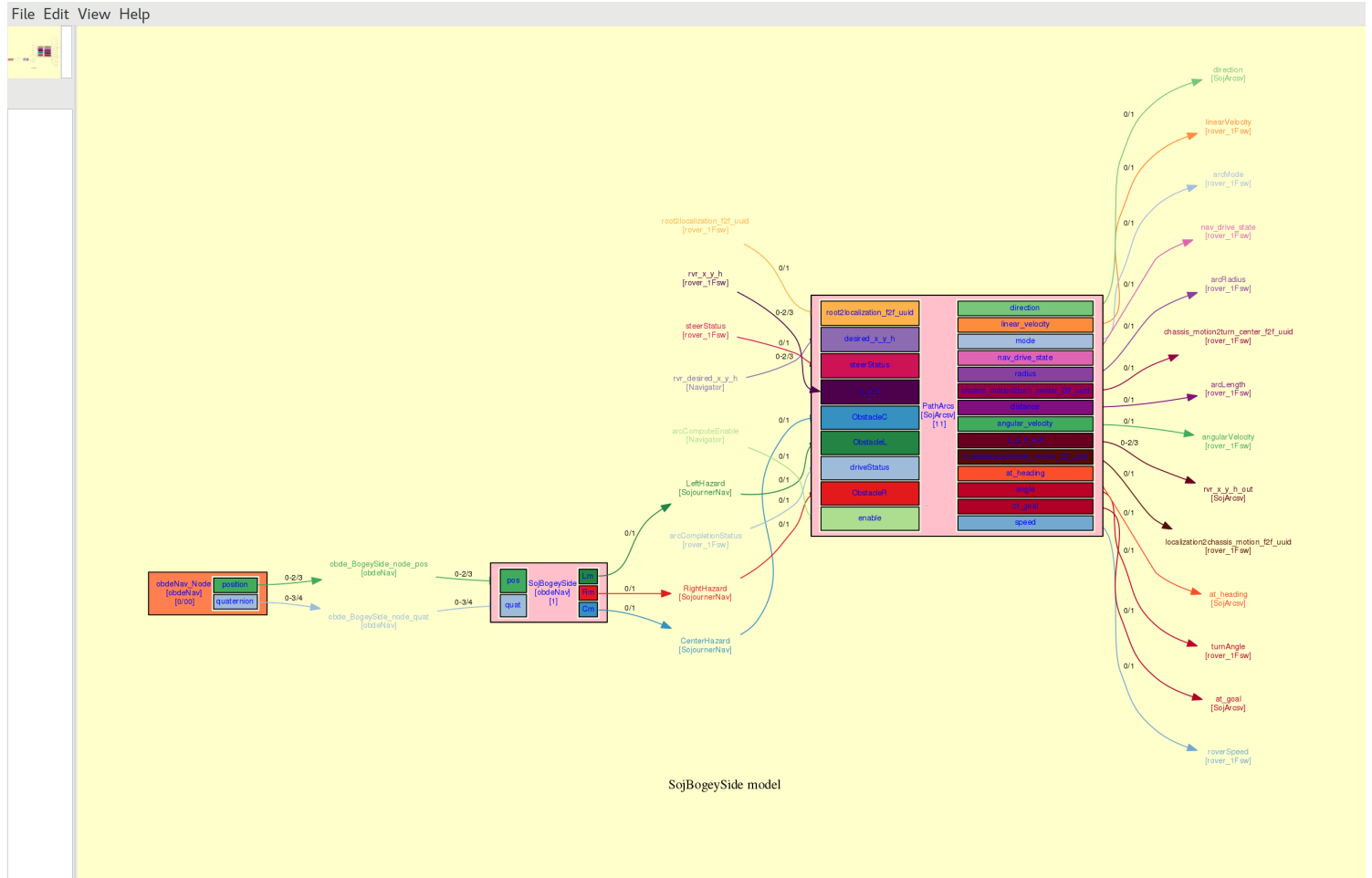




TBD: This figure needs to be cropped and enlarged to make it more readable.

shows an example dataflow from a realistic vehicle modeling simulation where the number of component models and interconnecting signals can be in the several tens and even hundreds. Figure Dataflow fragment shows a more detailed view of a fragment in the Figure Dataflow dataflow.

*A detailed view of a fragment of the dataflow*



TBD: This figure needs to be cropped and enlarged to make it more readable.

We next take a more detailed look at the different types of models that can be used in such Dshell simulation dataflows.

A general component Dshell Model model (shown in Figure DartsModel) has continuous states,  $x_c(t)$ , discrete states  $x_d(t)$ , flow inputs  $u_f(t)$ , flow outputs  $y_f(t)$ , inputs from DARTS multibody  $u_m(Q(t), U(t), \dot{U}(t), t)$  and outputs that are inputs into DARTS  $y_m(t)$ . Its overall state  $x(t)$  is a combination of the discrete and continuous states  $x(t) = [x_c(t), x_d(t)]$ , and the overall inputs  $u(t)$  are a combination of the flow inputs and the inputs from DARTS, i.e.  $u(t) = [u_f(t), u_m(t)]$ . The general representation of the dynamics of a ContinuousModel has the following form with  $f, g$  and  $h$  representing mapping functions:

$$\begin{aligned}
 y_f(t) &= g_f(x(t), u(t), t) && \text{update flowOut outputs} \\
 x_d(t + \Delta_t) &= h(x(t), u(t), t, \Delta_t) && \text{discrete state update} \\
 y_m(t) &= g_m(x(t), u(t), t) && \text{inputs into DARTS multibody model via nodes/hinges} \\
 \dot{x}_c(t) &= f(x(t), u(t), t) && \text{continuous state derivative}
 \end{aligned}$$

We break out the above equations into the update flowOut outputs update equation.

$$y_f(t) = g_f(x(t), u(t), t) \quad \text{update flowOut outputs}$$

the discrete states update equation,

$$x_d(t + \Delta_t) = h(x(t), u(t), t, \Delta_t) \quad \text{discrete state update}$$

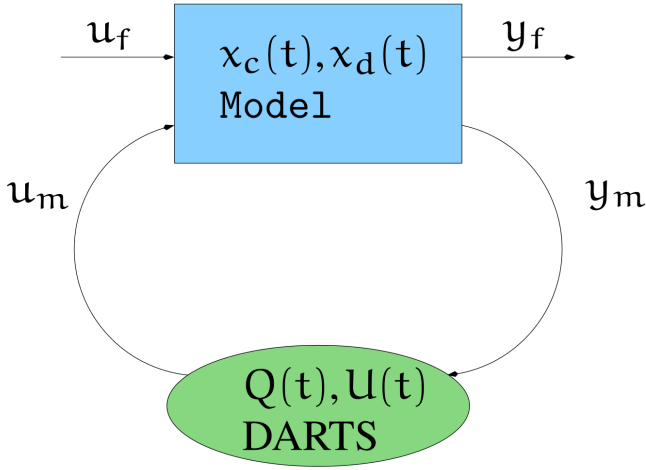
the inputs updates for the DARTS multibody model via nodes/hinges equation,

$$y_m(t) = g_m(x(t), u(t), t) \quad \text{inputs into DARTS multibody model via nodes/hinges}$$

and the continuous state derivative equation

$$\dot{x}_c(t) = f(x(t), u(t), t) \quad \text{continuous state derivative}$$

A general Model instance with its inputs and outputs.



Note that the  $u_m(Q(t), U(t), \dot{U}(t), t)$  (or just  $u_m(t)$  for short) inputs from DARTS are in general functions of the DARTS state  $[Q(t), U(t)]$  and the  $\dot{U}(t)$  DARTS state derivative (and dependent quantities such as inter-body forces).

### 1.2.1.1.3.3. Model methods

Table Model methods describes the various model methods and their role in evaluating the equation in Eq 6 within simulations when advancing the system step across an I/O step interval. Depending on the setting of nIntegrationSteps' value, this advancement may involve one or more numerical integration steps. Note that we can compute  $x_d(t + \Delta_t)$  in startHop when its value does not depend on the values of  $x_c(t), u(t)$  or  $t$  over the  $(t, t + \Delta_t)$  time interval. Otherwise compute it in startIntegrationStep.

Table 1. Dshell Model methods and their purpose.

| Method   | Computation  | Description   |
|--|--|---|
| <b>All Dshell Models</b>   |  |   |
| updateFlowOuts(t)  | Eq 7: $y_f(t) = g_f(x(t), u(t), t)$                    | compute flow outputs; frequency depends on flowout granularity for each model   |
| startIoStep(t_i, t_{i+1})  |  | called at the <b>start</b> of the (possibly multi-rate) I/O step for the model. <b>IO-step methods are called at model-specific multi-rate intervals only so calls can vary across models</b> |
| startHop(t_i, t_{i+1})   | Eq 8: $x_d(t + \Delta_t) = h(x(t), u(t), t, \Delta_t)$ | propagate discrete states across hop interval. <b>This is sim-wide; called in unison for all models</b>   |
| endIoStep(t_i, t_{i+1})  |  | called at the <b>end</b> of the (possibly multi-rate) I/O step for the model. <b>IO-step methods are called at model-specific multi-rate intervals only so calls can vary across models</b>   |
| <b>ContinuousModels (These are called in unison sim-wide for all models)</b> |  |   |
| startIntegrationStep(t_j, t_j + \Delta_t)                                    | Eq 8: $x_d(t + \delta_t) = h(x(t), u(t), t, \delta_t)$ | propagate discrete states across integration sub-step (if not done in startHop)//, see footnote #ftn:discretestate)   |
| preDeriv(t_d)  | Eq 9: $y_m(t) = g_m(x(t), u(t), t)$                    | compute inputs for DARTS multibody dynamics.  |
| postDeriv((t_d)  | Eq 10: $\dot{x}_c(t) = f(x(t), u(t), t)$               | compute model's continuous state derivatives; can also be done in preDeriv(t) if no dependency on multibody $\dot{U}(t)$ . startIntegrationStep).   |
| endIntegrationStep(t_j, t_j + \Delta_t)                                      |  | any actions to be taken at the end of an integration step   |
| endHop(t_i, t_{i+1})   |  | any actions to be taken at the end of a hop. <b>This is sim-wide; called in unison for all models</b>   |
| <b>DartsModels</b>   |  |   |
| maskedCoordinates(t)   |  | compute $Q/U$ masked coordinates DARTS kinematics inputs  |

Since not all Models are complex enough to require all of the equations in the general Eq 6, Dshell provides specialized Dshell Model classes for simpler versions of models and these are described in the following sections. Among these classes, only the DartsModel class supports the full complement of the equations in Eq 6.

The actual calling sequence of the Model methods listed in Table Model methods for advancing the simulation over a time interval is described in the Dataflow Model method calling sequence section.

- `setup(t_i, t_{i+1}) *startIoStep(t_i, t_{i+1})`: should be used to update discrete states across the hop interval. For FlowModels, this method is the replacement for the `tick()` FlowModel method. For Models, an option is to carry out the update discrete states within the `startIntegrationStep` method, since this method allows updates to be done at a finer granularity.
  - `startHop(t_i, t_{i+1})`: should be used to update discrete states across the hop interval. For FlowModels, this method is the replacement for the `tick()` FlowModel method. For Models, an option is to carry out the update discrete states within the `startIntegrationStep` method, since this method allows updates to be done at a finer granularity.
  - `endIoStep(t_i, t_{i+1})`: should be used for any multi-rate steps that need to be taken at the end of an I/O step for the model.
  - `updateFlowOuts(t)`: should compute and set flow out values. This is necessary because the flow outputs of one model are the flow inputs of another model, and this method's role is to publish the complete set of flow out values to ensure that all models have correct flow input values.
- The `updateFlowOuts` method should be callable independently, so that all computations needed to set the flow out values should be done with this method. This is necessary since this methods can be called at many different points (depending on the granularity value) and it should be able to fulfill its job of computing outputs on its own.
- `processParams(t)`



TBD: Add explanation for each of the methods

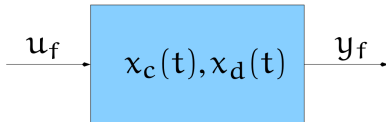
#### 1.2.1.1.3.3.1. ContinuousModels

All Dshell Models that need to be part of the numerical integration process are referred to as ContinuousModels. The following are one or more conditions that will require a Model to be part of the numerical integration process:

- the Model does have continuous states  $x_c(t)$  that would need to be numerically integrated
- the Model generates  $y_m(t)$  inputs into the DARTS multibody dynamics that affect the multibody dynamics equations of motion solution. Examples of this are actuators and gravitational models that apply forces on the multibody system.
- the Model generated  $y_f(t)$  outputs which while not being direct inputs into DARTS, happen to be inputs into other ContinuousModels that do affect the continuous time dynamics and are time-varying over an integration interval. An example of this is an analog PID controller whose output may feed into a motor model which in turn applies torques on hinges in the DARTS multibody model.

If any of these conditions is true, then the Model needs to be implemented as a ContinuousModel. Thus ContinuousModels can have continuous states, or their outputs can affect the continuous time dynamics during an integration step. In the Dshell model classification, we refer to the sub-set of continuous models with direct inputs (i.e.  $y_m(t)$ ) or outputs (i.e.  $u_m(t)$ ) into DARTS as DartsModels (which are discussed in the DartsModels section below) and reserve the ContinuousModel terminology for the remaining continuous models such as shown in Figure ContinuousModel.

*A ContinuousModel instance with its continuous states and inputs and outputs.*



The simpler version of Eq 6 governing a ContinuousModel thus is as follows (for continuous models):

$$\begin{array}{ll}
 y_f(t) = g_f(x(t), u(t), t) & \text{update flowOut outputs} \\
 x_d(t + \Delta_t) = h(x(t), u(t), t, \Delta_t) & \text{discrete state update across hop} \\
 \dot{x}_c(t) = f(x(t), u(t), t) & \text{continuous state derivative}
 \end{array}$$

Note that a ContinuousModel can have discrete states. As shown in Table Model methods, ContinuousModels have the following methods for state propagation:

- `startIntegrationStep(t_i, t_{i+1})`: For ContinuousModels, the number of integration sub-steps defines how many integration sub-steps the hop is split into. For ContinuousModels, the `startIntegrationStep` can therefore be used instead of the `startHop` method for updating discrete states across the integration interval for finer granularity updates when the number of integration sub-steps is greater than 1.
- `preDeriv(t_d)`: if a DARTS model computes any DARTS  $y_f(t)$  inputs (eg. forces, gravity, accels) that are inputs to the multibody dynamics component, then these computations must be done in `preDeriv`.
- `postDeriv(t_d)`: if the model's continuous state derivatives depend on multibody derivatives or inter-body forces, then these computations should go into `postDeriv`. Otherwise, they can be in either `preDeriv` or `postDeriv`. To be on the safe side, all continuous state derivatives computations should be done in `postDeriv`.
- `endIntegrationStep(t_i, t_{i+1})`: `endIntegrationStep` should normally be empty (unless it is changing non-traditional multibody inputs such as body mass which can affect CM calculations done downstream by some other model. Of course these computations should also be done in `preDeriv` etc methods.)
- `endHop(t_i, t_{i+1})` should normally be empty.



TBD: Add explanation for each of the methods

#### 1.2.1.1.3.3.2. DartsModels

The DARTS multibody dynamics model is responsible for the computation of state derivatives based on the body kinematics, mass properties, external gravity and applied forces. It is also responsible for computing the CM location, inter-body forces, current inertia properties, the  $T$  generalized forces for prescribed subhinges. These computations depend on the  $Q/U$  state coordinate values for the DARTS multibody model as well as the kinematic/mass parameters defining the DARTS multibody model.

Normally, the DARTS model's kinematic/mass properties are constant during a simulation, while the multibody  $Q/U$  state coordinate values are owned and propagated over time by the numerical integrator. However the input external and gravitational forces can vary during run-time and computing them is one of the roles of DartsModels. Another role is the computation of quantities dependent on the state and state derivatives of the  $Q/U$  coordinate values.

DartsModels are derived from the ContinuousModels class and represent the subset of continuous models that have direct input/output connections to the DARTS multibody dynamics model as shown in Figure DartsModel. DartsModels can be used for actuator (eg. thruster) and sensor (eg. IMU) device model implementations. Specifically, these models have  $u_m(t)$  and/or  $y_m(t)$  inputs/outputs that interact with the DARTS model. DartsModels thus support the full suite of equations in Eq 6. To simplify the interface with the DARTS multibody model, DartsModel instances are provided access to node/hinge objects from the DARTS multibody model object. Further specializations of a DartsModel are:

- SensorModel: these DartsModel instances are associated with a DARTS *body node* where they can query the node's pose, velocity and acceleration properties.
- ActuatorModel: these DartsModel instances are also associated with a DARTS *body node* where they can apply spatial forces at the node (as well as sense the node's pose, velocity and accelerations).
- EncoderModel: these DartsModel instances are associated with a DARTS *body subhinge* where they can query subhinge generalized coordinate, velocity and accelerations.
- MotorModel: these DartsModel instances are also associated with a DARTS *body subhinge* where they can apply generalized forces or prescribed accelerations at the subhinge (as well as query the subhinge like the EncoderModel).

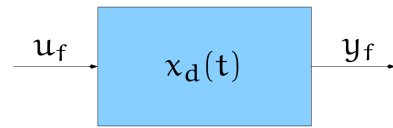
In addition to the methods for ContinuousModels in Table Model methods, DartsModels have the following methods for state propagation:

- maskedCoordinates(t): Some modeling situations deviate from the norm where some of the  $Q/U$  multibody coordinates are not managed by the numerical integrator (eg. trim states) and are referred to as masked coordinates. Also there are times when the multibody model's kinematic, mass properties are not constant during a simulation (eg. fuel consumption). For such off-nominal situations, the maskedCoordinates(t) method should be used for computing these quantities and updating the multibody model.

#### 1.2.1.1.3.3.3. FlowModels

All Models that are not continuous models are referred to as FlowModels. A FlowModel is derived from the Dshell Model class and its simpler structure is illustrated in Figure FlowModel.

A FlowModel instance with its discrete states and inputs and outputs.



A FlowModel does not have continuous states  $x_c(t)$  nor does it have  $u_m$  inputs and  $y_m$  outputs for DARTS, and its  $u_f$  inputs and  $y_f$  outputs do not change within an integration hop. The narrower version of Eq 6 for FlowModels is

$$\begin{aligned} y_f(t) &= g_f(x(t), u(t), t) && \text{update flowOut outputs} \\ x_d(t + \Delta_t) &= h(x(t), u(t), t, \Delta_t) && \text{discrete state update across hop} \end{aligned}$$

Since FlowModels do not have continuous states and do not have outputs that change at the granularity of continuous dynamic, the numerical integrator is not involved in the execution of the methods in FlowModels. This considerably simplifies their implementation. As seen in Table Model methods, the startHop method is the only method needed for state propagation for FlowModels.



TBD: Actually, startHop and endHop can no be used for state propagation now too.

#### 1.2.1.1.3.3.4. Why is DARTS not a model?

Since the DARTS multibody model is a dynamical system (and with continuous states), a reasonable question is as to why it is not itself implemented as a ContinuousModel within Dshell simulations. The key reasons for this are:

##### DARTS inputs/outputs are model dependent:

Most device and environment interaction models have fixed input/output interfaces and number of states. We take advantage of this and use an `mdl` model definition file to define the structure of a model and auto-generate C++ boiler-plate interface code for the model. The user simply needs to populate stub methods with code that captures the physics of the model (eg. thruster behavior; gravitational forces).

Compared with individual device models, the multibody dynamics models and solvers tend to be orders of magnitude more complex and usually developed by specialists. They need to handle different multibody topologies, rigid/flexible bodies, constraints etc etc. Within Dshell, our solution is to include the DARTS multibody general purpose multibody dynamics solver which can be used as middleware for a very broad class of multibody dynamics problems. In contrast with the component device models, with the multibody dynamics we have the reverse situation where the multibody dynamics physics is already available via DARTS, but the structure and interfaces with it depends on the specific nature of the multibody model. The benefits of the `mdl` file based fixed interface definition are not useful for this situation.

The inputs and outputs from the multibody model are defined by the many calls that DartsModel methods make to the DARTS methods, and thus varies from simulation to simulation. Thus there is no simple way to structurally pre-define the number and types of inputs for a DARTS model as is done via `mdl` files for Models. In recognition of the extensive interactions between the family of DartsModels and the DARTS model, we instead treat the DARTS model as a common foundational middleware available to all the component DartsModels.



TBD: Actually, startHop and endHop can no be used for state propagation now too.

##### DARTS inputs/outputs can change during run-time:

One of the important abilities of DARTS multibody models is that structural changes such as addition and deletion of bodies and nodes can be done on the fly. The connectivity of bodies and hinge types can also be changed on the fly. Such structural changes can alter the number of continuous states as well as the inputs and outputs from the model.

#### 1.2.1.1.3.3.5. State Propagator FlowModel

The overall dynamical system consists of the FlowModels with their discrete dynamics, and the continuous dynamics world containing the ContinuousModels, DartsModels and the DARTS multibody dynamics model. While the Models are connected together into a single dataflow, the special numerical integration requirements for the continuous dynamics requires them to be handled differently. For instance, suppose we have the dataflow in Figure DARTS and FlowModels.





Figure 3. A simple dataflow consisting of a mix of FlowModels and a DartsModel

We tie together the FlowModel and continuous dynamics worlds via a special FlowModel referred to as a *state propagator (SP)* model. The SP FlowModel's startHop method calls the integrator's step method to propagate the continuous state across the full continuous dynamics world across a time interval.

Using this SP model, we can view the simulation data flow consisting of two different model dataflows. The top-level dataflow consists of only FlowModels, with one of them being an SP FlowModel. The FlowModels dataflow corresponding to Figure DARTS and FlowModels is shown in Figure StatePropagator and FlowModels.

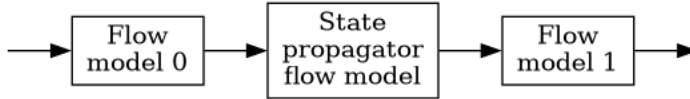


Figure 4. The FlowModels dataflow corresponding to Figure DARTS and FlowModels using the SP FlowModel

For the more complex dataflow in Figure More complex dataflow,

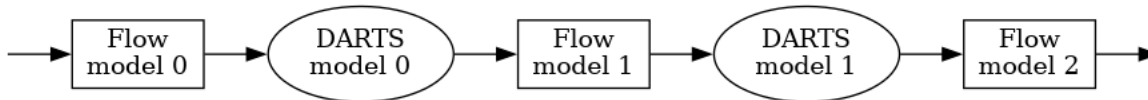


Figure 5. A more complex dataflow consisting of a mix of FlowModels and multiple DartsModels

the corresponding FlowModels dataflow has the form shown in Figure Other StatePropagator and FlowModels.

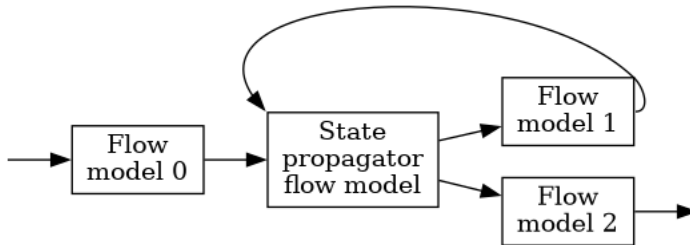


Figure 6. The FlowModels dataflow corresponding to Figure-??? using the SP FlowModel

The Dshell simulation advances time by traversing this FlowModels dataflow. When the SP FlowModel in this dataflow is executed, it causes the numerical integration step to take place, which causes the dataflow consisting of all the ContinuousModels and the DARTS multibody dynamics to be executed to propagate the continuous state.

#### 1.2.1.1.3.4. Dataflow Model method calling sequence

This section describes the proper calling sequence for the Model methods to compute the equations in Eq 10 for a single I/O step. The calling sequence is based on the following rationale. The methods such as startHop, preDeriv etc. that deal with state propagation, require correct and up to date inputs to work correctly. In the Models dataflow, the inputs of individual Models are connected to the outputs of other Models. In order to ensure that all the Models have up to date values, the updateFlowOuts method for each of the Models needs to be called before any of the state propagation methods are called. The calling sequence for the Model methods to advance the simulation across a single I/O step from time  $t_i$  to  $t_{i+1}$  is as follows:

The calling sequence for model methods during an I/O step.

- start I/O step: state  $x(t_i)$ , input  $u(t_i)$ , time  $t_i$
- call maskedCoordinates( $t_i$ ) for all **STARTIO** masking granularity DartsModels
- call updateFlowOuts( $t_i$ ) [ $y_f(t_i) = g_f(x(t_i), u(t_i), t_i)$ ] for all **STARTIO** output granularity Models
- call startIoStep( $t_i, t_{i+1}$ ) for all Models with matching  $t_i$  start I/O time
- start hop: state  $x(t_i)$ , input  $u(t_i)$ , time  $t_i$
- call updateFlowOuts( $t_i$ ) [ $y_f(t_i) = g_f(x(t_i), u(t_i), t_i)$ ] for all **STARTHOP** output granularity
- call startHop( $t_i, t_{i+1}$ ) [ $x_d(t_{i+1}) = h(x(t_i), u(t_i), t_i, t_{i+1} - t_i)$ ] for all Models
- call endIoStep( $t_i, t_{i+1}$ ) for all Models with matching  $t_{i+1}$  end I/O time



TBD: Explain why updating continuous state derivatives should be done in postDeriv instead of preDeriv. Does it really matter?

The state propagation FlowModel is responsible for invoking the numerical integrator to advance the state of all the 'ContinuousModel' models across a hop. The following describes the calls that happen within the startHop( $t_i, t_{i+1}$ ) call for the state propagation model:

The calling sequence for model methods during a simulation hop.

```

for i = 1 ... nIntegrationSteps() for all ContinuousModels & DartsModels
  start integration: state  $x(t_j)$ , input  $u(t_j)$ , time  $t_j$ 
  call updateFlowOuts( $t_i$ ) [ $y_f(t_i) = g_f(x(t_i), u(t_i), t_i)$ ] for all STARTINTEGRATION output granularity
  call startIntegrationStep( $t_j, t_j + \Delta_t$ ) [ $x_d(t_{j+\Delta_t}) = h(x(t_j), u(t_j), t_j, \Delta_t)$ ]
  {
    for each derivative sub-sample
      sub-sample deriv state  $x(t_d)$ , input  $u(t_d)$ , time  $t(t_d)$ 
      call maskedCoordinates( $t_d$ ) for all PREDERIV masking granularity DartsModels
      call updateFlowOuts( $t_d$ ) [ $y_f(t_d) = g_f(x(t_d), u(t_d), t_d)$ ] for all PREDERIV output granularity
      call preDeriv( $t_d$ ) [ $y_m(t_d) = g_m(x(t_d), u(t_d), t_d)$ ]
      call DARTS multibody derivative, compute  $\dot{U}(t)$ 
      call maskedCoordinates( $t_d$ ) for all POSTDERIV masking granularity DartsModels
      call updateFlowOuts( $t_d$ ) [ $y_f(t_d) = g_f(x(t_d), u(t_d), t_d)$ ] for all POSTDERIV output granularity
      call postDeriv( $t_d$ ) [ $\dot{x}_c(t_d) = f(x(t_d), u(t_d), t_d)$ ]
    end loop
  }
  end integration: state  $x(t_j + \Delta_t)$ , input  $u(t_j + \Delta_t)$ , time  $t_j + \Delta_t$ 
  optional bonus derivative call (repeat derivative subsample calls)
  call maskedCoordinates( $t_j + \Delta_t$ ) for all ENDINTEGRATION masking granularity DartsModels
  call updateFlowOuts( $t_j + \Delta_t$ ) [ $y_f(t_j + \Delta_t) = g_f(x(t_j + \Delta_t), u(t_j + \Delta_t), t_j + \Delta_t)$ ] for all
  ENDINTEGRATION output granularity DartsModels
  call endIntegrationStep( $t_j, t_j + \Delta_t$ )
end loop
call updateFlowOuts( $t_{i+1}$ ) [ $y_f(t_{i+1}) = g_f(x(t_{i+1}), u(t_{i+1}), t_{i+1})$ ] for all ENDHOP output granularity
call endHop( $t_i, t_{i+1}$ ) for all ContinuousModels
end hop: state  $x_n$ , input  $u_n$ , time  $t_n$ 

```

The purpose of the optional bonus derivative call at the end of an integration step is to recompute all the state derivatives (including those of DARTS) with the new integrated state value, and to update all the inputs and outputs that depend on DARTS accelerations. This call is only needed when there are Model flow ins and outs that are multibody acceleration dependent.



TBD: Is the bonus call in the right place?



TBD: Don't we also need the bonus deriv call if we want to log accelerations?

#### 1.2.1.1.3.4.1. Accessing time in Model methods

The simulation time is well-defined between hops since the full system state has been propagated to the end of the hop. However simulation time is not well defined in the midst of a simulation hop since model states are propagated individually, in the middle of a state update, some models will already have been updated (they will contain state at  $t_1$ ), while others will not yet have been updated (they will contain state at  $t_0$ ). Thus the overall simulation is in an inconsistent state. To avoid such ambiguities, the pertinent time for the model methods is explicitly passed in as arguments (see the Using time in the model methods section).

#### 1.2.1.1.4. Model implementation guidelines

##### 1.2.1.1.4.1. System model decomposition guidelines

The idea behind a good decomposition into component models is that the individual components respect encapsulation, so that the only interaction between models is through their public flow inputs and outputs. Characteristics of a good decomposition are:

- A model only uses its own state, state derivative, and flow ins and does not require direct access to the state or state derivative of any other model. The only access to the information from another model is through flow inputs and output connections. It is perfectly reasonable for a model to publish its internal state  $x(t)$  into its  $y_f(t)$  flow output, which can then be connected to another model to use - and thus allowing models to communicate state information. However, since we do not allow a model's flow output  $y_f(t)$  to depend on the model's state derivative  $\dot{x}_c(t)$ , there is no way to publish the state derivative information. As discussed in the Introduction section, the state derivative (or a function thereof) can be made available as an output at the cost of introducing feed-thru in the model. *So when a model appears to need state derivative information from another model, it is possible that the decomposition is too fine and incorrect, and that the pair of models should be combined into a single one.*

Note that the above restriction only applies among models. In general models are allowed access to the state and state derivatives of DARTS via the DARTS entities they are associated with (bodies, nodes, or hinges).

- A model only calls its own methods, and does not require calling the methods of any other model. An example of violating this principle is the steering dynamics implementation, where calls to the steering model methods were being made by the aerodynamics model. The correct solution there was to break up the aerodynamics model so that the steering model could be placed between the two parts. *Thus when there appears to be a need to call another model's method, it is likely that the decomposition is too coarse and incorrect, and the model needs to be split up.*
- A model should not deactivate itself. An example of violating this principle is a model that deactivates itself when a lander spacecraft has reached the surface, and it deactivates itself at this point. The issue with a model deactivating itself is that it can interfere with the state rollback process that is used in zero-crossing detection. The rollback process is not equipped to change the active/inactive state of models, and thus active/inactive state changes of models should only be done between hops - and at the simulation level.
- All models have access to the `Simulation` simulation manager instance, and through it the `DartsMbody` multibody instance. However, models should avoid using this access to query or change variables outside its local scope. Such out of scope interactions introduce couplings that violate the encapsulation goals of component based design.



TBD: Find a home for the text below

The expectation is that each of the methods be fully responsible for doing its assigned computation on its own, and not be dependent on other methods having been called earlier (except via data passed in via flow ins). This is an important point since currently there are a number of non-compliant model implementations that violate this expectation. In several such models, the `updateFlowOuts(t)` assumes that the `preDeriv(t)` method does the necessary flow outputs computations and has already been called, and `updateFlowOuts(t)` thus only does the additional step of publishing the flow output values. Such implementations are problematic since `updateFlowOuts(t)` will generate incorrect outputs were `preDeriv(t)` not to have been called earlier.

The `endToStep` method for a model is called at the end of its multi-rate I/O step.

#### 1.2.1.1.4.2. Guidelines for model methods

The following describes guidelines for implementing the model methods:

- 1. Design models for tomorrow:** The initial implementation of a Model is often motivated by a near-term simulation need. However, The fact that we are choosing to implement it as a Model comes with the implicit promise that this component Model may be reused by other simulations. Thus it is important, that the Model methods and design be not narrowly focused on the near-term, but instead be written in a way that it can be used in future simulations with entirely different needs and goals.  
  
Lets say for instance, we decide to add extra outputs to the model for perhaps future generality, or perhaps for easier data logging of the outputs - even though the current simulation does not have any downstream models using the outputs. There is a temptation to put the computation of these outputs in methods such as `preDeriv` (for local use) or `startHop` (for data logging) since it meets the needs of the current simulation. However, such an implementation can impact future simulation which do choose to use these outputs as input for downstream models. Putting the computation of these outputs into incorrect methods may cause them to not be updated at the rate needed by downstream models.  
  
The correct approach is that the output computations be implemented in `updateFlowOuts`. For, the original simulation, the output granularity can be set such that this method is called to satisfy just the needs of the simulation (where no downstream models are using the output). However, for other simulations that do connect these outputs to other models, the output granularity can be set to a higher value so that the output is updated more frequently as needed by the new simulation. With such an implementation, there would be no way to get the right behavior for future simulations. Thus for instance, if a model outputs the CM location for a group of bodies, it must do so in the `updateFlowOuts` method, even when the `preDeriv` method is also a user of the information.
- 2. Setting model outputs:** Only `updateFlowOuts`, and none of the other methods, should compute and set flow out values. This is necessary because the flow outputs of one model are the flow inputs of another model, and this method's role is to publish the complete set of flow out values to ensure that all models have correct flow input values.  
  
The `updateFlowOuts` method should be callable independently, so that all computations needed to set the flow out values should be done with this method. This is necessary so that the frequency of calling this method can be controlled by properly setting the output granularity level for the model.
- 3. Selecting output granularity level:** `[[itm:outgran]]` A model's output granularity level controls how often the model's `updateFlowOuts` method will get called. The correct setting of this value is determined by the following factors:
  - **Intrinsic factors:** Often a model's methods (eg. `preDeriv`) require data computed for setting outputs. Since this computation is needed for setting outputs, its implementation should be done in `updateFlowOuts`. However for the model's other methods, it would be preferable to just use such computed data and avoid recomputation. This can be accomplished by including in the level for each user method in the the output granularity for the model. This this is known at the time of the model implementation, the default value for the output granularity for the model should be set to value defined as a combination of all the user method levels within the model. This default value should be compiled into the model's setup or constructor methods. This default value defines the the minimum granularity level for the model.
  - **Extrinsic factors:** Additional requirements may come from extrinsic factors such as how the model's outputs are connected to the inputs of other models, or logged in a specific simulation. A downstream model, or data logging needs, may require a certain frequency update rate that may or may not be the same as the default value defined by the model. Meeting this requirement, can be met by simply registering these additional levels to the model's granularity level. For example, if a model flow output is registered for data logging - and data logging typically happens at a hop boundary - then this extrinsic need would require that the output granularity include the `endHop` level.
- 4. Updating masked coordinates:** `[[itm:maskcrd]]` When a `DartsModel` - instead of the numerical integrator - manages *masked Q/U* multibody coordinates, the computation and setting of these masked coordinated values should be done by the `maskedCoordinates` method. These masked coordinates are inputs into the DARTS kinematics component and directly affect the positions and velocities of frames in the system. The `maskedCoordinates` method should be callable independently, so that all computations needed to set the masked coordinate values should be done within this method.  
  
This method is called before the `updateFlowOuts` so that multibody parameter updates that can effect the multibody kinematic, static and dynamic computations (eg. CM location) have already been applied.  
  
Generally, the outputs of a `DartsModel`, depend on the coordinate values. So one would expect that the masking granularity to normally be as at least as fine as the output granularity level. Indeed, coordinate value changes change properties of the DARTS model, and hence can impact the computation of other `DartsModels`. It is therefore not unusual to set the masking granularity to the `ALL` level due to its global impact.  
  
At times, when the cost of computing the masked coordinates is expensive (eg. trim computations), it is possible that the masked coordinates granularity may be set to a coarser value while trading off fidelity for performance.  
  
Just as for output granularity, the proper setting of masking granularity should be set based on intrinsic (the way the model methods are implemented) and extrinsic (how the model interacts with the other data flow models) factors.

Besides their extrinsic effect from their effect on output computations, there is another potential extrinsic factor to be considered when setting the masking granularity. This is discussed next in Updating parameters item.

5. **Updating DARTS model parameters:** `[[itm:maskparam]` There are situations when the DARTS model's kinematic and mass-property need to be updated during run-time due to fuel consumption or other such situations. These computations and multibody updates should also be done in the `maskedCoordinates` method.

These multibody parameter updates change the DARTS model and can thus impact of other models. Thus these updates are another type of extrinsic impact - beyond that on the outputs. As argued in the Updating masked coordinates item, the masking granularity should be set to `ALL` in the absence of other factors.

#### 6. Do unto others as thou shalt have done unto you:

While designing the contents of a model's methods, there can be a tendency to focus on the state of the model and the variability of its inputs. However, models do not exist in isolation, and their outputs are the inputs for downstream models in the data flow. Thus it is important that updating a model's outputs be given as much importance as the attention the model's inputs (which are created by other upstream models).

The key point is that a model's proper functioning depends on its inputs being up to date and having correct values. The same requirement also applies to its outputs, since these outputs are the inputs for downstream models. This requirement is over and above its own internal use of the data associated with the output.

Thus for instance, if the model is computing the CM location of the multibody system as an output. The model has no idea how a downstream model will use this output value. So this computation must be implemented within the `updateFlowOuts` method so that with the (externally) appropriate setting of output granularity for the model, the downstream models can be assured that this output has the correct value whenever it is used. In the instance that the CM location is *not* an output of the model, but is used within the the model - say within its `preDeriv` method, then its computation should be implemented within the `preDeriv` method. One advantage of the latter is that the `preDeriv` method gets called less often, and also the `updateFlowOuts` method may not exist.

**TBD:** Use examples to communicate this idea. Use the aero case, where the forces need to be computed at the CM location. This can be implemented as 2 models, one for computing the CM location as an output, and a second for computing the forces. Or this could be a monolithic single model that does both. What goes into `updateFlowOuts`, `preDeriv` will be different for the 2 cases. Now let's add the case, where the CM location is not changing just due to coordinates, but also due to fuel consumption. Use this to motivate that this update should be done in `maskedCoordinates` - applies to both outputs and `preDeriv`.

7. **Setting DARTS inputs:** if a DARTS model computes any DARTS  $y_f(t)$  inputs (eg. forces, gravity, prescribed accels) that are inputs to the multibody dynamics component, then these computations must be done in `preDeriv`.
8. **Dependence on multibody accelerations:** if the model's continuous state derivatives depend on multibody derivatives or inter-body forces, then these computations should go into `postDeriv`. Otherwise, they can be in either `preDeriv` or `postDeriv`. To be on the safe side, all continuous state derivatives computations should be done in `postDeriv`.

**TBD:** Explain why

9. **Discrete state updates:** `startHop` should be used to update discrete states across the hop interval. For `FlowModels`, this method is the replacement for the `tick()` `FlowModel` method. For `Models`, an option is to carry out the update discrete states within the `startIntegrationStep` method, since this method allows updates to be done at a finer granularity.

For `ContinuousModels`, the number of integration sub-steps defines how many integration sub-steps the hop is split into. For `ContinuousModels`, the `startIntegrationStep` can therefore be used instead of the `startHop` method for updating discrete states across the integration interval for finer granularity updates when the number of integration sub-steps is greater than 1.

10. **Ending state propagation:** `endIntegrationStep` should normally be empty (unless it is changing non-traditional multibody inputs such as body mass which can affect CM calculations done downstream by some other model. Of course these computations should also be done in `preDeriv` etc methods.)

The `endHop` should normally be empty. The `endIoStep` should be used for any multi-rate steps that need to be taken at the end of an I/O step for the model.

11. **Multi-rate models:** The time propagation of the continuous and discrete states are carried out in a lock-step across all the `Models` in the data-flow. However individual models may have actions that need to be carried out at specific rates. Support for such *multi-rate* comes in two forms. Firstly, it is possible to set such a specific rate for any `Model` - with the default rate being defined by the simulation's step size. The simulation ensures that hop boundaries are set to honor multi-rate boundaries set across all the `Models`.

Secondly, each model's `startIoStep` and `endIoStep` methods are called at the start and end respectively of the multi-rate step for each model. Thus any multi-rate actions that should be carried at the start of a multi-rate step (eg. resetting counters and timers) should be implemented within the `startIoStep` method. Similarly, any multi-rate actions that should be carried at the end of a multi-rate step (eg. camera/sensor output) should be implemented within the `endIoStep` method.

12. **Object parameters:** It is possible to pass in any object derived from the `NdartsBaseObject` class as a parameter to a model for its use. This approach is recommended over passing in of `uuid` or `id` parameters to models followed by internal object look up.

13. **Model initialization:** `_lockObject()` should be used to initialize pointers etc that need to be initialized one time (as long as this initialization does not depend on flow input values).

14. **Remove unneeded methods:** methods that are not needed by a model should not be implemented at all rather than having empty stub implementations.

15. **Hide model methods:** Model methods should be protected - and not public - so that they cannot be called externally by other objects such as other models that may interfere with the correct calling order set up by the simulation.

16. **Use time arguments:** All model methods are passed in the time arguments for when they are being called. These should be used instead of using the model or simulation level `time()` method.



**TBD:** Add description of `freshenOutputTime` and that it should only be called in UFO.



**TBD:** Add description and use of `coordMaskType` and `coordMaskData` methods.



**TBD:** Add description and use of `processParams` method.



**TBD:** hops on the other hand are the emergent way a simulation state advances. So when there are multirate models, the hops can be smaller size than the I/O step sizes specified.



**TBD:** The integrationSteps are used to execute hops - and when there are multiple we need multiple ones.\* **TBD:** Thus for continuous models, we need I/O step methods for external rate control, hop methods to do the actual state propagation, and integration step methods to support the integration stepping



**TBD:** For flow models, we also need start/end I/O step methods to allow users to control the multi-rate stepping, and hop methods to control the actual discrete state propagation. The existing tick method will do the actual work of doing the hop and should be named hop() method. The Simulation stepping should call - UFO(start hop time) if startHop granularity is set, hop()/tick(), UFO(end hop time) if endHop granularity is set.\* **TBD:** in ContinuousModels, can only use fsm in start/end IO or integrationStep methods, since time can bounce around in the deriv calls



**TBD:** Make note in models style guide that updateFlowOuts should have no side effects and should not effect any internal states. This is so it can be called multiple times with no effect - the idea is that we should be able to call the 'y' output equation (which updateFlowOuts is responsible for) for the current state. Thus, during initialization, once the state has been initialized across the system, we should be able to call updateFlowOuts to refresh all the signals. This would be the replacement for step(0), but without any deriv etc calls - to answer the question about whether a model has feed thru, we need to answer the question of whether when we wiggle any flowIn, does the flowOut value change. If so, then the model has fee thru. - sim.step(0) call should simply be a call to all updateFlowOuts - the idea is to initialize the state first, and then call updateFlowOuts to update all the signals without effecting the states - the granularity setting is use to control whether updateFlowOuts is called \*after\* the method specified by the granularity setting is called



**TBD:** if no UFO, granularity should be undefined\*



**TBD:** if no flow in or flow out, feed thru should be false



**TBD:** if granularity of UFO or masking is unset, or feed thru is unset, generate a warning



**TBD:** if no startIOStep, granularity for it should not be set



**TBD:** if no endIOStep, granularity for it should be undefined



**TBD:** if in compliance mode, granularity should be endHop and not tick



**TBD:** if no tick() method, then granularity for it should not be set



**TBD:** Read up feed thru issue in Models style guide - especially for DARTS block

#### 1.2.1.1.4.2.1. Model guidelines summary

Following the detailed guidelines discussion about Model methods, we summarize below the expectations for each model method:

1. **Constructor method:** (all Models) Should set the default output and masking granularity levels, model compliance level and feed thru attribute for the model.
2. **setup() method:** Called after flow inputs/outputs have been tied to signals. Should set the size of any variable size flow inputs/outputs. Should *not* do default initialization of parameters or states (to allow detection of uninitialized values).
3. **\_lockObject() method:** (all Models) One time initialization of any members or objects needed by the model such as frame, body, helper class objects. Parameters would have been set by the time this method is called. Do not use this method for initialization that depends on flow input values since these may not be up to date.
4. **maskedCoordinates method:** (DartsModels only) Only available for DartsModels. Should include computation of masked coordinates, and any other DARTS model parameter updates (eg. kinematic, geometric, mass/inertial properties) as discussed in Updating masked coordinates and `///itm:maskparam`. Updating parameters items.
5. **masking granularity:** (DartsModels only) Default value should be hard-coded in the Model's constructor or setup() method based on the model's intrinsic properties as discussed in Selecting output granularity item. Should generally be set to `ALL` since this effects the DARTS models properties in the model's setup() or constructor methods as discussed in the Updating masked coordinates item.  
A *coarser* setting for masking granularity may be used if the maskedCoordinates computations are expensive - but this may result in a reduction in simulation fidelity.
6. **feed thru:** (all Models) Value should be set in the constructor. The value should be `NONE` if the model does not have *both* flow inputs and outputs.
7. **updateFlowOuts(t) method:** (all Models) Should update all the flow outs for the Model. Call `freshenOutputTime()` as needed for each of the outputs being set.
8. **output granularity:** (all Models) Default value should be hard-coded in the Model's constructor or setup() method based on the model's intrinsic properties as discussed in the Selecting output granularity item. The level should be `NONE` if the Model has no flow outputs. If the output of the model is being data logged, then the output granularity should include the endHop level.  
The Model's output granularity can be *increased* by a simulation based on extrinsic factors such as the the use of the model's outputs by other models or for data logging. The output granularity may be *lowered* if the cost of the updateFlowOuts method is expensive - but this may result in a reduction in simulation fidelity.
9. **startIOStep method:** (all Models) Should include anything that needs to be done at the start of a multi-rate step for the model. Also, should be used for any one time initialization that depends on flow inputs and thus cannot be done in the `_lockObject()` method.
10. **startHop method:** (all Models) Use for discrete state updates across the upcoming hop for FlowModels. For ContinuousModels, you have the choice of doing such updates in the startIntegrationStep method for finer granularity.

11. **startIntegrationStep method:** (ContinuousModels only) Any discrete state updates across the integration sub-step if not already done in the startHop method.
12. **preDeriv(t) method:** (ContinuousModels only) Any external forces, gravity, prescribed acceleration, generalized forces computations that may effect the DARTS computation of accelerations.
13. **postDeriv(t) method:** (ContinuousModels only) Computation of any continuous state derivatives for the ContinuousModel. Also any computations that make use of the multibody generalized acceleration values.
14. **endIntegrationStep method:** (ContinuousModels only) Generally empty, but can also do discrete state updates if not done in startIntegrationStep.
15. **endHop method:** (ContinuousModels only) Generally empty, but can also do discrete state updates if not done elsewhere.
16. **endIoStep method:** (all Models) Should include anything that needs to be done at the end of a multi-rate step for the model.
17. **processParams(t) method:** (all Models) Should process new parameter values set by the user.



TBD: Where does the size of variable size parameters get set?

#### 1.2.1.1.4.3. Guidelines for output granularity values



TBD: Define legacy stepZero() - call UFOs. always done in the beginning after bindState(). All UFOs call at end of hop?????

The stages where the updateFlowOuts method is called is determined by the ContinuousModel's **output granularity** value. For each of the Model methods described in Table Model methods, there is a granularity level available and arbitrary combinations of these can be selected for a Model's output granularity. For each of the assigned granularity levels, updateFlowOuts method is called together with the Model method to ensure that all the flow inputs and outputs are refreshed. The **ALL** output granularity setting forces the updateFlowOuts(t) method to be called before each methods sweep across all the Models to ensure that all the model inputs are up to date.

As discussed in the Dataflow Model method calling sequence section, it is important that all the inputs for the dataflow models be updated before any state propagation methods are called. In principle this implies that a granularity of **ALL** should be the default for models so that the updateFlowOuts method gets called ahead of each state propagation method call.

While in the most general case, flow outs can depend on the inputs  $u(t)$ , the state  $x(t)$  and the current time  $t$ , the dependency is often narrower for individual ContinuousModels. The user can however change this setting by calling the models updateFlowOutsGranularity method with the desired value.

- If the flowOuts do not depend on the flow ins, they only need to be called when the multibody state, the ContinuousModel's state or time changes.
  - This implies that they need to be called at most with `STARTINTEGRATION` and **PREDERIV** granularity.
  - If the outputs depend only on the discrete states, then the granularity needs to be just **STARTINTEGRATION** depending on whether the discrete state updates happen.
  - In addition, if the flowOuts depend on the  $\dot{U}(t)$  multibody accelerations, then the output granularity levels should include the **POSTDERIV** level.
  - In addition they may need to have the **ENDINTEGRATION** granularity to update the flowOuts with the integrated state and time values.
  - The calling sequence in the Guidelines for output granularity values section corresponds to a granularity setting of **STARTIO + PREDERIV + POSTDERIV + ENDINTEGRATION**.



TBD: Check this last item

- If the flowOuts depend on the flow in inputs, then they need to have **ALL** granularity since the inputs can change for any level of sweep call. This defensive posture however can result in several calls to the method and unneeded flowOuts computations even when the input flow ins have not changed. A way to reduce such unnecessary computations is to check the **freshness** of individual flow ins to detect whether they have changed in value. The updateFlowOuts(t) method can use this check to only update the flowOuts whose values are affected by changes in time or input flow ins.

While the above describes the steps needed for ensuring that the inputs to the Models are up to date and correct, there are times when the computation of the flowOuts may be very expensive and/or the flowOuts change very slowly. An example of this are the atmosphere **GRAM** based models where the **GRAM** calls can be quite expensive. In this case, the developer is free to used coarser granularity settings such as simply **STARTIO** or `STARTINTEGRATION` to trade off some fidelity for reductions in computational cost.

#### 1.2.1.1.4.4. Guidelines for masking granularity values

Similarly the stages where the maskedCoordinates method is called is determined by the DartsModel's **masking granularity** value. The user can change this setting by calling the models updateMaskingGranularity() method with the desired value.



TBD: Why do we need granularities for masking?



TBD: Add example from Havarad about time delay or Vicon model, that are multi-rate, have a random noise generator that need to be called just once per I/O step. Need **STARTIO/ENDIO** output granularity for this.



TBD: Some flow outs might be only be there to support data logging - may never connect to any other model.

#### 1.2.1.1.5. Dataflow model calling order and sorting

Models are coupled to each other through their flow inputs and outputs, and to DARTS through their inputs and outputs into DARTS. This connectivity has implications on the calling order of the models. The basic **model ordering rule** is that *a Model with feed thru must be called prior to any Model that its outputs provide inputs to*. Enforcing this rule when selecting a model calling order will ensure that the inputs for each Model have the correct and up to date values for its methods to use. Using this model ordering rule, the correct calling order within a dataflow is easy to determine when there are *no loops* in the connectivity of the models.

Having said this, we do need to take into account the hybrid nature of a dataflow containing both the discrete FlowModels and ContinuousModels. As discussed in the State Propagator FlowModel section, the system actually can be viewed as two distinct dataflows. One dataflow includes a state propagator FlowModel and consists of just FlowModels. The second dataflow contains the ContinuousModels and DartsModels that are encapsulated within the state propagator FlowModel. The overall model sorting process sorts each of the dataflows separately. While this is straightforward in the absence of loops, as we saw earlier, while the dataflow in Figure Other StatePropagator and FlowModels is free of loops, its corresponding FlowModels dataflow in Figure Other StatePropagator and FlowModels does contain loops once the state propagator is introduced. The next section takes on the topic of sorting the models in the presence of dataflow loops.

#### 1.2.1.1.5.1. Topological and Algebraic dataflow loops

The simulation knows to sort the models, and to evaluate the update equations for upstream models prior to the downstream ones, but this will fail if we have loops such as in Figure Other StatePropagator and FlowModels or Figure Simple dataflow with loop.

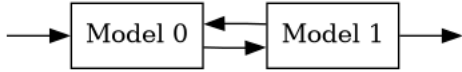


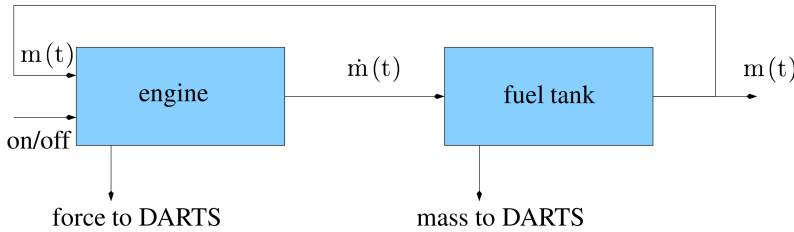
Figure 7. A simple two model dataflow with a loop



TBD: Fix Figure ??? so the model 1 to model 0 feedback line goes around as in Figure ??? so the loop is more obvious.

In this section we look at determining the correct calling order when loops are in fact present. A sequence of Models forms a **topological loop** when their the output of each Model is connected to an input of the next Model with the final ContinuousModel's output connected to the inputs of the first ContinuousModel. The pair of engine and fuel tank models in Figure Topological models loop form a topological loop.

*Propulsion system model with topological, but no algebraic loops.*



A topological loop is said to be an **algebraic loop** when every Model in the loop has feed-thru.

When a topological loop is *not* an algebraic loop, we can directly use the model sorting rule to determine a calling order for the Models that complies with the rule. Note that the model sorting rule only determines a *partial order*, which means that there can be multiple model order solution sequences that are in compliance and legal (and which will generated the correct output). From a graph sorting perspective, the only edges that are relevant are the ones connecting the outputs of a Model with feed-thru to the inputs of other Models. Since a non-algebraic loop has at least one Model that does not have feed thru, the edges in the graph do not form a loop, and no cuts are needed to determine a legal sorting order.

On the other hand, the graph for algebraic loops contains cycles that is a reflection of the implicit algebraic relationship between the input/output values for the sequence of Models in the loop. No sequencing order is possible for such loops that complies with the model ordering rule. The only rigorous option is to solve the implicit conditions for the correct input/output values. As an alternative to this expensive option, we require the user to specify a *break* in the graph's cycle to allow a determination of a calling order for the models. This is an approximation since one of the inputs will be one call behind. Instead of specifying a break, an alternative option is to allow disabling the feed-thru property of one of the Models in the loops for sorting purposes so that it affectively becomes a topological loop and correct sorting can be carried out. We do not pursue this approach, since it is possible for models to belong to multiple loops, and the breaks approach allows the user to more flexibly control the sorting for each of the loops using different strategies.

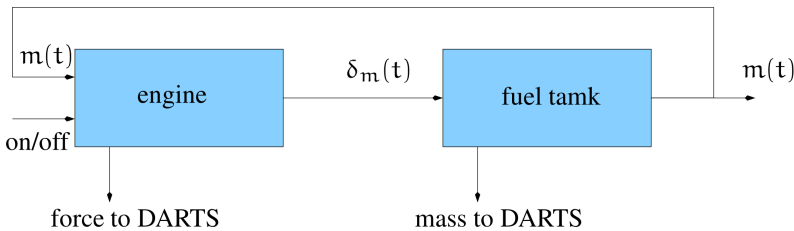


TBD: May want to reword "one call" above; one "hop", "step", etc?

#### 1.2.1.1.5.1.1. Propulsion system example

To illustrate the discussion of loops, let us examine the modeling of a propulsion system in more detail. The thrust generated by the engine is a function of the on/off input, and the remaining fuel  $m(t)$ . As the engine fires, it generates a force input for DARTS and consumes fuel. The fuel consumption information is passed on through an output to the fuel tank model which updates the fuel remaining and provides an input to DARTS to update the fuel tank mass properties in the model. Figures Topological models loop and Algebraic models loop illustrate two different implementations of such a propulsion system using a pair of connected DartsModels. The difference between them is whether the fuel mass is a continuous or a discrete state.

*Propulsion system model with topological, but with algebraic loops.*



In Figure Topological models loop, the fuel mass is a continuous state of the fuel tank DartsModel. The engine DartsModel's `updateFlowOuts` method uses the  $m(t)$  remaining fuel input to compute the  $\dot{m}(t)$  fuel mass consumption rate for its flow out, while its `preDeriv` computes the ( $m(t)$  dependent) engine thrust and sets it in the DARTS model. The fuel tank's `postDeriv` method uses its flow input to set its  $\dot{m}(t)$  state derivative value, and its `updateFlowOuts` method copies the current  $m(t)$  state value into its flow out, while its `preDeriv` method sets the fuel tank mass property in the DARTS body. While the engine model has feed-thru, the fuel tank does not in this model. Thus while the two DartsModels form a topological loop, they do not form an algebraic loop. As a consequence the calling order of the two models *does not matter*, and either of the two possible orders will generate the same result. The order does matter however in the second implementation discussed below since it has algebraic loops.

In Figure Algebraic models loop, the fuel mass is a discrete state of the fuel tank DartsModel. The engine DartsModel's `updateFlowOuts` method uses the  $m(t)$  remaining fuel input to compute the  $m(t)$  dependent  $\delta_{m(t)}$  fuel mass consumption amount (not rate) since the beginning of the time step for its flow out. Its `preDeriv` computes the  $m(t)$  dependent engine thrust and sets it in the DARTS model. For the fuel tank the flow input is used to update the remaining fuel  $m(t)$  value and the `startIntegrationStep` method updates the discrete state and the fuel tank mass property in the DARTS body with new remaining fuel value. There are two options for the fuel tank model's flow output:

1. The fuel tank's `updateFlowOuts` method also computes the new remaining fuel value and uses it to set the remaining fuel flow output. In this case the input immediately affects the output, and this model has feed-thru.
2. The other option is for the fuel tank's `updateFlowOuts` method to use the current state value to set the flow output. Since the state value is updated later, this case, the flow output value will affectively lag the state value by one update. However, for this case this model will not have feed-thru.



TBD: May want to reword "one call" above; one "hop", "step", etc?

For option (1), both of the engine and fuel tank Models will have feed thru, and thus the topological loop will in fact an algebraic loop. For this case, the user has to define a break to determine which of the `ModelupdateFlowOuts(t)` method should be called first. This will introduce approximations since some of the data used in the calculations will lag one step.

#### 1.2.1.1.5.2. Feed thru property



TBD: Add feed thru settings available and setting. \* TBD: Add feed considerations - only when both inputs/outputs, DARTS feed thru

#### 1.2.1.1.5.3. Sorting order of the DARTS multibody model

The DARTS multibody model does not have explicit flow ins or flow outs. Its input values are set directly using function calls by DartsModels, and likewise outputs from it are obtained by DartsModels using function calls. The `preDeriv` method for all the ContinuousModels is called before DARTS's dynamics solver is called, ensuring that all its inputs are correct.

The DARTS model by itself normally does not have feed thru since its implicit inputs only affect the dynamics component which does not have feed thru. However, the kinematic component can have feed thru if some of the  $Q$  and  $U$  coordinate values are masked and managed by DartsModels instead of by the integrator. Feed thru is also possible if a Model directly uses the  $\dot{U}$  values or inter-body constraint forces computed by the dynamics block. We only have an algebraic loop condition if any of these implicit outputs from DARTS feeds thru into the output of any of the DartsModels which in turn is connected to a DartsModel whose inputs into the DARTS models directly affected these outputs. It is left to the user to explicitly handle the model ordering for such rare situations.

Note that when a ContinuousModel's flow outs depend on multibody derivatives, its output granularity should include the `POSTDERIV` granularity setting.



TBD: Should the loops be evaluated for each granularity setting? This will be useful when sorting algebraic loops, since it is possible that the algebraic loop might not exist for the individual granularity settings.

In such a case we must introduce a *break*. This ignores a broken signal connection for the purposes of ordering. A broken signal will be one cycle out-of-date when the update equations are evaluated.

This is all self-consistent for *flow* models:

- State updates look at signals and each model's *own* state only
- We're making sure signals are updated in order

#### 1.2.1.1.6. Discussion

##### 1.2.1.1.6.1. Limitations of prior calling sequence

The prior calling sequence for models called the `preDeriv` and `updateFlowOuts` method in succession for a model before moving onto the next. This calling sequence is fine, except for the case when there is an algebraic loop, i.e. a sequence of Models all of whom with feed thru that form a loop - and we explore this case in more detail here. Algebraic loops require the user to specify an inter-model break for the purposes of determining a calling order for the Models in the loop. With the prior calling sequence, there is no good way to get all the inputs needed by the first Model in the sequence (some of its inputs are connected to the outputs of the last Model in the loop) have a correct value so that it can do its computations correctly. The new calling sequence fixes this problem because all the `updateFlowOuts` methods across all the Models in the loop are called first so that their outputs (and thus inputs) are correct, and only then are the `preDeriv` etc methods called with correct inputs.

##### 1.2.1.1.6.2. Proposed future changes

- model autocoder changes for ContinuousModels
  - if there are no flowOuts, do not generate `updateFlowOuts` stub
  - if there is no  $g_m()$ , i.e. no inputs to DARTS, then do not generate `preDeriv` stub
  - if there are no continuous states, do not generate `postDeriv` stub
  - if no params, do not generate `processParams` stub
  - if no discrete states, do not generate `startIntegrationStep` stub
- Change the Model method calling sequence in the `Simulation` class to the one described in the Guidelines for output granularity values section.
- Remove `preDeriv` method from `Sensor` and `Encoder` models since these are not supposed to generate inputs for the DARTS model.
- Rename the `preDeriv` method to the more appropriate `updateDartsDynamicsInputs` (or `preDartsDeriv`) since the purpose of this method is compute and prepare all the inputs needed for the DARTS dynamics computation.
- Similarly rename `postDeriv` to something like `updateStateDerivs`, because the purpose of this method is to compute the continuous state derivatives for the ContinuousModel.
- Change the default output granularity to be `ALL` as per the calling sequence in the Guidelines for output granularity values section.
- Add API for `updateFlowOuts(t)` to detect at what stage of the calling sequence it is being called. This will allow the `postDeriv` level calls to only update the flowOuts that depend on the multibody accelerations.
- Add support for handling masking granularity and the `maskedCoordinates` method.



- Rename ContinuousModel methods to better reflect their function. preDeriv should become updateDartsInputs. etc.
- Add a feed-thru attribute for ContinuousModel, with the default value chosen to be the most conservative option of `True`, unless the model does not have either flow ins or outs, in which case the value would be `False`.



**TBD:** Should we allow updateFlowOuts to check the current calling level for the model so that they can change their computations depending on whether the model is at preDeriv, endIntegrationStep etc levels.



**TBD:** If Q/U masking, then preDeriv should also be called in endIntegrationStep because the multibody state value would be changed. Is that so? Since have new integrator state, the whole state value would be different. Since we call updateFlowOuts, should not we also call preDeriv in updateFlowOuts. FIX - the kinematics part must be called at all granularities, not the dynamics parts. Make a distinction between kinematics and dynamics DARTS inputs.

#### 1.2.1.1.6.3. Bringing existing Models into compliance

The existing Models need to be brought into compliance with these guidelines. The following phased approach is suggested:

1. Taking stock of the existing Models to assess their current implementation.
2. Scrub the Models to deactivate unneeded empty stub methods
3. Refactor the methods to bring them into compliance.
4. Change the calling sequence in the `Simulation` class to the new one.
5. Set the feed thru attribute explicitly to ContinuousModels.
6. Update the `Simulation` class's sorting to use topological loops, but require breaks only when there are algebraic loops

#### 1.2.1.1.6.4. Taking stock checklist for ContinuousModels

1. What are the Eq 10 thru Eq 8 math equations for the ContinuousModel?
2. **[DONE]** Does the model have continuous states, discrete states, flow inputs, flow outputs, inputs into DARTS, outputs from DARTS, params?
3. Are there methods with empty bodies? Can they be deactivated?
4. Is setup() really needed? Can we move granularity setting (that is typically there) into `_init()` and get rid of this method?.
5. If there are flowOuts, are all flowOuts being updated only by updateFlowOuts?
  - If not, which other method is doing so.
  - If yes, can updateFlowOuts be called standalone or is it dependent on computations from other methods.
6. If have updateFlowOuts, what is its output granularity and is it being set explicitly?
7. If the model computes masked Q/U coordinates, then are these computations being done in the maskedCoordinates method?
8. If have maskedCoordinates, what is its masking granularity and is it being set explicitly?
9. Is preDeriv only being used to update DARTS inputs?
10. If there are continuous states
  - a. where are the continuous state derivatives computations being done?
  - b. are any flow outputs dependent on the continuous state derivatives?
11. Does the model depend on DARTS accelerations? Are the accelerations being processed in the postDeriv(t) method or elsewhere?
12. Does the model look obsolete - does the on-line catalog show it being exercised by any script?
13. Is the model appropriately named?
14. Does the model have feed thru, i.e. do the flow outs depend directly on the flow inputs. Set the flow thru attribute for the model.

#### 1.2.1.1.7. Examples

Example models include

- Fixed time delay
- Fixed rate sensor (Vicon)
- Model with continuous states (second order response)
- Expensive model with coarsened granularity (GRAM model)
- Model with masked Q/U
- Prescribed accel model
- IMU model (continuous states and accel dependency)
- Feed thru model

#### 1.2.1.1.8. Registering Model multi-rate callbacks

Here we provide implementation details on the multi-rate capability discussed in the Multi-rate models section. Generally model do not have special multi-rate calling needs. By default models inherit the simulation I/O step size as their default I/O step size. One has to register a multi-rate callback for models that have special multi-rate needs. For the special case of models that need to be run at a specific fixed rate, a multi-rate callback method can be registered for a model using the following call:

```
sim.registerMultiRateModel(<model>, IO_step_rate_hz)
```

More generally, in C++ one can register an I/O step callback that will be called to determine the next I/O step for a model as follows:

```
sim.registerMultiRateModel(<model>, multirate_callback_cb)
```

Again for completeness, while hop boundaries are determined by the common line timeline as described above, a hop can end earlier than its boundary. This can happen when there is a step-validation callback registered with the simulation. The simulation is always monitoring this callback, and will terminate a hop at a time when such a condition is satisfied. This is described in more detail in the {Dshellpp\_sgstepvalidation\_section\_uri[Zero-crossing events]} section.

#### 1.2.1.1.9. Using time in the model methods

A user often wants to query the simulation time in side a model method. This is a well-defined operation between hops: in startIoStep/endIoStep for instance. However this is *not* well-defined for methods that are invoked mid-step (preDeriv(t)) for instance. Since model states are propagated individually, in the middle of a state update, some models will already have been updated (they will contain state at  $t_1$ ), while others will not yet have been updated (they will contain state at  $t_0$ ). This is an inconsistent state, and asking the simulation about the current time is non-sensical: both  $t_0$  and  $t_1$  are right.

To resolve this ambiguity, it is no longer necessary to make these time queries at all, and in fact it is quite discouraged. Instead, each method now takes arguments that tell the method what time we're transitioning *to* and *from*. Currently the full method declarations look like this:

```
virtual void startIoStep(// now
    const Time::TimeSpec &t_startIoStep,

    // planned io-step-end-time. The step possibly
    // may not end here, if we have step-validation
    // callbacks
    const Time::TimeSpec &t_endIoStep);

virtual void startHop(const Time::TimeSpec &t_startHop,

    // planned hop-end-time. The hop possibly may not end
    // here, if we have step-validation callbacks
    const Time::TimeSpec &t_endHop);

virtual void startIntegrationStep(// now
    const Time::TimeSpec &t_startIntegrationStep,

    // planned integration-step-end-time.
    // The step possibly may not end here,
    // if we have step-validation
    // callbacks
    const Time::TimeSpec &t_endIntegrationStep);

virtual void preDeriv(const Time::TimeSpec &t);

virtual void postDeriv(const Time::TimeSpec &t);

virtual void endIntegrationStep(// When this step started
    const Time::TimeSpec &t_startIntegrationStep,

    // now
    const Time::TimeSpec &t_endIntegrationStep);

virtual void endHop(const Time::TimeSpec &t_startHop,

    // planned hop-end-time. The hop possibly may not end
    // here, if we have step-validation callbacks
    const Time::TimeSpec &t_endHop);

virtual void endIoStep(// When this step started
    const Time::TimeSpec &t_startIoStep,

    // now
    const Time::TimeSpec &t_endIoStep);

virtual void updateFlowOuts(const Time::TimeSpec &t);

virtual void maskedCoordinates(const Time::TimeSpec &t)
```

#### 1.2.1.1.10. Zero-crossing conditions and roll-back implementation

Here we provide more details on the zero-crossing detection feature described in the {Dshellpp\_sgstepvalidation\_section\_uri[Zero-crossing events]} section. To use the zero-crossing detection functionality, each zero-crossing condition callback returns the following enum:

```
// The state value resulting from a step-validation callback. This is a part
// of the full value returned by one of these callbacks:
CB_integration_result enum CB_integration_state
{
    // have never overshoot. normal steps. t_retry invalid
    NOT_SEARCHING,

    // overshoot before. this search step undershot. t_retry valid
    SEARCHING_NOT_OVERSHOT,

    // overshoot before. this search step hit the target. t_retry invalid
    SEARCH_COMPLETED,

    // overshoot before. this search step overshoot. t_retry valid
    SEARCHING_OVERSHOT
};
```

We define a new zero-crossing condition by subclassing this:

```
class CallbackStepValidation
{
    ...

    // Check a hypothesis t. Did we overshoot the target?
    virtual CB_integration_result execute(double t_min, double t) = 0;

    // We just hit the target
    virtual void target_was_hit(void);

    // Reset the seeking logic
    virtual void reset(void) = 0;

    ...
};
```

Subclassing can be done in C or in Python; C is strongly preferred, and the Python support exists mostly for testing. We provide a C++ subclass that implements the most common use case: a bisection search of a given DVar hitting some target. The only user-facing function is the constructor:

```
CallbackFindTargetBisection( \
    const char* _name,
    // Observation DVar. Must have length 1
    const DVar::DoubleVectorLeaf& _x_var,
    // Target we're trying to hit
    double _x_target,
    // Approaching the target from below or from above?
    bool _from_below,
    // Target hit if 0 <= target-x <= eps (unidirectional)
    double eps = 1e-6
);
```

Using the zero-crossing detection from Python is trivial:

```
from Dshell.Dshell_Py
import CallbackFindTargetBisection cb = \
    CallbackFindTargetBisection('find-target-height', height, 950, False)
sim.stepValidationCallbackAdd( cb )
```

#### 1.2.1.1.11. Change history

##### 2020-03-01:

- Lots of updates on guidelines for model methods.

##### 2020-02-25:

- Added change history appendix.
- Fixed model methods table to make startHop apply to all models, and endHop to just continuous ones.
- Added index.

#### 1.2.2. Dshell Simulation object

##### 1.2.2.1. Model order sorting using feed-thru (Proposed)



TBD: update and clean up this section

- Had a discussion with Havard on <2017-02-27 Mon> regarding the sorting of dynamcis models and why it is necessary. In the absence of feed thrus, Havard says that the calling order should not matter (incorrect, it does for the outputs computation). This is how Simulink does it.
- The idea is that if we have  $\dot{x}_1 = A_1 x_1 + B_1 u_1, y_1 = C_1 x_1 + D_1 u_1$  and a second system  $\dot{x}_2 = A_2 x_2 + B_2 u_2, y_2 = C_2 x_2 + D_2 u_2$ , which are interconnected so that  $u_2 = y_1$ , and  $u_1 = y_2$ , then these 2 form a loop. Assuming that either of  $D_1$  or  $D_2$  is 0 (i.e. both do not have feed thru) then (other than round off error) the calling order for system 1 or 2 should not matter. When both have feed thru, then we have circularity at the y's level and only then does the loop need to be broken.
- The idea is almost correct. However there is a ordering that is implied here as well. The main point is that all the y outputs must be evaluated first (before the derivs). However, to evaluate the y's, when one depends on another, we need to have them evaluated in order so that the dependencies get resolved properly.
- Come to think of it, what is currently being done is actually right. Lets take the 2-systems example we had from this morning, withonly one of them having a feed thru. Thus

```
y_1 = C_1 x_1 + D_1 y_2
y_2 = C_2 x_2
```

- To evaluate this, we have to evaluate  $y_2$  first, and then  $y_1$  since the latter depends on  $y_2$ . So even in the case of no feed thru, there is an ordering dependency. The sorting graph built by Dshell should add a edge based on feed thru alone. For the above equations, Dshell will thus see the output of system 2 connected to the input of system 1 (but not the other connection), and will sort them so that system 2 is evaluated before system 1.
- When  $y_2$  also has a dependence on  $y_1$ , i.e. has a feed thru, then we have a true loop. And only then a break is required from the user.

- Note however, that only the strict loops at the y level are a problem in that they require breaks to be specified by the user. However, we will have a connection from system 1 to system 2 so that y\_1 can be used for the \dot x\_2 computation. Dshell cannot tell the difference between whether the input is for the y or the \dot x evaluation, so assumes the worst and assumes it is for a y evaluation and insists on a break. Do we have a way of telling the system what kind of coupling there actually is, so that the loop from the \dot x\_2 dependency does not get seen?
- Actually, for the **most part** DARTS does not feed thru. Input forces, accels etc that are set only effect \dot x and not any outputs. The only exception are
  - when masked Q/U coords are set. These immediately effect frame locations and velocities that may effect downstream models
- when prescribed accels are set, and there are downstream models (eg. IMU, accelerometer) that depend on accel values
  - For these cases where DARTS has feed thru, how do we do the sorting? Is this where we leave it to the user to add constraints to ensure that these dependencies that are hard to discover automatically must be specified by the user?
- With this being the case, we do not need to include the inputs/outputs to and from DARTS in updateFlowOuts since for the most part they do not effect the ordering since there is no feed thru.
- So I believe, that what we are close to having a correct implementation - just need to make sure that all the outputs are computed first. I think we need to document this like this so that it is clear as to what is going on.
- See Havard's email from <2017-02-27 Mon> for a nice long discussion about this. He also says that Simulink S-functions have a flag to indicate whether the block has feed thru. See [[<https://www.mathworks.com/help/simulink/sfg/s-function-concepts.html?requestedDomain=www.mathworks.com>][Mathworks note]] on the precise semantics.

### 1.2.3. Dshell Assemblies

#### 1.2.3.1. Assemblies creating signals



TBD: update and clean up this section

(from Abhi's draft notes)

- the real reason for having an assembly to have an addSignal() call for each of its signals is so that it can run as a stand-alone assembly for unit testing which will ensure that it has no external dependencies. This is the only requirement. We will refer to any signal with an addSignal call within an assembly as being **owned** by the assembly.
- The owned signals (i.e. those with addSignal calls) define the interface for an assembly. These are the only signals that can be tied at its level (by regular or interface ties). All other ties are ignored and simply fall through.
- An assembly only has two categories of owned signals: **model signals** use to tie to the models created by the assembly, and **connector signals** meant to be used to connect models from different children assemblies. A signal can belong to one or both categories. Though by and large, the model signals are consumed locally, while the connector signals are added to the assembly's pending ties, or directly to children's signal ties.
- For the purposes of the discussion below, we assume that we have 3 levels of assemblies denoted A→B→C.
- If an assembly uses the signal to tie to a model it creates, then it **must** have an addSignal() call for it to work
- However, if the signal is really only being used by C level sub-assemblies, and passed on as a signal tie from A via B, then there is really no need for an addSignal call within B since B can run fine stand alone even though it does not need to create the signal. The C level assembly who actually ties the signal to a model will have to however create and own it.
  - one exception is if the higher level assembly B wants to use a different name for the signal (for it to be more reflective of its use) than the name in the lower level C assembly. Then it B must have an addSignal call and own it as well.
  - this is also the case when we want to have a composite signal at B's level, and want to pass on a slice as a tie to C. Then we must have an addSignal call for it at the B level for it to own it as well.
  - So far the principle seems to be that pure feed thrus from A to C that are not used at the B level do not need to have an addSignal call within B, i.e. B does not need to own the signal. Anything that mutates the B's signal into C however requires B to own the signal and have an addSignal call for it (i.e. if there is mismatch between B and C's view of the signal then we need a ties and both need to own their versions).
- **Problem statement:** So, thus when B is a serial link assembly for a robotic manipulator, decides to pass individual signal ties down to the C lower level motor assemblies so that they are accessible to the user at the B level, what do we do? If this is the only way that the B arm assembly will be used, it is preferable that the user just provides accessor methods in B for the C level signals instead of passing down signal ties.
  - However, what happens the day someone wants to attach an arm controller or pose manager for the arm, so that these signals need to be exposed to the common A parent assembly for the B arm and the controller assemblyies. Clearly A must own the signals and have addSignals calls. How does A pass the signals down through the B arm assembly to the C motors without us insisting that the B arm assembly have addSignals calls for signals as well? The point is, that these are run-time ties that can change from run to run and context to context. How do get the B intermediate assembly to know to pass on these signal ties to the C level assemblies? In the current paradigm, we would need the B intermediate assembly to be knowledgeable about these signals so that (a) it knows to put them in the signal ties of the C child assemblies, and (b) and therefore B would need to have addSignals call for them and own them even though it has no direct use for them.
  - If we take this far enough, and we want to cover every possible use case, then each assembly must have addSignals call for and own every signal contained at every level of its sub-tree to cover every possible use! Not a great idea.
  - **Option A:** mske use of structural and interface ties
    - Thus clearly we need to make a distinction between **structural** signal ties meant for the child's immediate use (these are hard-coded in), and **interface** one's that are for the whole subtree to use (these are defined at run-time by the context). We call the first **structural ties** and the latter **interface ties**.
      - **structural ties:** These are hardcoded ties between signals owned by a parent to signals owned by a child. Ownership is required since otherwise some of the signals may not exist in certain situations making stand alone testing impossible.
      - **interface ties:** These are signal ties that change from context to context. Assembly B may get from its parent A, and it uses them as well as passing on to its children C for use. This is like the pending signal ties idea being used by the assembly builder. Each B assembly uses the combination of the input structural and interface signal ties to use within itself to alias out its signals as requested. For its C children assemblies
        - the structural signal ties for C are hard-coded in as before.

- The interface signal ties passed by B to C contain all the input interface signal ties, minus the entries that were used by B itself. This way the interface signal ties keep getting passed down until they are fully consumed by the lower level assemblies. B can add its own pending ties to the list passed to C.
    - So what happens when assembly B has a structural signal tie for C's signal, but also gets an alias request for C's signal via the interface signal tie passed from A to B. This should be an **error**. A should change the interface signal ties to B to refer to alias B's signal rather than to C's signal to avoid such conflicts. The principle is to tie to the highest level possible in the assembly hierarchy.
    - **Remove feed thrus?:** Feed thrus are signals in B that we want to alias to signals with the same name in the child C. There is normally no need to do this aliasing unless there is a pressing use for the signal to be owned at the B level. Most likely it is not needed once we introduce interface ties.
      - In the past we have had to use feed thru because we have not had the notion of the interface signal ties.
  - **Needs:** We need the following methods:
    - Add a `interface_ties` argument to Assembly constructors. The parent assembly will mark all the passed on interface ties as having been consumed.
    - get rid of the notion of feed thru and the `updateTies()` method. Now structural ties should be manually created.
    - `if_ties = self.childInterfaceTies(structural_ties, input_if_ties)`: this returns interface ties that consist of the input interface ties minus the interface ties consumed by the assembly itself. This method is used to get interface ties to pass onto children assemblies. Why do we need the structural ties argument? Do we need an additional argument for the assembly to add its own interface ties?
    - `ties = self.unprocessedTies()`: This will return ties consisting of the `childInterfaceTies()` minus the ones that have been passed onto children. This method is invoked by an assembly's `lockObject()` call to ensure that the return value is empty.
    - Add checker about proper ownership of the signals in the structural ties entry. Add warning when feed thrus are detected in the structural ties.
  - **Impact/Transition:**
    - all assembly constructors will have a new `if_ties` argument. (Not backwards compatible unless we put this at the end).
    - the existing signal ties argument will be treated as a structural ties and the interface ties will be empty. Thus things should continue to work.
    - Update assembly `lockObject()` to check that there are no unprocessed ties.
    - Down the line, get rid of the use of `updateTies`. Treat the second argument specific ties as being structural ties for the child. Get rid of feed thru.
  - **Audit:** We need to catch typos etc that can lead to missed or incorrect connections. Some ideas:
    - **Ownership:** Check that for any structural tie passed between parent/child assemblies, that one side is owned by the parent and the other by the child.
    - **Structural/Interface overlap:** Check that a signal already being tied structurally is not also being tied via an interface tie.
    - **Unprocessed ties:** As the pending ties are passed down, we need to ensure that they all get consumed by some assembly in the hierarchy. One way to do this is for each assembly to keep track of the pending interface ties (i.e. input ones minus the ones it consumes, and minus the ones it passes it on to children). During assembly locking, there should be a error if this is non-empty.
    - **Unneeded structural ties:** How do we discover unneeded structural ties? Clearly feed thru should be remove. Rather, a better check is if the left and right side names are the same in structural ties. That means it is unneeded.
  - **Questions:**
    - should the interface ties entries use qualified names for the signals?
    - when A wants to structurally alias to a signal in C, should B own the signal as well or should we let structural ties be handed down recursively as for interface ties?
    - why cannot structural ties be combined with interface ties? why do they need to be distinct. What if we combine them, and pass down remaining signal ties to the child assemblies?
- **Option B:** treat all signal ties like current pending signal ties
  - Like now, each assembly passes signal ties to its child. The child consumes some of the ties, combines the remaining ones with some new ones of its own and passes them down to its children - and so on. This avoids the need for feed thrus since unconsumed ties are automatically passed down to children.
    - This way assemblies are written to locally just create signal ties that they know about to pass to their children. However when the context is different, an ancestor assembly can pass in signal ties to override what this assembly is doing, or pass in addition ones to pull up and expose low level signals to the higher level.
    - This option does not require a change to the assembly constructor API.
    - The `addSignal` methods used passed in signal ties if provided. The locally consumed ties are filtered out from the signal ties stack. Local ties are added to the remaining ones and provided to child assemblies.
    - This approach also allows both hard-coded ties within assemblies, as well as the passing in of different ties at the top-level.
  - **Difficulty with Option B:** All ties are passed in via the signal ties argument. The pending ties that are given to the children are the unconsumed ones. The way they get consumed is via the `addSignal` calls in the assembly. How does one bootstrap the process? While we are able to propagate pending ties, how do we add new signals to pending ties? Add a method for it?
  - **Needs:**
    - `updateTies()` works as before, except that it ignores, that have been consumed, adds in local ties, and treats the feed thru ties as if they are clones (like now). So the only change is to add in the ties that were passed in, and were also not consumed (i.e. are in the `_tied_signals` map).
    - `consumed_ties = self.consumedTies()`: This will return ties consisting of consumed input ties by the assembly, i.e. the locally consumed input ties (`_tied_signals`) and the input ties that have been consumed by the children assemblies (from calling their `consumedTies()` and filtering out for the input ties only).
      - This method will throw an exception if any of its children have not consumed all of the local ties given to it.
    - `unconsumed_ties = self.unConsumedTies()`: This will return ties consisting of unconsumed input ties by the assembly, i.e. the inputs signal ties minus the locally consumed ties (`_tied_signals`) and the ones that have been consumed by the children assemblies (from calling their `unConsumedTies()`).
      - This method will throw an exception if (`children_unconsumed_ties - input signal ties`) is non-null. The idea is that at every level, the only possible unconsumed ties are the ones passed in (since they might be consumed on a different branch).

- This method is invoked by the simulation's lockObject() call to the top level assembly. An exception will get thrown if there are locally generated ties that are not consumed within the sub-tree. The top level assembly should return an empty map.
- **Impact/Transition:**
  - Change updateTies() to include unconsumed ties in the returned signal ties.
  - Update Simulation lockObject() to call topasm.unConsumedTies(). This should not throw an exception and should return an empty list.
  - May be able to get special handling in builder, and the notion of pending signal ties.
- **Audit:** We need to catch typos etc that can lead to missed or incorrect connections. Some ideas:
  - **Ownership:** There is no ownership check. Any unconsumed ties are assumed to be for further down the hierarchy. So typos etc will result in some ties never getting consumed, leading to errors during locking.
  - **Overriding ties:** We can have a case where B has a tie for C in its local ties. However, A may pass in a tie to B for the same signal. In this case updateTies replaces the local tie with the one from A. Typos, will lead to non-replacement and again alerts during locking.
  - **Unconsumed ties:** We need to ensure that any local time passed to a child assembly get consumed to catch typos and tie errors. This gets tricky if what we pass to the child is one ties argument with the local ties and unconsumed parent ties mixed in. It becomes difficult to handle the situation when there is an overlap between the local ties and parent ties - since we cannot easily tell when an unconsumed tie is a local tie (which would be an error) or it came from the ancestor (which would be fine).
    - An option is to only pass local ties to the child. The child can figure out the unused ancestor ties by taking the difference between the parent's input ties and the parent's local ties. This can be handled within the addSignal method.
    - One implication of this would be that the signalTies() method would only contain the local ties passed in by the parent assembly (and not the feed thru from the ancestor). Thus signal ties would contain just the 'structural' ties, while the 'interface' ties would be computed.
    - Another implication would be that an assembly's tied signals map could contain more entries than those passed in via the signal ties (since it can also contain entries from the pending ties from the parent)
  - **Unconsumed ties (OLD):** The call to consumedTies() at any level will throw an exception if any local created ties are not consumed by the children assemblies.
  - **Unneeded addSignals?:**
    - **Bad pending additions:** Make sure that we cannot put fake pending ties into only an assembly. Should not be possible if we use updateTies(). In any case, the slices on the right need to exist.
    - **Catch no updateTies use:** Assembly's should use updateTies() to compute ties for children so that unconsumed ties get added in. The reason is that any input signalTies() will not get passed onto children if the assembly is directly computing the children's signal ties.
      - To check this, at each assembly we should get the list of unconsumed ties, and verify that this list is contained in the list of signal ties for each child assembly, else generate a warning.
    - **Unneeded ties:** How do we discover unneeded structural ties? What we want to find is ties from A that are essentially feed thrus (i.e. left and right side names are the same) and ones that are consumed by just one child assembly. These represent unneeded feed thrus. We should only have such feed thrus if there is another child assembly that is using the same signal.
      - we are looking for ties (where left name = right name) that are locally generated (i.e. not passed in), and which was consumed by only one child assembly.
      - thus call to updateTies() should add to a list of local\_ties. The difference between child.consumed() ties and the local ties gives the list of local ties consumed ties by the child. We need to keep a count of the number of children consumers for each local tie. After polling all the children, create a warning about unneeded feedthru if any of the local ties is consumed by just one child, and is one where left name = right name. Note that the check that the number of consuming children is 0 is equivalent to the check done by unConsumedTies.
  - **Missing addSignal calls:** do we have a way to catch missing addSignal calls?
    - a tie consumed by more than one child assembly's sub-tree should be owned by the assembly
    - a model tie should be owned by the assembly
  - **addSignal at correct level?:** For every signal, show use its ties
  - **Missing signal ties:** do we need a way to catch ties that are in pending ties instead of signal ties?
    - if the right side signal slice belongs to the parent assembly, then the tie should be a signal tie, and not a pending tie.
- **Questions:**
  - Say we have a case where B has a local tie key which is also in the input signal ties. This is often the case when B needs to share its signal with multiple children, but A wants to own the signal at its level. But there is technically the possibility that B is using the key to point to a different signal than the one from A is pointing to. This should be fine, but when we are checking for consumed/unneeded ties, we should not just be focusing on the keys otherwise we will misread situations like this.
  - should the addSignal command add a 'tag' or a 'signalName' entry to the tied signal's map?
  - How do we target/uniquify where a pending tie gets consumed? Should we qualify the name; should we be listing all the targetted consumers of a pending tie?
  - Should we be calling pending ties bridge or connector or interface ties?
  - How do we handle the case where there is a key in both the local ties passed in and the pending list? Clearly, the local ties should take precedence since it is structural and the one in the pending list should be ignored. Right?
  - So we should indeed ensure that anything in signalTies has an addSignal within the assembly (since it is structural)
  - what should getSignalSlice() return - only stuff in signalTies?
  - Yes, only stuff from tied signals or locally created signals. The reason is that this method is used to get the slice to tie to models, and so has to work when the assembly is stand alone and there are no pending ties. So this method is restricted to the owned signals, i.e. those having addSignals call.
  - How to detect unneeded addSignals() calls?
  - Should the tied map be split up into ones for the structural and the ones for the pending.

- **Use cases:**

- B/C: C owns signals, B does not care.
- B/C: B has signal tie to alias out C's signal
- B/C: B has signal slice tie for C's signal
- A/B/C: A/B and B/C have ties - business as usual
- AD/B/C: A has a child D, and has interface signal ties for B and D
- AD/B/C: A has a child D and has interface signal ties for C
- AD/B/C: A has interface signal ties for C and B also structural one for C (conflict)
- A/B/C: A has interface signal ties for C and B also interface one for C (conflict)
- AD/BE/C: B has a child E, and adds interace ties to the ones from A while passing on to C.

### 1.2.3.1.1. Finding the signals belonging to an assembly

How to find out all the signals owned by an assembly?

- signalList() only returns the ones that are actually created
- how do we get a list of all the ones that addSignals was actually called for? There now appears to be a `_signals_made_with_add_signal` member which keeps a list of all the signal names that addSignal was called for. The `tag`'s table for signals only keeps track of signals that belong to the assembly. The `asm`'s `tiedSignalsTable()` keeps track of all the local signals that were aliased by signal ties.
- Thus the answer is that the signals owned by an assembly, the ones actually created are returned by `signalList()` or the keys to `signalsTable()`, and the ones aliased out by ties are keys to the `tiedSignalsTable()` map. So we really do not need the `_signals_made_with_add_signal` member, do we?

### 1.2.3.2. Assemblies creating bodies

The `addBody` method is used to add bodies to an assembly. If there is a body with the same name in the given scope (see below for more details) then that body will be returned, otherwise, a new body with the given name will be created and returned. The method is designed to be called using the following:

```
addBody(self, bodyName: str, parent: "DartsBody" = None, tag: str = None)
```

PYTHON

When the method searches for an already existant body with the name given by the `bodyName` argument, it refines the scope of its search using a root body. This root body is taken as one of the following (in order of preferences):

1. The parent keyword argument.
2. The `ANCESTOR_BODY` context field; `ANCESTOR_BODY` is specified in `DshellCommon.Constants`.
3. The multibody virtual root.

The following flowchart shows the information above graphically:

```
skinparam backgroundColor transparent
@startuml
start
if (parent body is specified?) then (yes)
    :root body = parent body;
    stop
else (no)
    if (ANCESTOR_BODY context field is specified?) then (yes)
        :root body = ANCESTOR_BODY;
        stop
    else (no)
        :root body = mbody virtual root;
        stop
    endif
endif
@enduml
```

PLANTUML

If a body is found, then that body is returned. Otherwise, a new body is created whose inboard body is the root body found earlier.

Before returning the aforementioned body, the body is registered to the simulation with a tag. The tag is one of the following (in order of preference):

1. The tag keyword argument
2. The bodyName argument

The following flowchart shows the information above graphically:

```

skinparam backgroundColor transparent
@startuml
start
if (tag keyword is specified?) then (yes)
    :tag = tag keyword;
stop
else (no)
    :tag = body name argument;
stop
endif
@enduml

```

The assembly registers the body, regardless of whether it was looked-up or created. The reason is that when a user calls the `addBody` method from an assembly, they expect the assembly to own the body, i.e., the body should be registered to the assembly.

#### 1.2.3.2.1. Legacy behavior

For backwards compatibility purposes, the function can also be called using the following:

```
addBody(self, parent: "DartsBody", bodyName: str, tag: str)
```

If the `DartsCommonMode` is `ndarts`, then the behavior of the method is the same as previously described. However, if it is `dartsp`, then the method will not search for a new body, it will only create one. Moreover, when in `dartsp` mode, the `ANCESTOR_BODY` context field is not used when determining the root body, and the tag must be supplied.

##### 1.2.3.2.1.1. Further resources

A full team talk on this topic can be found [here](https://dartslab.jpl.nasa.gov/technotes/Talks/2022-04-21-sprint-assembly-addBody.mp4) (<https://dartslab.jpl.nasa.gov/technotes/Talks/2022-04-21-sprint-assembly-addBody.mp4>). For examples of the `addBody` method, see the reg tests in `Dshell++/test/test_add_body`.

#### 1.2.3.3. Assemblies creation/recreation



TBD: update and clean up this section

For old abhi's notes

- we currently have multiple distinct ways of creating a simulation and the computation graph
  - via assembly builder
    - calls `SimExecNdarts.createAssemblies(asm_info_dict, integ_options)`
      - calls `SimExec.createAssemblies(asm_info_dict)`
        - calls `topasm.addAssembliesFromAsmInfo(asm_info_dict)` to create assemblies
        - calls `setupDynSolver(integ_options)` to create cegraph, dynsolver, integrator and bindParams
    - via ROAMS
      - `SimExecRoams` & `SimExecRoams2` constructors call `_createSolver`
        - calls `setupDynSolver(integ_options)` to create cegraph and call bindParams
          - it calls `createDynSolver(integ_options)` to create cegraph, dynsolver, integrator
            - attach mbody root frame if unattached
            - call `createCompGraph` to create cegraph
              - creates a **constrained subgraph** and returns it
            - create & set right type of **dynSolver** based on integrator type
            - create right type of **integrator** and set it in `dynSolver`
          - calls `bindParams` for the top level asm
        - calls `resetState` for all assemblies
        - sets integrator tolerances
        - **addRover** to add a new rover
          - calls `addAssemblies(lasms)` to recreate cegraph if bodies have changed
            - if no solver calls `createDynSolver()` to create cegraph, dynsolver, integrator (see above) with default integ options
            - checks if bodies have changed via lasms and sets flag
            - if bodies have changed call `teardownCEGraph` for all assemblies
            - calls `_setupSolver(lasms, bodies_changed)` to recreate cegraph
              - if bodies changed, call `recreateCompGraph`
                - calls `createCompGraph` to create new cegraph
                - changes `dynSolver`'s sugraph to be new cegraph
                - calls `deleteCompGraph` to delete old cegraph
                  - changes `dynSolver`'s sugraph to be mbody



- calls **\_deleteCompGraph** to delete the cegraph
      - for each body in the cegraph, reset masking interface and removes the body from the cegraph and any associated aggreg. subgraphs (why is this not done by the teardownCEgraph call????)
      - deletes the cegraph object using del
    - call **bindParams, bindState, resetState** for new lasms
    - if bodies changed, call **updateCEGraph** for all asms
    - calls **lockObject**
  - in script
    - tend to call createAssemblies which does the setup work
- **PROPOSED CHANGES**
  - Update setupDynSolver so that after the bindParams call, updateCEGraph is called for all assemblies
    - the reason for this is that currently if we were to add one of the CE assemblies (eg. double wishbone), the compound body would never get set up!
  - We want to clean things up so that the sim.lockObject() will do all the work of recreating the comp graph if bodies have changed. The idea is that each assembly should set the bodies\_changed flag each time a body is added by an assembly. Within lockObject's Python prewrapper, the cegraph will be recreated via a call to renewCompGraph() (see below) if bodies have changed.
  - Rename \_setupSolver to renewCompGraph. This method should only be called if bodies have changed (so remove the bodies\_changed argument to it)
    - move the bindParams method out of \_setupSolver
  - Get rid of sim.addAssemblies method
  - Make it the user's responsibility to call bindParams, resetStates() & bindState after an assembly is added post-initialization
  - add call to resetStates() after bindParams in setupDynSolver
  - get rid of \_setupSolver method
  - update Assembly.setup method to automatically detect if the number of bodies has changed after a call to addAssemblies or addModels and use this to set a flag in sim object.
  - setupDynSolver should not create cegraph - this will be done by lockObject
  - **End result**
    - During initialization, will call setupDynSolver as now. Roams will set integrator tolerances explicitly after that
    - within the sim.lockObject() call, if there is a cegraph, check if bodies have changed, delete the cegraph. If no cegraph create it and call updateCEGraph for all assemblies.
    - adding assemblies after initialization will require user to call bindParams, bindStates, resetStates themselves
    - we can then get rid of the addAssemblies, \_setupSolver, \_createSolver, and all the methods they call.
    - these changes will simplify things, but will make Dshell only work with Ndarts and will break compatibility with Darts++
    - The subgraph's CM frame is now attached to its virtual root instead of the basebody by default

## 1.2.4. Dshell Models

### 1.2.4.1. Model Types



TBD: Add descriptions for other types of models

#### 1.2.4.1.1. ContinuousModel

- This class is for a model that may participate in continuous dynamics (eg. has preDeriv/postDeriv functions and may have continuous states) but would not require a node or body to attach to. This would be useful for operational models that might need to perform activities in preDeriv/postDeriv calls, provide multi-rate EndIOStep events, have continuous states, etc, but are not classic models (Sensor; Actuator; Motor; Encoder).
- The issue is not whether a model itself has continuous states, but rather whether its input or output needs to be updated as part of the numerical integration. Thus even though a thruster does not have continuous state, we implement it as a continuous model so that the force values can be changed based on state. Gravity models are a good example of a model that does not have continuous states, but effects continuous dynamics.

#### 1.2.4.1.2. DartsModel

- These are derived from ContinuousModel but have access to the mbody() instance (but not nodes or bodies). Also could have a flow variant that does not even have mbody() access but has continuous state methods.

#### 1.2.4.1.3. FlowModels

- Flow models have very limited functionality. Essentially they have tick() methods that are called each default IOStep.
- The tick() function is essentially equivalent to the EndIOStep() callback. Perhaps one method could call the other:
  - Restructure the model hierarchy.

### 1.2.4.2. Models method guidelines



TBD: update and clean up this section

- only updateFlowOuts should be setting flowOuts - none of the other methods should

- this method should be callable independently, so that all computations needed to set the flowOuts should be within the method
- if the model sets any values in the mbody (eg. forces, accels) that effect the multibody dynamics, then these computations must go into preDeriv
- if the model's continuous state derivatives depend on mbody derivatives, then these computations should go into postDeriv. Otherwise, they can be in either pre or postDeriv. Indeed, this means that as a rule of thumb all continuous state derivatives can be safely put into postDeriv.
- if the flowOuts do not depend on the mbody derivatives, then their granularity should be PREDERIV. If they do depend on mbody derivatives, then granularity should in addition also be POSTDERIV.
- if the flowOuts do depend on mbody derivatives, then their granularity should be set to POSTDERIV. If their output goes into the input of another model there is a potential loop between these models and DARTS. A break and constraints may be required from the user.
- beginIntegrationStep should be used to update discrete states across the integration interval
- there is no need for BEGININTEGRATION or ENDIO granularities
- endIntegrationStep() should normal be empty (unless it is changing non-traditional mbody inputs such as body mass which can effect CM calculations done downstream by some other model. Of course these computations should also be done in preDeriv then.)
- endIOStep() should be empty.
- beginIOStep() should be empty unless some pointers etc need to be one time initialized.
- models do not know how their outputs will get used. So any time there is a loop calling model methods (preDeriv, endIntegrationStep etc etc) we need their inputs to be correct. Hence every model's updateFlowOuts should be called right before such loops by default. In fact, the default granularity should be BEGINIO+PREDERIV+ENDINTEGRATION. POSTDERIV should only be added as needed.
- Removing PREDERIV or ENDINTEGRATION granularity is a way of coarsening the output computations for speed, and is entirely left to the user.
- Should we allow updateFlowOuts to check the current calling level for the model so that they can change their computations depending on whether the model is at preDeriv, endIntegration etc levels.
- Remove preDeriv method from Sensor and Encoder models.
- Create table which summarizes what the rules for each method are
- **loops with DARTS**
  - what is the loop when a model's input depends on Udot?  $u_m(Q, U, Udot, T)$  in general, though typically just  $u_m(Q, U)$ .
- **autocode generator**
  - if there are no flowOuts, do not generate updateFlowOuts stub
  - if there is no  $g_m$ , i.e. no inputs to mbody, then do not generate preDeriv stub
  - if there are no continuous states, skip postDeriv stub
  - if no params, skip processParams
  - if no states, skip beginIntegrationState

### 1.2.4.3. Models using objects as parameters



TBD: update and clean up this section

From abhi

- Object passing issues
  - Having trouble with CombinedModels
    - Due to virtual inheritance, only the concrete classes should be calling the constructor for DshellBaseIF (see [\[link\]](https://www.cprogramming.com/tutorial/virtual_inheritance.html))
      - So the Frame class should be calling DshellBaseIF constructor
      - But what about the DartsNodeBase and DartsBody concrete classes which also derive from the Frame class. We have a diamond patter here, where these classes also derive from DshellBaseIF. So should they or should they not be calling DshellBaseIF constructor? (Frame is already calling it)
    - This test breaks when changing the Frame param at run time
      - the param changing works when passing in another Frame, however breaks when passing in a node or a body
      - Is this a DartsBaseObject problem - or one to do with object passing?? Create a DartsBaseObject by itself to check this. DONE - no problems at this level.
      - For the non-Frame values, we are finding that the DVar is changing the pointer value - which apparently is OK, since this is ok when up & downcasting classes with multiple inheritance
        - is this a SWIG issues? Can we create a pure C++ example?
        - the Frame class does not have diamond inheritance issue, is this why it works?
        - is the issue to do with mismatch between expected value, and a different passed value (node, body)?
    - Observations
      - DartsBaseObjectLeaf takes pointers of type DshellBaseIF (not NdartsBaseObject).
      - DshellBaseIF is all that models needs to now to get as objects. So we want to keep it minimal. NdartsBaseObject has a lot of stuff in it which is unnecessary for DScene objects or for Models. Hence we keep them separate.

- The upCast method for DshellBaseIF needs to work for all classes - but DshellBaseIF does not have typeString method. So we need to check and invoke different and appropriate upCast methods
- The fromDictionary method needs to handle all DshellBaseIF instances since DVar params are the way communicate with Model params.
- When I hack the DartsBaseObject's value() method to cast the in value to a NdartsBaseObject, the Dshell models works, i.e. the model's param value is getting set properly. However querying the spec value back breaks
- Are we getting multiple vttables for DshellBaseIF. Use gdb's 'info vtbl' command.
- Better issue - Found that if we take the set param value, this pointer is different from what we would obtain by static casting the raw param value to DshellBaseIF, and then dynamic casting back to a NdartsBaseObject/Frame!!!!
- Found a work around - change param value to the value obtained by the double cast to DshellBaseIF followed by Frame cast!!!
  - however this screws up the spec, so that calling value() on it gives back a bad result
  - we can get back the raw pointer from the derived value by casting back to DshellBaseIF
- Why is the IF.cc while using reinterpret cast when creating the param spec leaf?
- Who all should be calling DshellBaseIF constructor
- New design
  - decouple the spec buffer from the element used in the model. So use xframe\_buf for the DshellBaseIF raw buffer registered with the buffer.
  - When a value is set, the checker should use dynamic\_cast to verify that it is valid, and then assign this cast value to the Frame xframe member within the class.
- Jonathan
  - who all's constructor is calling DshellBaseIF()
  - why are we using reinterpret\_cast in IF.cc file?
  - create a separate buffer for the object spec to use

#### 1.2.4.4. Multi-rate models



TBD: Flesh out this section

All models can be multi-rate.

#### 1.2.4.5. modelLinker

The `DshellCommon.BaseParam` class contains a static method called `modelLinker` that can be used to easily populate a parameter class's data fields using a model's parameters, and set a model's parameters using the data fields of a parameter class instance. For more information see the [DshellCommon documentation](#)

(<https://dartsgitlab-internal.jpl.nasa.gov/darts-lab/jpl-internal/all-access/development/simulation-framework/dshellcommon/-wikis/DshellCommon-Documentation#user-content-modellinker>).

#### 1.2.5. Linearization



TBD: only partially complete for now

##### 1.2.5.1. Variable-step finite difference

By default, the step size used for numerical differentiation in linearization is governed by `_eps`. However, in some cases, we may want to vary the step size based on the magnitude of the state or input being perturbed. In these cases, each column of the A, B, C, and D matrices would have its own step size. Our approach is based on what is done in `HC2CoaxialHelicopter/Helicopter/Analysis/analysis.py`, which Havard says was chosen based on Matlab's linearization.

To accomplish this variable step size, we add in a new variable `_eps1`, which has a default value of zero and can be set using the `set_eps1` function. The function throws an error if it is used while we are using one of the Richardson extrapolation methods, because we think those methods are not compatible with using a variable finite-difference step size.

While we are calculating the effects of perturbations, we perturb the current state or input by the `eps` that was passed in (which may be `_eps`, `-_eps`, `2*_eps`, etc. as governed by the finite differencing scheme), plus a variable component based on the state or input's magnitude, which is `_eps1 * eps * abs(<current state or input>)`. We also take care of the division-by-step-size part of taking a finite difference at this point, dividing the matrix column by the absolute value of the size of the step we took. The rest of the finite difference math is done later, once perturbations by each `eps` required for the finite differencing scheme have been completed.

##### 1.2.5.2. Finite difference methods

This section describes the finite difference methods used by DARTS linearization to estimate derivatives. For each method, this document describes how the derivative is estimated, what the error in the derivative is, and what the minimum step size should be.

Note that all errors are given as an order of magnitude, and the exact error value will differ based on the function,  $f$ , whose derivatives are being approximated. As a result, the `re1_err` variable used to set tolerances in the simulation linearization is not exact (e.g. if `re1_err = 1 × 10-10`), it is possible that the result has an error larger than  $1 × 10^{-10}$ ). This possibility is exacerbated as the step size, which is calculated based on `re1_err`, approaches the minimum step size, because the minimum step size itself is merely an order of magnitude estimate.

#### Nomenclature

Let a derivative with respect to the variable  $x$  be denoted by a prime (i.e.,  $\frac{df(x)}{dx} = f'(x)$ ). Let a hat denote a quantity estimated by a finite difference. For example, if the true value is  $f(x_0)$ , then the finite-difference approximation is  $\hat{f}(x_0)$

#### 2 Point Central Difference

How it is calculated:

$$\hat{f}'(x_0) = \frac{f(x_0 + h) - f(x_0 - h)}{2h}$$

**The error in the method:**

$$f(x_0 + h) = f(x_0) + hf'(x_0) + \mathcal{O}(h^2)$$

$$f(x_0 - h) = f(x_0) - hf'(x_0) + \mathcal{O}(h^2)$$

Taking the difference between the two equations above and simplifying yields,

$$f'(x_0) = \frac{f(x_0 + h) - f(x_0 - h)}{2h} + \mathcal{O}(h^2).$$

Therefore, the error in the method is  $\mathcal{O}(h^2)$ .

**Minimum step size:** Suppose that all calculations have a round-off error of  $\delta$  and  $|\delta| \leq \epsilon$  where  $\epsilon$  is the machine-level precision. Then, the error,  $e$ , in the method can be bounded by,

$$e = |f'(x_0) - \hat{f}'(x_0)| \leq \left| f'(x_0) - \frac{f(x_0 + h) - f(x_0 - h)}{2h} \right| + \left| \frac{\delta(x + h) - \delta(x - h)}{2h} \right|.$$

As shown previously, the order of error in the method is  $\mathcal{O}(h^2)$ . Therefore, the error is  $ch^2$  for some constant  $c$ . In addition, the round-off error is bounded by  $e$ . Therefore,

$$e \leq ch^2 + \frac{\epsilon}{h}$$

Differentiating  $e$  with respect to  $h$  and setting the result equal to zero yields,

$$h_{min} = \sqrt[3]{\frac{\epsilon}{c}}.$$

Note that the constant  $c$  in the above equation is different than the original constant. Moreover, the constant will be different for each function  $f$ ; however, we can approximate the order of magnitude of  $h_{min}$  as,

$$\mathcal{O}(h_{min}) = \mathcal{O}\left(\sqrt[3]{\frac{\epsilon}{c}}\right) = \mathcal{O}(\sqrt[3]{\epsilon})$$

#### 4 Point Central Difference

**How it is calculated:**

$$\hat{f}'(x_0) = \frac{f(x_0 - 2h) - 8f(x_0 - h) + 8f(x_0 + h) - f(x_0 + 2h)}{12h}$$

**The error in the method:**

$$f(x_0 \pm 2h) = f(x_0) \pm 2hf'(x_0) + 2h^2f''(x_0) \pm \frac{4h^3}{3}f'''(x_0) + \mathcal{O}(h^4)$$

$$f(x_0 \pm h) = f(x_0) \pm hf'(x_0) + \frac{h^2}{2}f''(x_0) \pm \frac{h^3}{6}f'''(x_0) + \mathcal{O}(h^4)$$

Taking a linear combination of the above equations such that  $f(x_0)$ ,  $f'(x_0)$ , and  $f''(x_0)$  are eliminated, and simplifying yields,

$$f'(x_0) = \frac{f(x_0 - 2h) - 8f(x_0 - h) + 8f(x_0 + h) - f(x_0 + 2h)}{12h} + \mathcal{O}(h^4)$$

Therefore, the error in the method is  $\mathcal{O}(h^4)$ .

**Minimum step size:** Suppose that all calculations have a round-off error of  $\delta$  and  $|\delta| \leq \epsilon$  where  $\epsilon$  is the machine-level precision. Then, the error,  $e$ , in the method can be bounded by,

$$e = |f'(x_0) - \hat{f}'(x_0)| \leq \left| f'(x_0) - \frac{f(x_0 - 2h) - 8f(x_0 - h) + 8f(x_0 + h) - f(x_0 + 2h)}{12h} \right| + \left| \frac{\delta(x - 2h) - 8\delta(x - h) + 8\delta(x + h) - \delta(x + 2h)}{12h} \right|.$$

As shown previously, the order of error in the method is  $\mathcal{O}(h^4)$ . Therefore, the error is  $ch^4$  for some constant  $c$ . In addition, the round-off error is bounded by  $c$ . Therefore,

$$e \leq ch^4 + \frac{18\epsilon}{12h}.$$

Differentiating  $e$  with respect to  $h$  and setting the result equal to zero yields,

$$h_{min} = \sqrt[5]{\frac{\epsilon}{c}}.$$

Note that the constant  $c$  in the above equation is different than the original constant. The constant will be different for each function  $f$ ; however, we can approximate the order of magnitude of  $h_{min}$  as,

$$\mathcal{O}(h_{min}) = \mathcal{O}\left(\sqrt[5]{\frac{\epsilon}{c}}\right) = \mathcal{O}(\sqrt[5]{\epsilon}).$$

## Richardson Extrapolation

**How it is calculated:** Richardson extrapolation can be used on any finite difference method to further reduce the error. To generalize, let the finite difference method for a given step size  $h$  be given by  $D(h)$ . Then, the Richardson extrapolation method is given by,

$$\tilde{f}(x_0) = \frac{\alpha^n D(\frac{h}{\alpha}) - D(h)}{\alpha^n - 1}.$$

**The error in the method:** For a given step size  $h$ , the original method has the error;

$$D(h) = \tilde{f}(x_0) = f'(x_0) + ch^n + \mathcal{O}(h^{n+1}),$$

for some unknown constant  $c$ . Similarly, for a different step size  $\frac{h}{\alpha}$ , where  $\alpha$  is some positive real number,

$$D(\frac{h}{\alpha}) = \tilde{f}(x_0) = f'(x_0) + c(\frac{h}{\alpha})^n + \mathcal{O}((\frac{h}{\alpha})^{n+1}).$$

Richardson extrapolation combines these two estimates as follows,

$$R(h, \alpha) = \tilde{f}(x_0) = \frac{\alpha^n D(\frac{h}{\alpha}) - D(h)}{\alpha^n - 1} = f'(x_0) + \mathcal{O}(h^{n+1}).$$

Therefore, the error in the method is  $\mathcal{O}(h^{n+1})$ : an order of magnitude more accurate than the original method!

**Minimum step size:** Richardson extrapolation uses a linear combination of the original methods. Hence, the minimum step size used should be the same as that of the original method.

### 1.3. Usage

### 1.4. Software

### 1.5. Raw documentation



TBD: Need scrubbing before integration.

#### 1.5.1. DshellCommon: Split Assembly into BaseAssembly and Assembly



TBD: Needs scrubbing. Notes brought over from [issue](#)  
([https://dartsgitlab-internal.jpl.nasa.gov/darts-lab/jpl-internal/all-access/development/simulation-framework/dshellcommon/-/issues/30#note\\_9372](https://dartsgitlab-internal.jpl.nasa.gov/darts-lab/jpl-internal/all-access/development/simulation-framework/dshellcommon/-/issues/30#note_9372)).

The old `Assembly` class has been split into `_Assembly_Cpp`, which contains the essential methods for an assembly, and `AssemblyLegacy`, which imports from `_Assembly_Cpp` and adds the methods missing from `_Assembly_Cpp`. Thus, `AssemblyLegacy` is equivalent to the old `Assembly`. Importing `Assembly` from `Dshell.Dshell_Py` is equivalent to importing `AssemblyLegacy` from `DshellCommon.assemblies.AssemblyLegacy`.

Moreover, `DAssembly` has been created, which imports from `_Assembly_Cpp`.

See [https://docs.google.com/spreadsheets/d/10nJ2E0wQqg5aDPqvdwz1v9\\_wrDZtZ78602HBwzmQI\\_4/edit#gid=0](https://docs.google.com/spreadsheets/d/10nJ2E0wQqg5aDPqvdwz1v9_wrDZtZ78602HBwzmQI_4/edit#gid=0) to know what methods are only available in `LegacyAssembly` (marked with "Assembly only") and what methods are available in `_Assembly_Cpp` and thus in both `LegacyAssembly` and `DAssembly` (marked with "BaseAssembly (both)").

In the distant future when `AssemblyLegacy` are made obsolete, we may rename `_Assembly_Cpp` to `Assembly` so that the SWIG wrapper and C++ classes have the same name.

#### 1.5.2. Proposal: New state propagation design

With the introduction of support for multi-rate mode, event scheduling, zero-crossing events, etc. capabilities it has become clear that the current state propagation design - based on the notion of `simulation steps` is outmoded and overly complex. This section proposes a cleaner and simpler design. Specific actions are being tracked in the [gitlab issue](#)  
(<https://dartsgitlab-internal.jpl.nasa.gov/darts-lab/jpl-internal/all-access/development/simulation-framework/dshellpp/-/issues/23>).

##### 1.5.2.1. Overall approach

The new paradigm focuses entirely based on state propagation from `hop` to `hop`. `hop` intervals are defined by the integrated timeline of `hop barriers` derived from model periods, events, and user requests. Now that `hop barriers` can be `explicit` and `implicit`

- `explicit` `hop barriers`:: These are defined as one where the time of occurrence is known, such as ones defined by model multi-rate periods, time-based events, user-specified time advancements.
- `implicit` `hop barriers`:: These barriers are usually defined by `zero-crossing` events, where the `hop` is required to end exactly when a specified condition is met, such as impact with the ground, certain load conditions, etc. The exact time of occurrence is not known, and has to be sought out using an iterative procedure.

A key assumption regarding multi-rate models is that:

“While the whole system state evolves together over `hop` intervals, the `startIOStep()` and `endIOStep()` methods for a model only get called when the simulation is at `hop` boundaries matching the multi-rate period request for the model.

The following `Simulation` methods are proposed for state propagation:

- `_advanceOverExplicitHopInterval()` :: This method is the basic workhorse, and attempts to propagate the system state to the end of the next `explicit` hop barrier. We say attempt, because while this is the norm, the propagation may end sooner such as when we have a zero-crossing condition which when achieved, terminates the simulation prematurely (eg. impact with the ground) without propagating all the way to the requested end-time.
- `_run()` :: This method consists essentially of a `while` loop calling `_advanceOverExplicitHopInterval()` in each pass to advance the system state over the next hop interval. After each such pass, this method will call the set of user-registered `stopping` callback functions, and terminates the `while` loop if any of these function returns `true`. Moreover: all of the functions that return `true` are unregistered from the list of stopping functions. A rough prototype of this method is:

```

std::vector<std::string> Simulation::_run()
{
    /* check that there is at least one registered stop callback to
    avoid infinite loop */
    assert(_run_stop_cbss.size() > 0);

    bool stop = false;
    std::vector<std::string> stop_cb_ids;
    std::vector<std::size_t> stop_cb_indices;
    while (1)
    {
        /* advance the state over the next explicit hop interval */
        _advanceOverExplicitHopInterval();

        /* loop over all stop methods and see if they are
        * requesting a stop */
        size_t index = 0;
        for (auto it: _run_stop_cbss)
        {
            if (it->second())
            {
                stop = true;
                stop_cb_ids.push_back(it->first);
                stop_cb_indices.push_back(index)
            }
            ++index;
        }

        if (stop) break;
    }

    /* unregister the stop cbs that have triggered */
    for (auto it = stop_cb_indices.rbegin(); it != stop_cb_indices.rend(); ++it)
    {
        _run_stop_cbss.erase(_run_stop_cbss.begin() + *it);
    }
    return stop_cb_ids;
}

```

CPP

Our current simulation state propagation methods can be re-implemented using the `_run()` function as described below:

#### `advanceTimeBy(delta_t)`

This method will remain unchanged and will simply call `advanceTimeTo()` to do the work.

#### `advanceTimeTo(final_t)`

This method will register a no-op timed event for time `final_t` to create an explicit hop barrier, and will register a stopping callback function that returns true if the current simulation time is equal to `final_t`, and then call `_run()`. The `_run()` method will terminate when it encounters the stopping function registered by this method.

#### `step()` and `step(n)`

These methods will simply call `advanceTimeBy(n*_default_model_multirate_period)` method.

#### sim terminate methods

Fsm's often will set the `terminate` flag in the simulation. They can also register a stop callback that calls `Simulation::isTerminated()` to terminate the `_run()` loop.

#### clock classes

The `ROAMS`, `CADRE` etc classes call the `sim.step()` methods, and should remain unaffected.

Thus much of our current simulation state propagation can be achieved by using the `_run()` method in the new design. The current `_advanceTime()` method can be retired.

#### 1.5.2.2. Hop propagation

The state propagation over a hop consists of calling the `startHop` methods of all the `flow` models. When there are continuous states in the system (either from models or from the multibody system), one of the `flow` models is expected to be a state propagator flow model, (typically `DebugFlowAdvanceDynamics`) whose job is to call the `_advanceContinuousStates()` method to have the integrator numerically integrate and propagate the continuous states. The `_advanceHop()` and `_advanceState()` methods pretty much do what we need.

The proposed `_advanceOverExplicitHopInterval()` method can use much of this code as is with clean up, with the additional functionality for it itself to figure out what the next explicit hop barrier is to propagate the state over.

#### 1.5.2.3. Handling model time delays

There are multiple possibilities and options for handling time delays. The primary requirement here is that we should be able to insert a hop barrier corresponding to any such model time delay so that the model is then able to do its actual work at the right time.

Let us assume for now that the time delay for a model is explicitly known. This is likely to be the case in most circumstances, where we have a constant and known delay value, or the delay amount depends on the state. In either of these cases, the value is known even at the start of the I/O step and/or the hop. We have two paths that we can pursue:

1. The simplest approach would be for the model's `startIOStep()` method to register an explicit hop barrier based on the time delay for the model.
2. In this approach, the explicit time barrier would be registered by the `startHop()` method. The one problem with this approach is that the new time barrier may precede the end of the hop that is in progress. This situation can be handled using the available `step validation` capability. This feature allows the simulator to reject a hop advancement and roll back the state to the start of the previous hop. The extra cost here is that the work done for the rejected step would be wasted. With that caveat in mind, we can allow `startHop()` to register the new barrier, and if it is too short, then also register a step validation function that will cause the simulator to reject the ongoing hop state when it is completed. Once the hop is rejected, the simulator will initiate a new hop - but this time to the newly registered earlier barrier, and everything can proceed normally. As noted, this process should work but will incur the additional cost of a wasted propagation step every time this is done, so may not be desirable. However, this approach can be handy for situations where the time delay value is only known at the start of the hop.

If fact, we may need to make this a standard part of our checks that when a hop completes, we can check if the next explicit hop barrier is in the past, and in which we should automatically reject the hop since the barrier must have been added during the hop, and we need to repeat the hop while taking into account this new barrier:

### 1.5.3. Working with FSMs

We often see usage where there is a time stepping loop that has a check for the terminate flag being set to exit the loop. This is often done in the context of state machines managing the simulation, where we want the the simulation to stop when a certain state has been reached. The problem with this approach is that the size of the individual steps are somewhat arbitrary and have no real basis. Furthermore such a loop forces a return to the Python layer after each step which reduces performance. A better way would be to have the simulation start running with only an end callback based on the FSM final state specified. This will get rid of the loops in the scripts, and also allow the execution to all happen at the C++ level.

### 1.5.4. Notes from [issue](https://dartsgitlab-internal.jpl.nasa.gov/darts-lab/jpl-internal/all-access/development/simulation-framework/dshellpp/-/issues/23) (https://dartsgitlab-internal.jpl.nasa.gov/darts-lab/jpl-internal/all-access/development/simulation-framework/dshellpp/-/issues/23) on the new time stepping scheme.

#### 1.5.4.1. Summary of changes to sim stepping scheme

- Now try to advance time to the next barrier, rather than trying to go to the next IO step.
- We keep stepping from barrier to barrier until a stop callback is reached or the `e_stop` is triggered.
- We added a second timed event queue that runs before the step rather than after it.
- Added a priority minor counter to callback events. This allows us to sort events with the same time and priority value by the order they were registered in. This retains old behavior where `begin/endIOStepEvents` with the same priority would execute in the order they were registered in.
- `beginIOStepEvents` are now registered as timed events in the before-step timed queue. This has some consequences.
- Previously, `beginIOStepEvents` ran at every IO step and in the residual between steps. For example, if the IO step was 1.0 seconds but I advanced time by 0.3 seconds, the event would fire at 0.3 seconds when I tried to step the simulation again. This doesn't happen anymore, the IO step events are only called at the IO step barriers. If you want to call the events for some reason (to maintain previous behavior) there is a `sim.executeBeginIOStepEvents` method for this.
- `endIOStepEvents` are not registered as timed events in the timed event queue. They have a +1000 added to their priority to ensure they fire before other timed events (this is to maintain backward compatibility, as the previous time-stepping scheme fired these events before timed events). The same consequences that applied to `beginIOStepEvents` also apply to `endIOStepEvents`: see above for details.
- `stepValidationEvents` used to stop the sim from moving forward in time. Now, if you want that behavior, you need to request it by setting `sim.e_stop = True` in your callback. The default is now to continue stepping the sim forward.
- Added `syncDefaultModelPeriodWithCurrentTime` to sync the `io_step` tracker with the current time and `setNextIntendedGlobalIOStepEnd` to set it to a user-defined time. These can be used to reproduce old behavior if `advanceTimeBy` or `advanceTimeTo` end at non-integer multiples of the default model period.

#### 1.5.4.2. Notes for keeping old tests running

- Step validation events no longer stop the current step unless explicitly told to do so. You can stop the sim from running by adding `sim.e_stop = True` to your callback.
- `Begin` and `EndIOStepEvents` now run at the actual IO steps rather than at the IO steps + the beginning/end of the residual step. You may need to call `sim.executeBeginIOStepEvents` or `sim.executeEndIOStepEvents` at the appropriate time if you are stepping just over/under the IO step size often.
- For example, suppose the IO step size is one second, but you call `sim.advanceTimeBy(0.7)`. In the previous architecture, this would call the `EndIOStepEvents` at 0.7 seconds and the `BeginIOStepEvents` at the same time when you begin to step the sim again. In the new paradigm, we only run these events at the IO boundaries. If you'd like to retain the old behavior, then you will need to add `EndIOStepEvents` and `BeginIOStepEvents` in the appropriate spots.
- Model IO step events also used to run at off-IO boundaries when `advanceTimeTo` or `advanceTimeBy` were called. Sometimes, calling an `updateFlowOuts` manually is required. For example, in `DshellDsends/test/test_AspireThrusterAssembly`.
- ~~-May need to use `setNextIntendedGlobalIOStepEnd` and `syncDefaultModelPeriodWithCurrentTime` to make the `io_step` tracker track with time in the way it used to. This can be used to reproduce old behavior if `advanceTimeBy` or `advanceTimeTo` end at non-integer multiples of the default model period. See `RoverVehiclesTest/test/test-Integrators/test-ATRVJr/test_implicitTheta` for an example.~~~

I've added a `ROLLBACK_TIME_STEP_REFACTOR` macro in `Simulation.cc`. It can be enabled by defining `ROLLBACK_TIME_STEP_REFACTOR` at the top. This will print out warnings that can be useful in helping to update reg tests. Users should also ensure deprecation messages are printed out, as there are some helpful messages there as well.

## 1.6. Sphinx documentation

### 1.6.1. Dshell Simulation

#### 1.6.1.1. Physics Encapsulation Architecture

- The model connection to multi-body dynamics enforces *state views* consistent with the underlying physics.

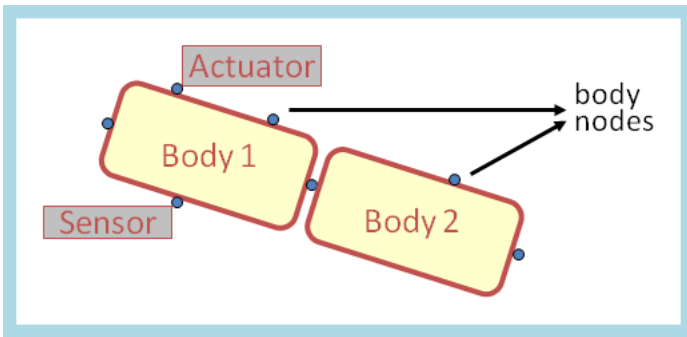


Figure 8. Physics Encapsulation of Multi-Body Dynamics

- Full state is deliberately not made available to models.
- Makes inter-connect definitions simple and avoids errors and global variables. Example: Moving a sensor from one body node to another is transparent to the user - the data view is automatically updated by the underlying architecture.
- Note This physics encapsulation strategy is specific to S/C dynamics modeling and is not appropriate for flight software design.
- The complex non-linear differential equations governing the dynamics are simplified through a combination of:
  - multi-body dynamics encapsulation
  - data-flow modularization (via "signals")

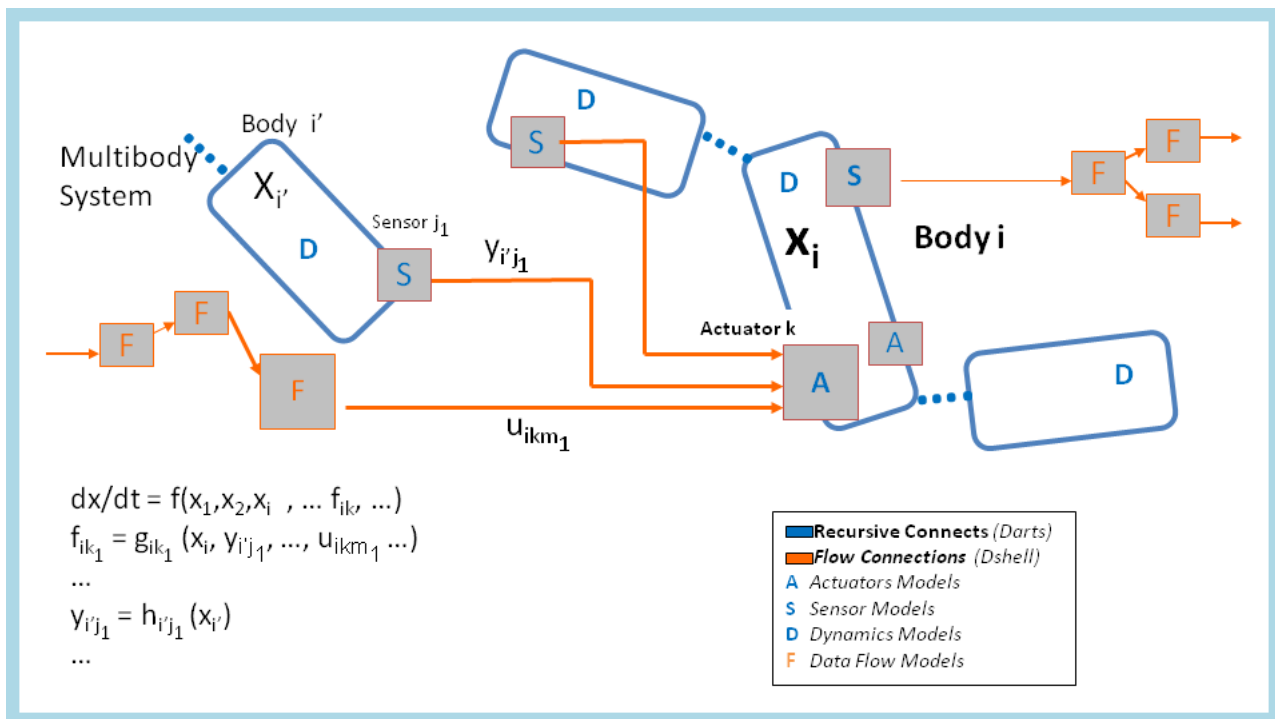


Figure 9. Physics Encapsulation Dataflow

1.6.1.2. Basic simulation loop



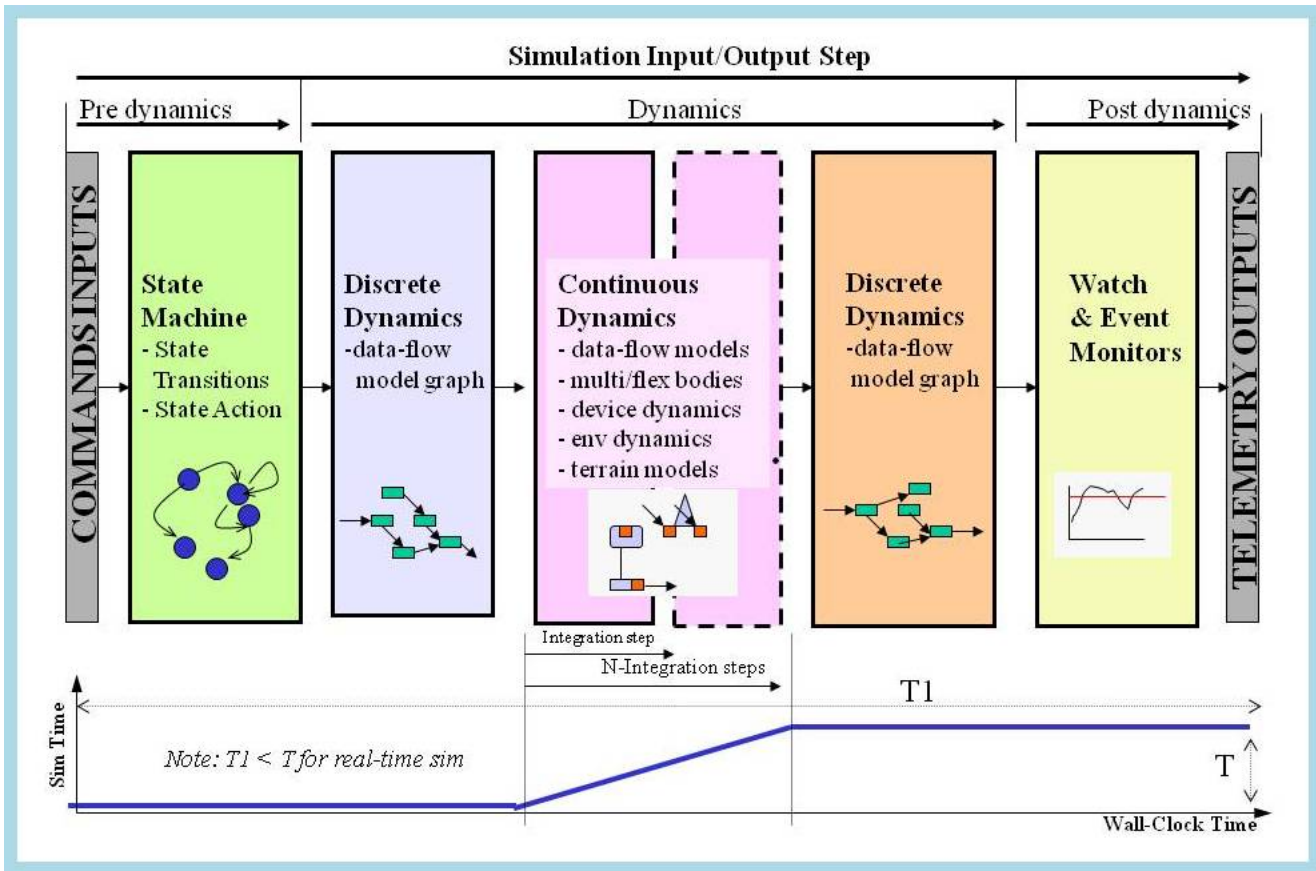


Figure 10. Simulation Time Step

This illustrates the sequence of actions inside Dsends at every step of the simulation. Each step spans the following (in order);

- Inputs at start of input/output step including commands and data
- Pre dynamics State-Machine execution within which the following occur (in order):
  - State action procedures associated with each active state
  - Transition check procedures for each active state
  - State transitions as triggered by the transition checks
  - Execution of all of the side-effects specified for each transition made
- Dynamics Data flow step within which the following occur (in order):
  - Discrete dynamics data-flow preceding continuous dynamics with the execution order managed by Dshell based upon the partial-ordering derived from the data-flow graph.of signal connections
  - One or more integration steps managed and sequenced by the Dshell integrator; with the Darts engine computing the multi-body dynamics related derivatives, the individual models providing additional derivative information as needed, and Dshell managing the execution order based upon the partial ordering constraints derived from the data-flow graph.
  - Discrete dynamics that follow the continuous dynamics with the execution order managed by Dshell based upon the partial ordering derived from the data flow graph of signal connections
- Post io-step Dshell managed watch-handlers that trigger their actions based on:
  - Monitored variables that change their value since he last simulation step
  - periodic user-specified events
- Simulation outputs at end of simulation input/output step

Note that all these are executed in a single thread. The execution time for the entire step is  $T_1$  and the corresponding simulation time interval is  $T$ . Note that  $T_1$  should be less than  $T$  if a real-time simulation is needed.

#### 1.6.1.2.1. Initialization

- Create assemblies, models, signals
- Connecting model flowIns/flowOuts with signals
- Update parameters
- Update initial state

#### 1.6.1.2.2. Simulation control

- Run scripts

- Setup:: These are the basic python code to initialize a sim:

```

# Initialize Darts::
from Dshell import DartsCommonMode DartsCommonMode.init(dartsp=True)
+
# Create Dshell object from Dshell import Dartsp_Py from
Dshell.Dshell_Py import DshellX from Dutils import Dvar_Py DshellObj =
DshellX()
+
# Initialize frames from Math.SOA_Py import Frame
Frame.createRootFrame() rootfr = Frame.getRootFrame()
+
# Set the mbody root frame. from Dshell.DartsBase_Py import
DartsBaseMbody DartsBaseMbody.setRootFrame(rootfr) # could be a
different frame

```

PYTHON

- Assembly/model configuration
  - See the Dshell::Model section
  - See the Dshell::Assembly section
- Select parameters
  - See the Dshell Parameter Classes section
- Manage runtime loop

You can use these commands to advance the simulation

```

# Display number of integration steps per i/o step
DshellObj.integrationSteps()

# Change the number of integration steps to 4
DshellObj.integrationSteps(4)

# Display number of integration steps
DshellObj.integrationSteps()

# Display duration of 1 I/O step
DshellObj.ioStepSize()

# Change I/O step duration to 2.0
DshellObj.ioStepSize(2.0)

# Advance sim by 1 I/O step
DshellObj.step() # advance by 1 tick

# Advance sim by 10 I/O steps
DshellObj.step(10) # advance 10 ticks

# Display current sim time
DshellObj.time()

```

PYTHON

- Events
  - The Dshell::CallbackEvent class keeps track of a list of callback functions (which can be written in C or python) which can be invoked at specified times (e.g. at the start of each I/O step). The user can add new events at the command line, specify how often they should be invoked, and can remove them from the event list.

single: Dshell Events Tutorial single: Tutorial; Dshell Events

### 1.6.1.2.3. Regression Tests

These are the same as in the table above.

test\_Assembly test\_Model test\_Model2 test\_Model7 test\_Model8 test\_Signals test\_Events\_io test\_Events\_model test\_Events\_timed2 test\_Events\_timed test\_Events\_add\_remove\_find test\_Events\_combined test\_Events\_step\_validation



TBD: clean up the above

#### 1.6.1.2.3.1. Dshell Basic Event Operations Example

**Tutorial source:**

##### Basic Dshell++ events regtest

- ▶ [Click to see the Dshell++/test/test\\_Ndarts/test\\_events/test\\_basic\\_event\\_operations/script.py script](#)

#### 1.6.1.2.3.2. Dshell Timed Events Example

**Tutorial source:**

##### Basic Dshell++ timed events regtest

- ▶ [Click to see the Dshell++/test/test\\_Ndarts/test\\_events/test\\_timed\\_events/script.py script](#)

#### 1.6.1.2.3.3. Dshell Combined Events Example

**Tutorial source:**

##### Basic Dshell++ combined events regtest

▶ [Click to see the Dshell++/test/test\\_Ndarts/test\\_events/test\\_combined\\_event\\_timeline/script.py script](#)

#### 1.6.1.2.3.4. Dshell Step Validation Events Example

Tutorial source:

#### Basic Dshell++ step validation events regtest

▶ [Click to see the Dshell++/test/test\\_Ndarts/test\\_events/test\\_step\\_validation/script.py script](#)

#### 1.6.1.2.3.5. Dshell Zero-Crossing Events Example

Tutorial source:

#### Basic Dshell++ zero-crossing events regtest

▶ [Click to see the Dshell++/test/test\\_Ndarts/test\\_events/test\\_zero\\_crossing/script.py script](#)

#### 1.6.1.2.4. Propagating state (integration)

- Use Darts + Dshell models to propagate state

One flow model should call `DshellFlow::advanceDynamics()` in its tick method. This will cause Dshell to do the DARTS computations. If no flow model makes this call, then Dshell will print out a warning and do it for you after all Flow models have had their tick methods called.

#### 1.6.1.2.5. Signal data flows

- See the `Dshell::Signal` section.

#### 1.6.1.2.6. Data Inspection

- See the `DVar` chapter.

#### 1.6.1.2.7. Restarting

##### 1.6.1.2.7.1. Clearing Dshell Tutorial

#### Clearing Dshell simulation tutorial regtest

▶ [Click to see the Dshell++/test/test\\_Ndarts/test\\_clear/script.py script](#)

#### 1.6.1.3. Integration

##### 1.6.1.3.1. Simulation State

Integrating involves propagating the system state which consists of the:

- **Multibody States** assigned to the generalized coordinates  $Q$ , their derivatives  $dQ/dt$  for the multiple (possibly articulated and/or flexible) collections of bodies in the system.
- **Other Model States** representing dynamics within various actuator, sensor, motor and encoder elements in the simulation (e.g. the rate-random walk process dynamics of an IMU model).

These are shown in the figure:

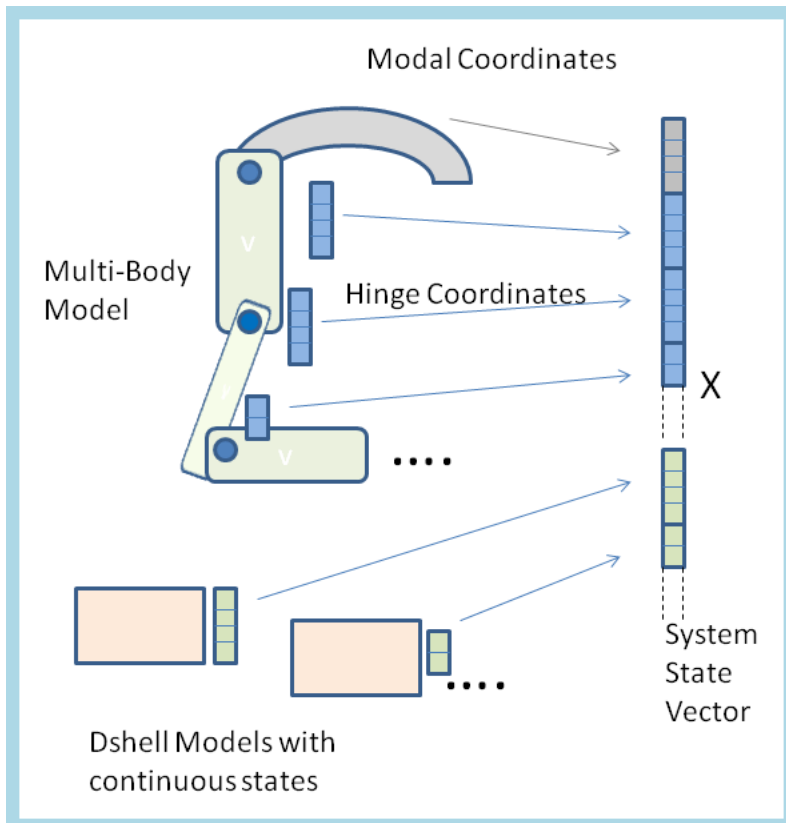


Figure 11. Simulation States

1.6.1.3.2. Multibody Prescribed and Non-Prescribed States

1.6.1.3.3. Darts Multibody Engine

The Darts family of dynamics engines can process both the Forward and Inverse Dynamics of the multi-body system with each hinge in the system being associated with either a Prescribed or Non-Prescribed state.

The corresponding integration schemes for each case involves embedding the Darts dynamics solver within an integration flow.

1.6.1.3.3.1. Integration from Hinge Generalized Force Inputs

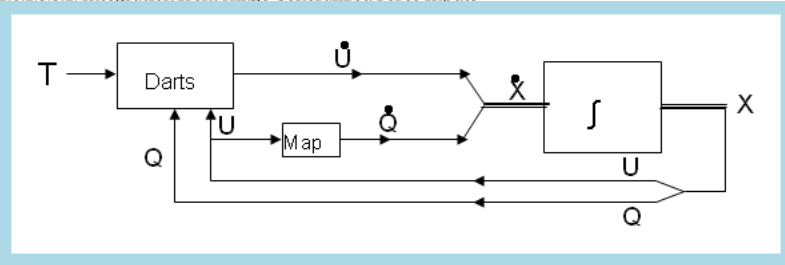


Figure 12. Integrating Dynamics

1.6.1.3.3.2. Integration from Hinge Generalized Acceleration Profile Inputs

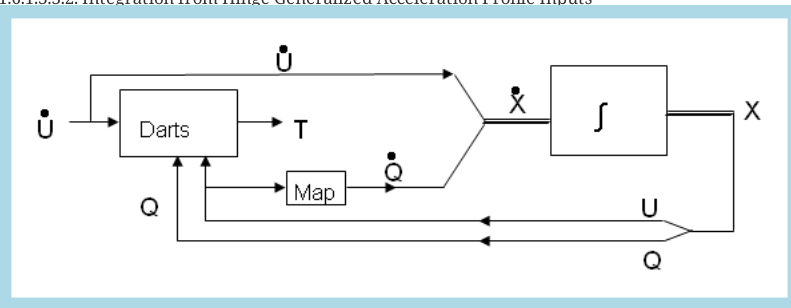


Figure 13. Integrating Prescribed Hinge Accelerations

1.6.1.3.3.3. Integration from Hinge Coordinate and Rate Information

The Darts dynamics engine is also capable of dealing with kinematic prescriptions of the generalized coordinates that are not at the acceleration level.

In this case an a generalized acceleration ( $dU/dt$ ) is provided by the user to allow dynamics computations to be performed, but only the  $dQ/dt$  (determined from the prescribed  $U$ ) is integrated to evolve the coordinate  $Q$ .

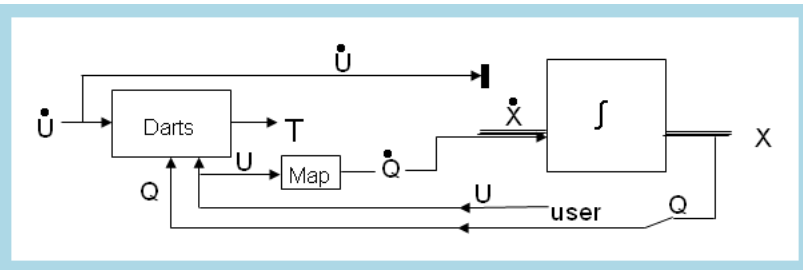


Figure 14. Integrating with Prescribed Hinge Velocities

In this case an a generalized acceleration ( $dU/dt$ ), velocity ( $U$ ), and coordinate ( $Q$ ) are provided by the user to allow dynamics computations to be performed, but no integration is performed.

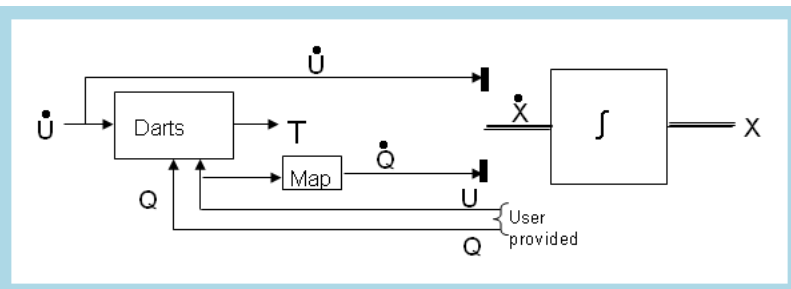


Figure 15. Integrating with Prescribed Hinge Coordinates

The distinction between the various prescribed hinge types is indicated in the simulation by joint-level flags:

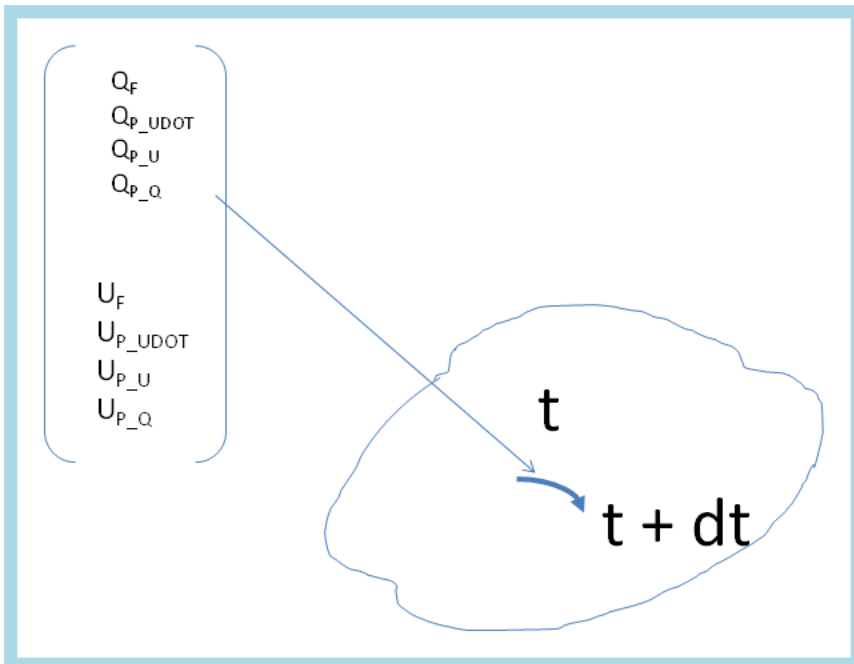
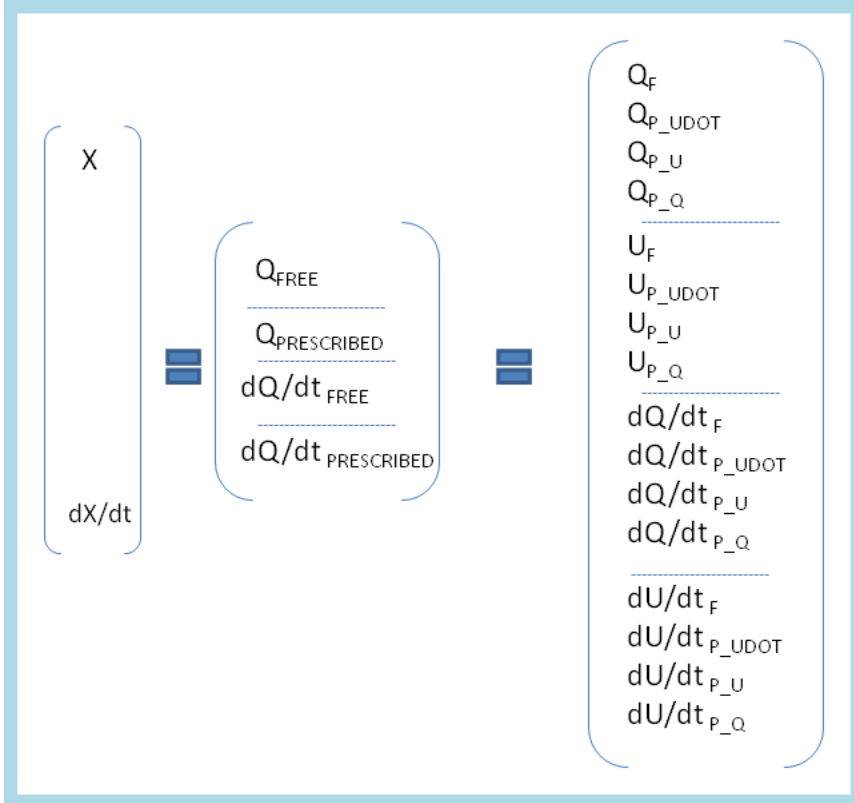
| Prescribed Quantity | Hinge Quantity | Hinge Flag | Hinge Prescribe Type |
|---------------------|----------------|------------|----------------------|
| $dU/dt$             | acceleration   | PRESCRIBED | UDOT                 |

| Prescribed Quantity | Hinge Quantity | Hinge Flag | Hinge Prescribe Type |
|---------------------|----------------|------------|----------------------|
| U                   | velocity       | PRESCRIBED | U                    |
| Q                   | coordinates    | PRESCRIBED | Q                    |



TBD: Fix the above - we only allow Udot prescribed motion. Use masking for Q and U

The Dshell state vector corresponding to the multi-body system can then be considered as having the components:



| State Component | Propagation   |
|-----------------|---|
| $Q_F$           | ODE solver based upon double integration of Darts $dU/dt$ |

| State Component     | Propagation   |
|---------------------|---|
| Q <sub>P_UDOT</sub> | ODE solver based upon double integration of Darts dU/dt |
| Q <sub>P_U</sub>    | ODE solver based upon single integration of Darts U     |
| Q <sub>P_Q</sub>    | Responsibility of user                                  |

#### 1.6.1.3.4. State Masking/Dummification

To propagate the state vector for the general case involving a mixture of Prescribed and non-Prescribed generalized coordinates, Dshell adopts the following strategy:

- Dshell always propagate the entire state by double integrating the dU/dt. However, the Dshell Integrator receives dummy and/or zero values for appropriate X and dX/dt components corresponding to PRESCRIBED; TYPE U & Q
- The propagated values are discarded as appropriate when updating Darts state

##### 1.6.1.3.4.1. Advantages

- Allows block (fast) copy of contiguous states between Darts and Models into integrator buffer

##### 1.6.1.3.4.2. Disadvantages:

- Ad-hoc scheme that is customized for the PRESCRIBED TYPE U and Q
- Fairly opaque code with various calls to marshalling related pre/post pack/unpack calls.
- Does not generalize to case state components need to be propagated using other propagators e.g. multi-rate integration, analytic propagation, high accuracy quaternion quadrature
- May complicate/confuse integrator's monitoring of relative and absolute tolerances as integration proceeds
- May be inefficient if many states corresponding to prescribed state are unnecessarily propagated and then discarded

#### 1.6.1.4. Derivative Evaluation

##### 1.6.1.4.1. Steps

We examine first the sequence of steps involved in computing the derivatives required by the integrator:

- **Unmarshalling.** The process by which the state vector is assigned to the Darts model multibody system hinge coordinates.
- **Mask.** The process by which certain prescribed quantities override portions of the Darts model state.
- **Dynamics.** The exercise of the Darts forward/inverse dynamics computation.
- **Dummify.** The process by which certain elements of the model state are zeroed out.
- **Marshalling.** The process by which the Darts model multibody state derivative is copied out for use by the integrator.

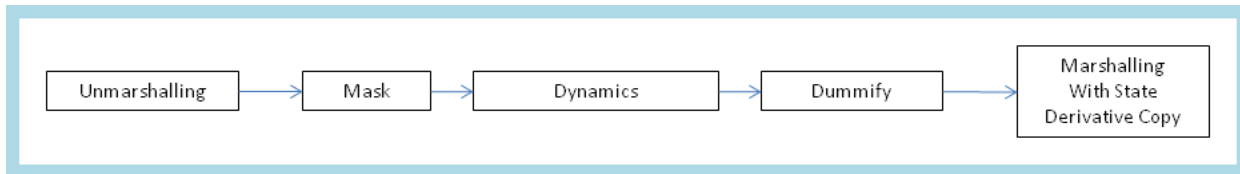


Figure 16. Steps for Derivative Evaluation

##### 1.6.1.4.2. Unmarshalling

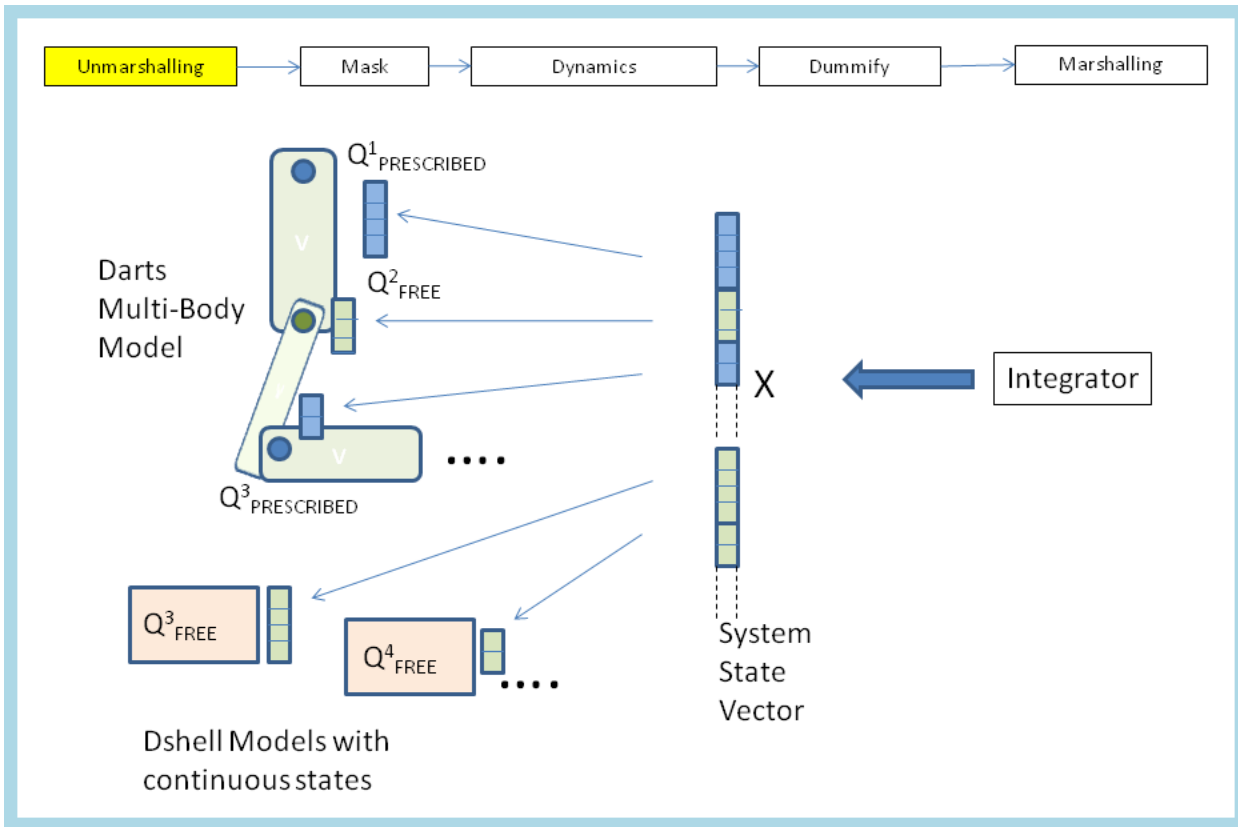


Figure 17. Simulation State Unmarshalling for Derivative Evaluation by Integrator

1.6.1.4.3. Mask

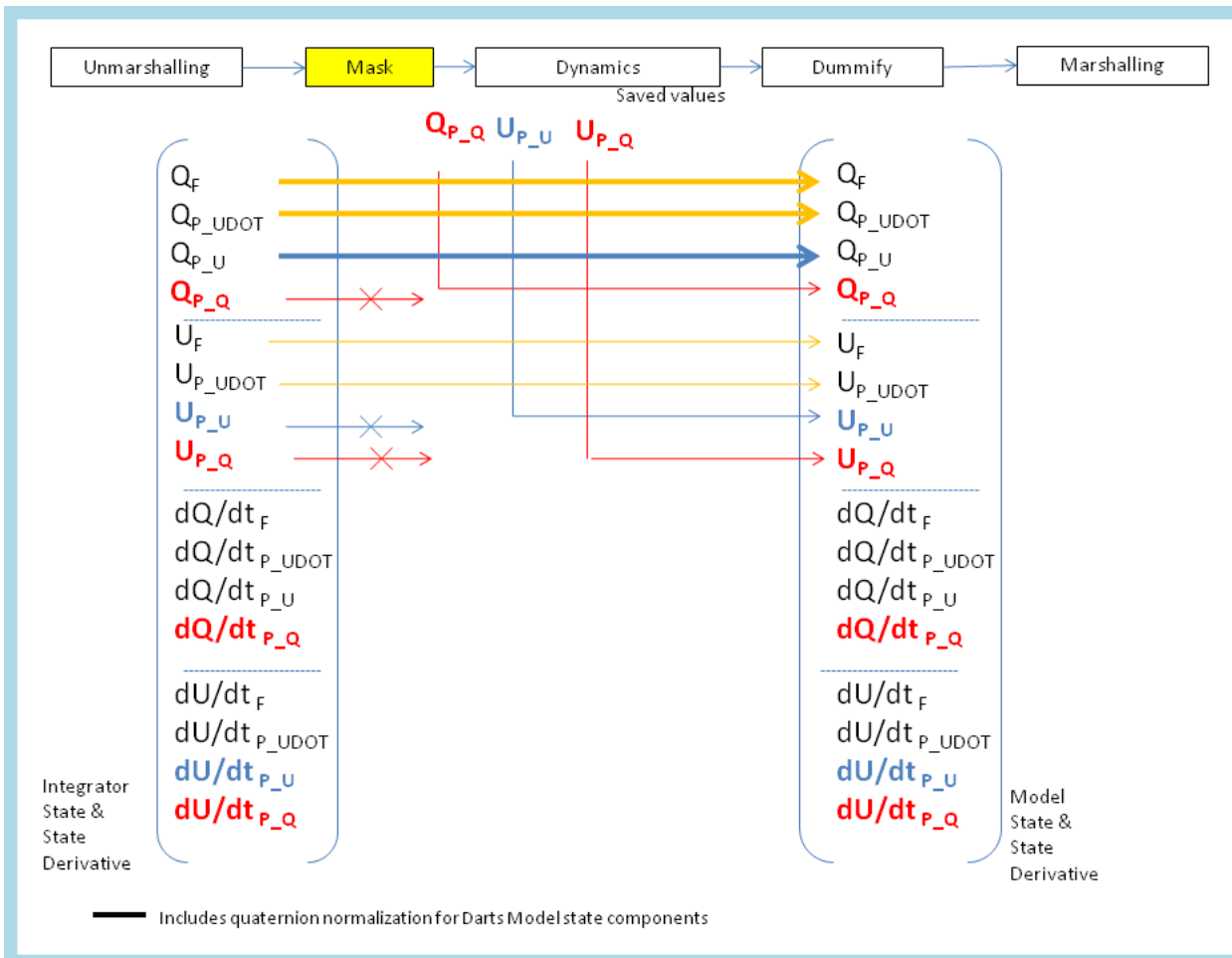


Figure 18. State Mask for Derivative Evaluation by Integrator

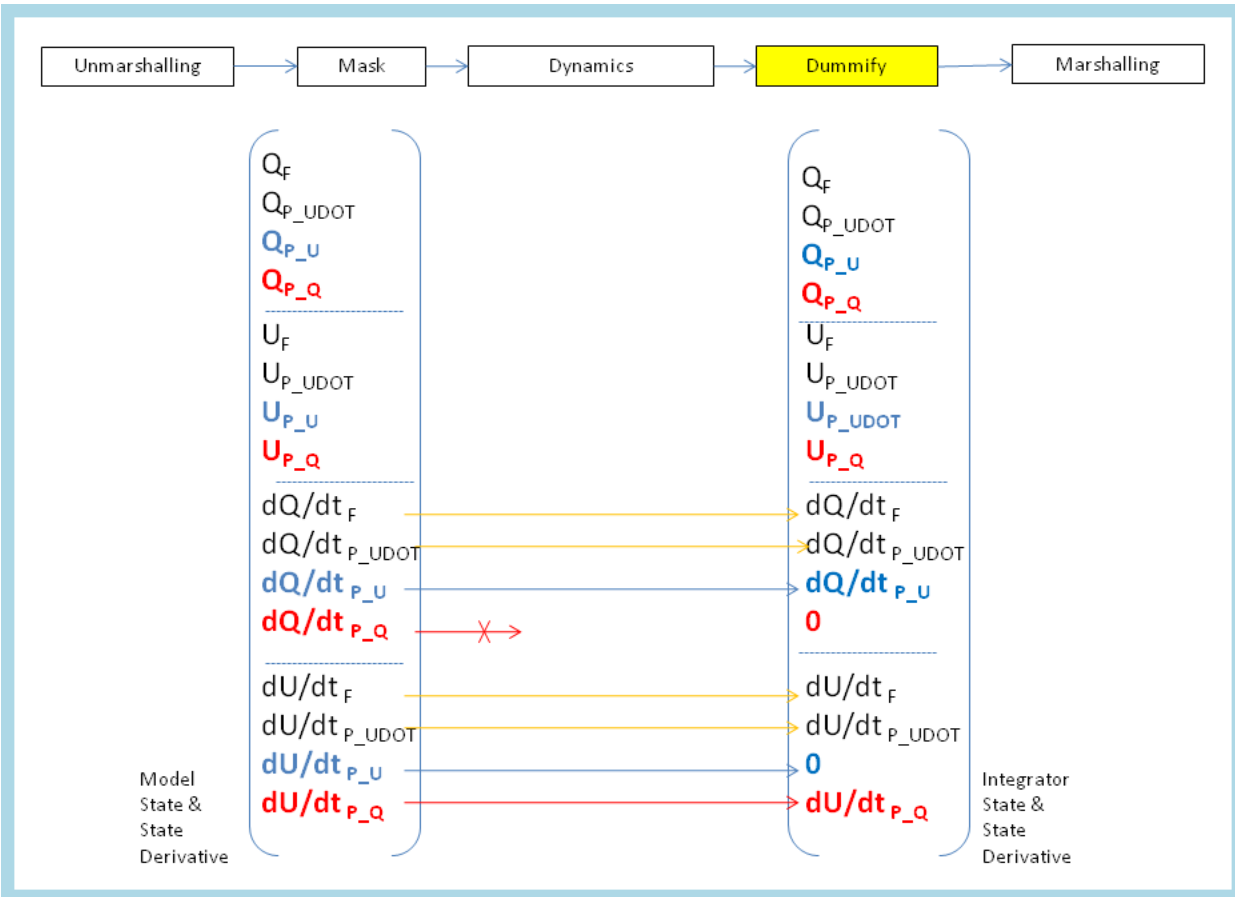


Figure 19. State Dummify for Derivative Evaluation by Integrator

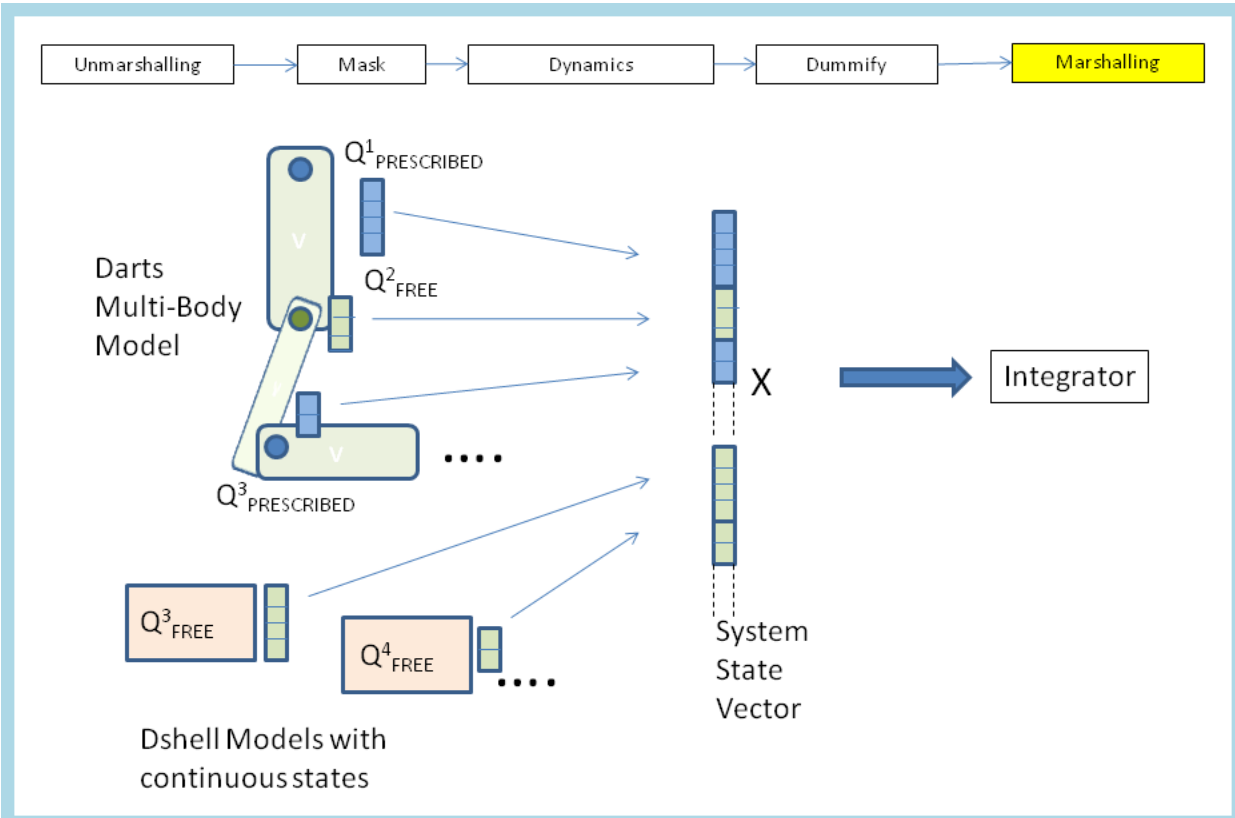


Figure 20. Simulation State Marshalling for Derivative Evaluation by Integrator



We next consider the sequence of steps involved in propagating the state across a time-step.

### 1.6.1.5.1.Steps

We examine first the sequence of steps involved in computing the derivatives required by the integrator:

- **Dummify.**
- **Marshalling.**
- **Integration.**
- **Mask.**
- **Unmarshalling.**

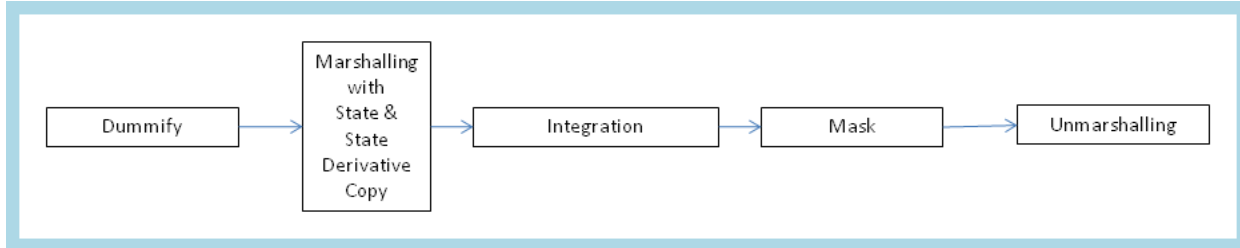


Figure 21. Flow for Integration Steps

### 1.6.1.5.2. Dummify

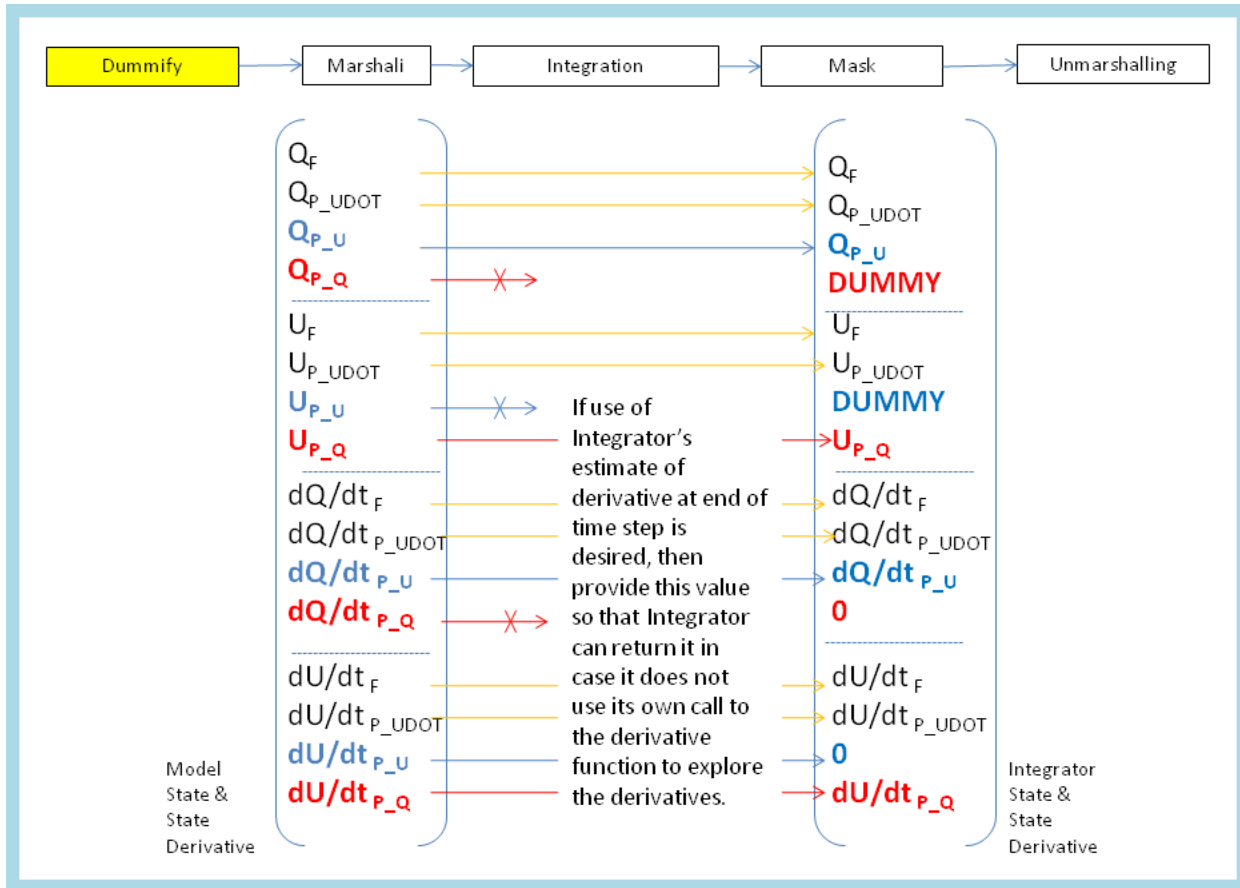


Figure 22. State Dummify for Integration Step

### 1.6.1.5.3. Marshalling

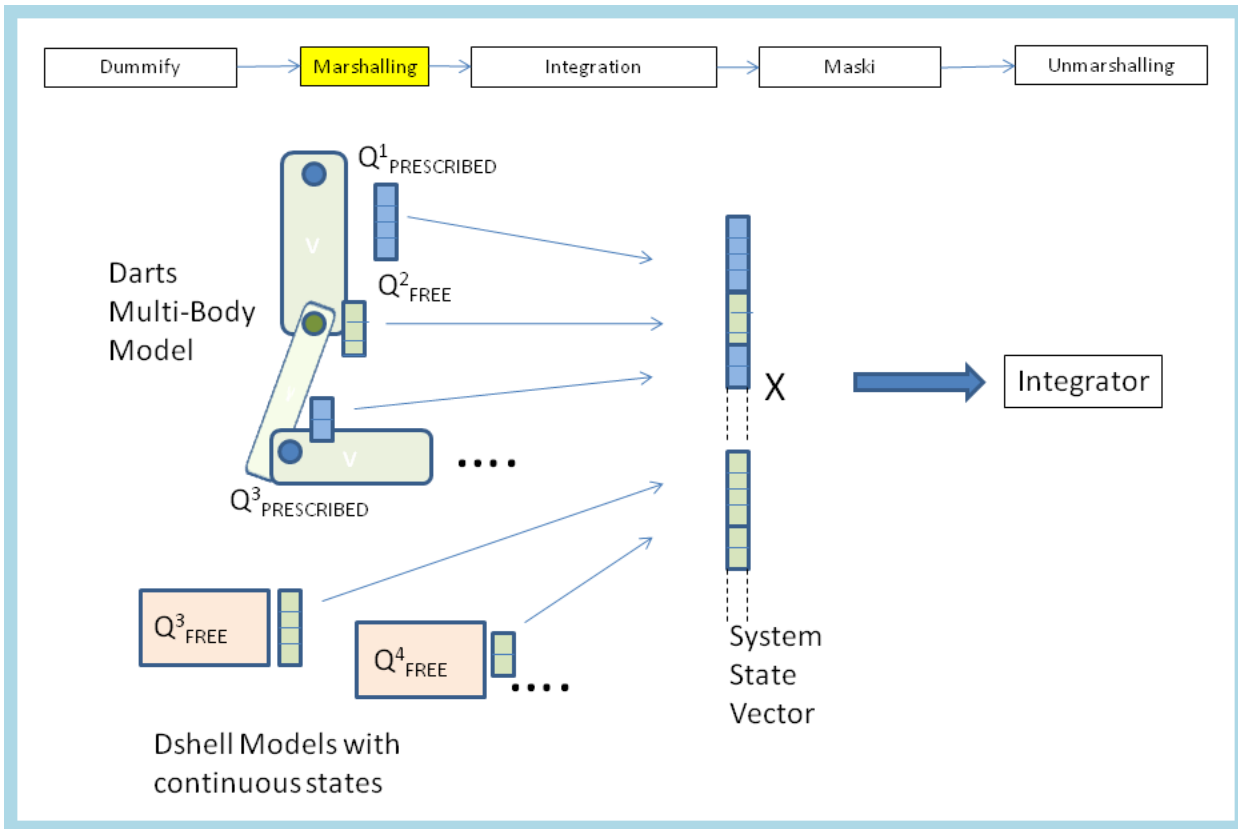


Figure 23. Simulation State Marshalling for Integration Step

1.6.1.5.4. Mask

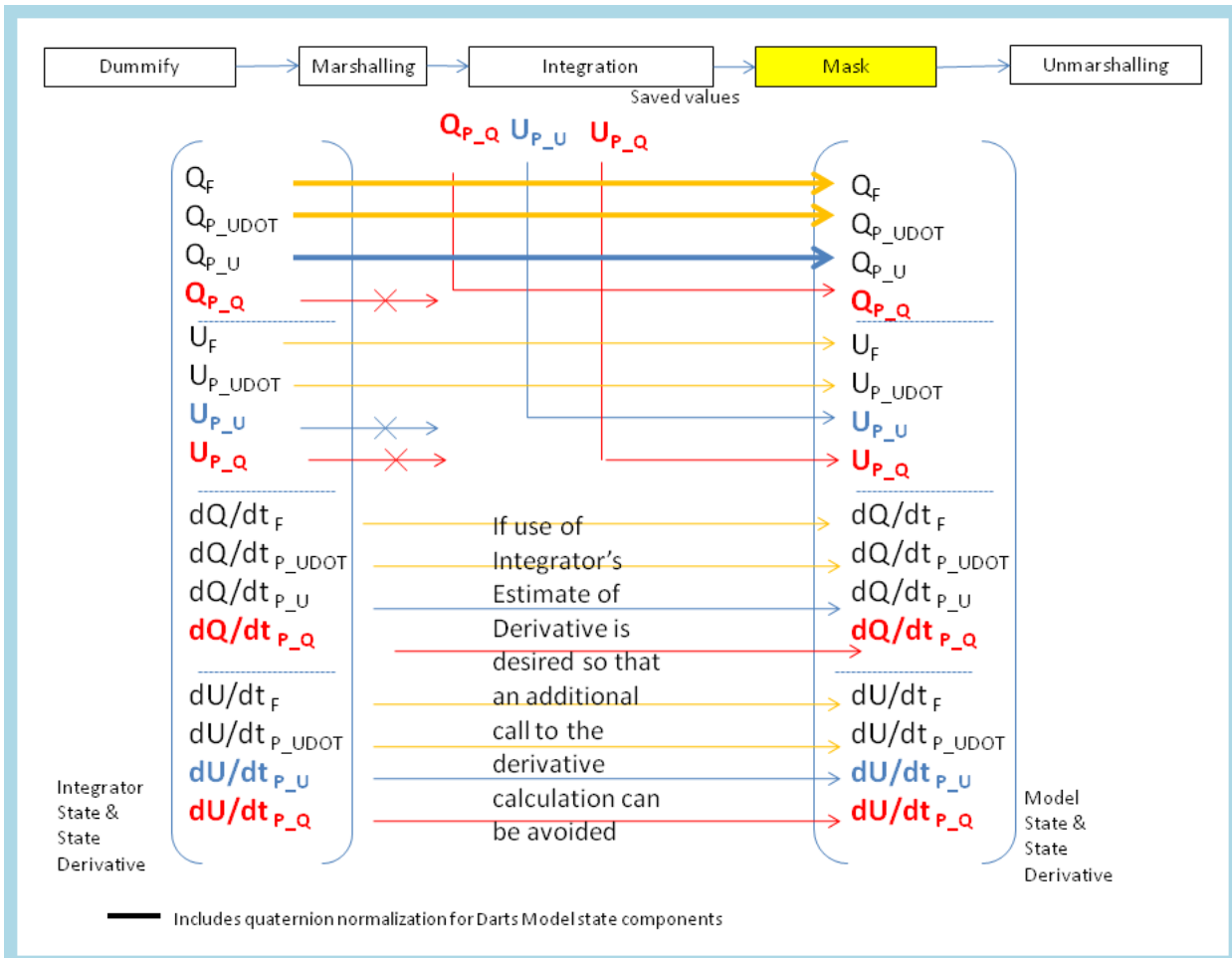


Figure 24. State Mask for Integration Step

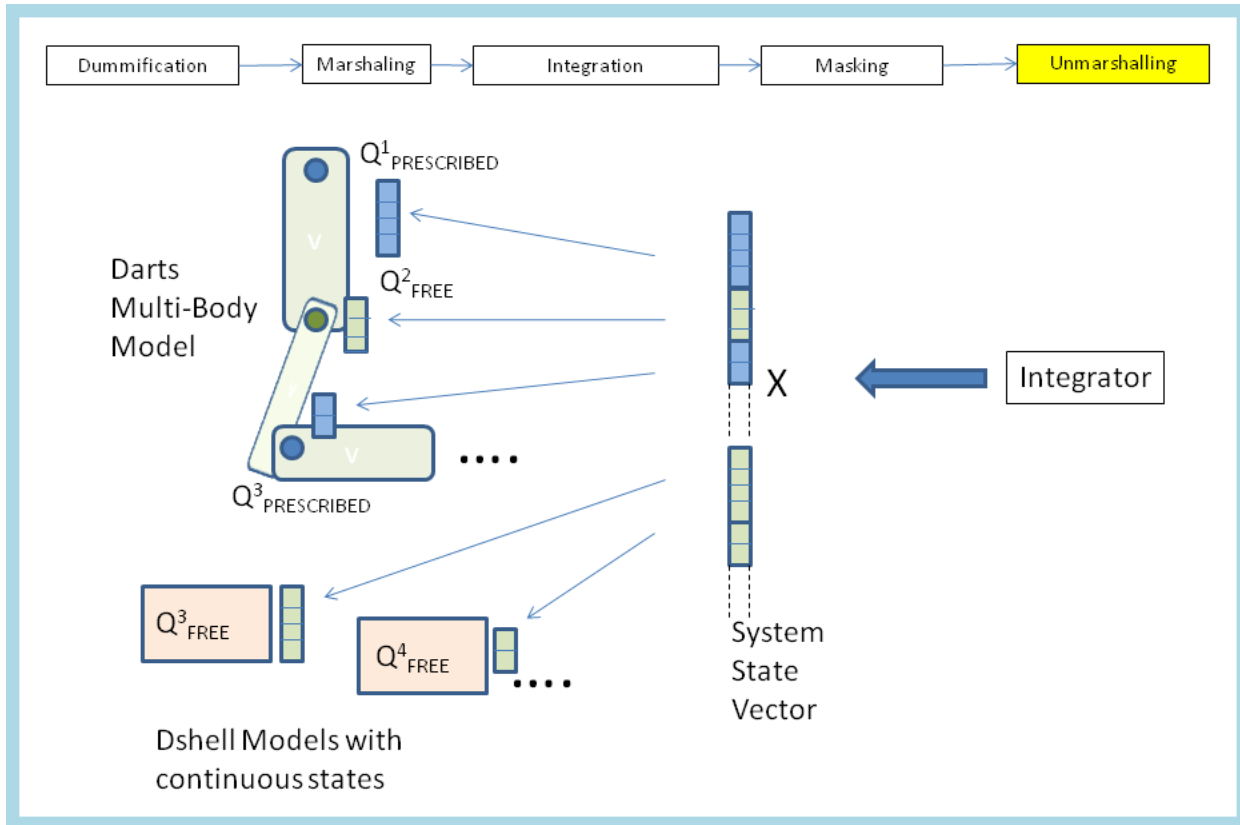


Figure 25. Simulation State Unmarshalling for Integration Step

#### 1.6.1.6. Implementation Notes

##### Dummify is implemented using

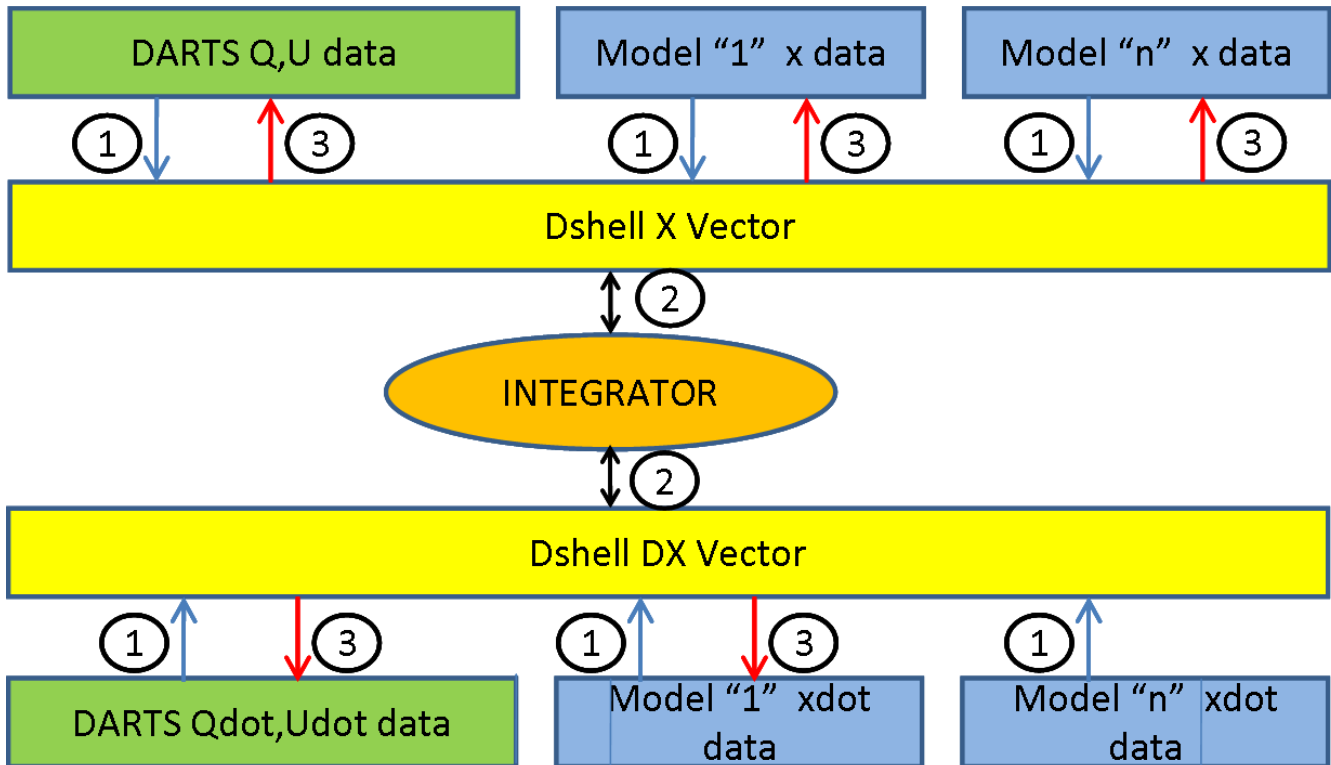
- mbody→prePack
- mbody→postPack

##### Masking implemented using

- mbody→prescribed
- Mbody→postUnpack

##### 1.6.1.6.1. Automatic construction of the big "X" vector

- Dshell creates a double array called "X Vector" large enough to hold all the Darts Q,U states and *active* models' x states (see figure below).
- Dshell creates a second double array called "XDOT Vector" large enough to hold all the Darts Qdot, Udot states and active models' xdot data.
- Changes to the number of states are handled as follows:
  - User must call DshellX::unlock() before making a topology change (adding/deleting/moving bodies or models).
    - The unlock() functions calls DartsMbody::unlock() and simply sets a "lock flag" to false.
    - Methods that change topology (e.g. adding a body) checks to make sure the lock flag is false otherwise an exception is raised.
    - Activating/Deactivating models do not require unlocking.
    - Prescribing/Unprescribing states do not require unlocking.
  - User must call DshellX::lock() when the changes are complete.
  - The DshellX::unlock() function does the following:
    - Calls DartsMbody::lock() which creates the big Darts Q,Qdot,U,Udot buffers.
    - Creates the Dshell X,XDOT vectors.
  - lock()/unlock() should only be called at I/O step boundaries.



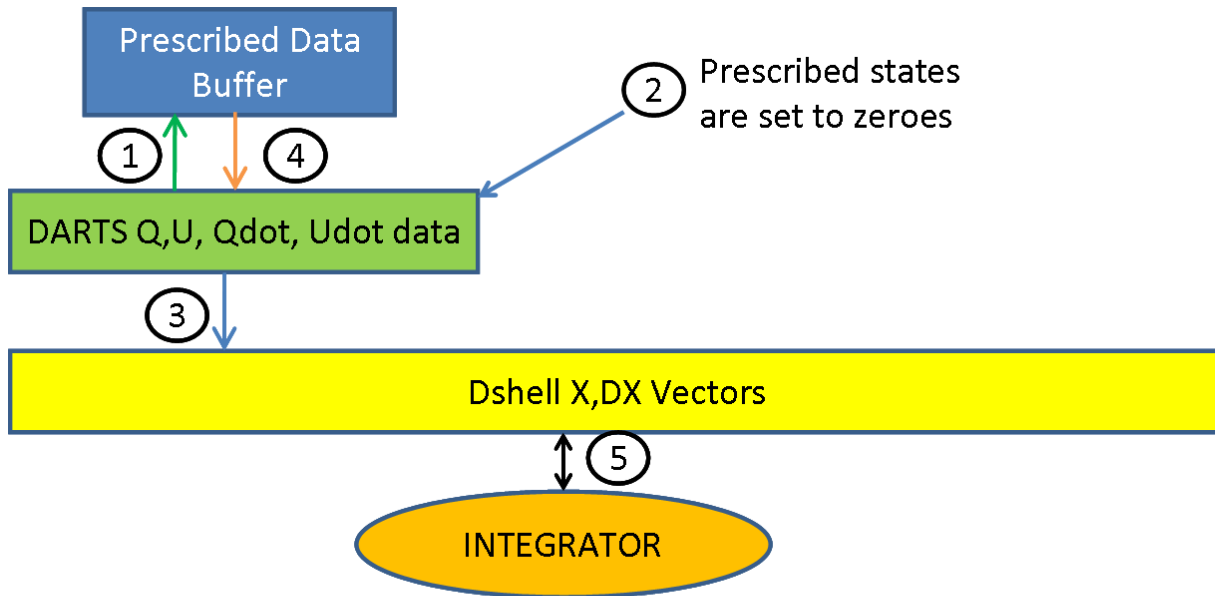
### Summary of State Propagation:

1. Darts and model state data are copied to the Dshell X and DX vectors
  - In Darts, all hinge Q data are stored in a single big Q double array (same with Qdot, U, Udot)
  - Model x,xdot data are kept in local DshellModel objects.
2. The X and DX vectors are passed to the integrator for processing.
3. The X and DX vectors are copied back to Darts and models (model xdot not updated).

- Prescribed States

Prescribed states are handled in Darts not Dshell. The prescribed states are preserved by temporarily copying the prescribed states to a private buffer in Darts then restoring the values (see figures below).

- How prescribed states are preserved when copying from Darts to Dshell

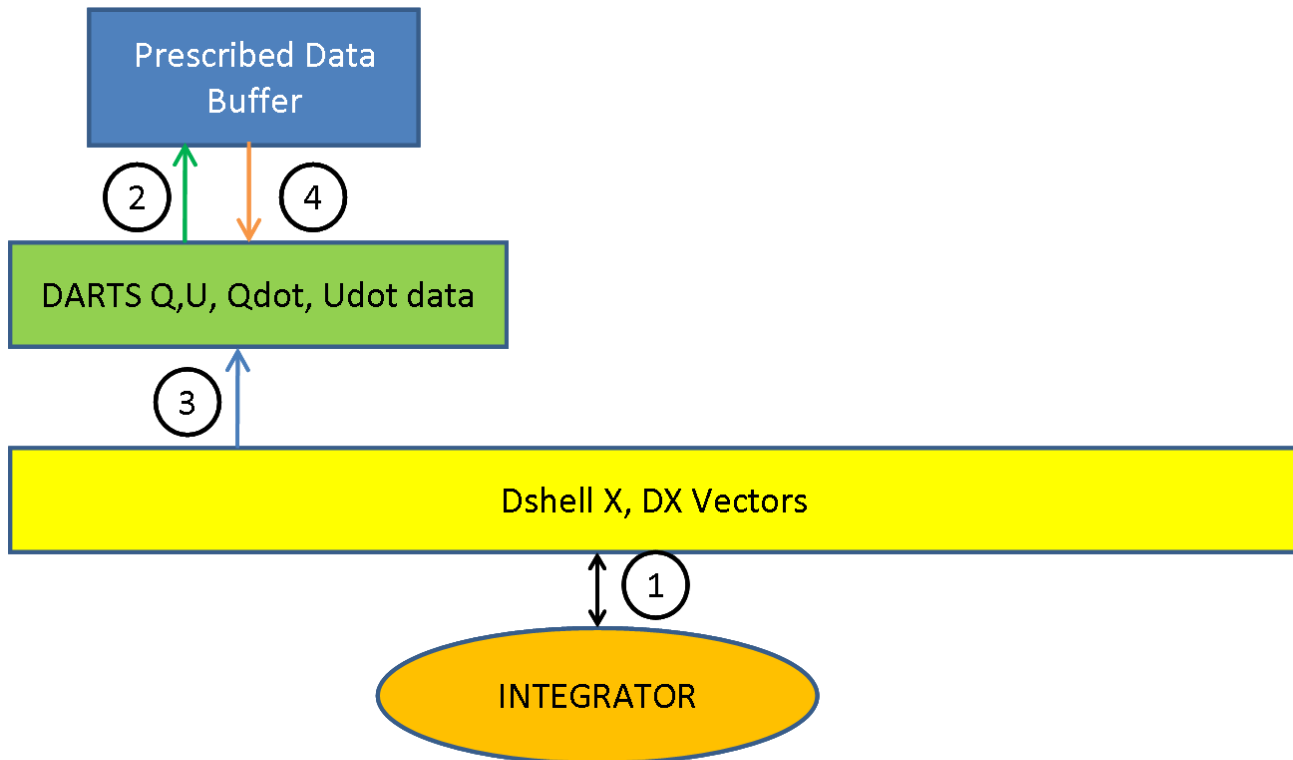


② Prescribed states are set to zeroes

**How Prescribed States Are Preserved when Copying from DARTS to Dshell:**

- 1. and 2.: Dshell calls `DartsMbody::preUnpack()`:
  - 1. Darts copies prescribed state values to a private Prescribed Data buffer.
  - 2. Prescribed states are zeroed in the Darts vectors.
- 3. Darts data values are copied to the Dshell X and DX vector using `memcpy()`
- 4. Dshell calls `DartsMbody::postUnpack()`: The saved prescribed values are copied back to Darts Q,U,Qdot,Udot buffers.
- 5. Integrator process Dshell X, DX data

o How prescribed states are preserved when copying from Dshell to Darts



**How Prescribed States Are Preserved when Copying from Dshell to DARTS:**

- 1. Integrator processes Dshell X, DX vectors
- 2. Dshell calls `DartsMbody::prePack()`: Darts copies prescribed state values to a private Prescribed Data buffer.
- 3. Darts data values are copied to the Dshell X and DX vector using `memcpy()`
- 4. Dshell calls `DartsMbody::postPack()`: The saved prescribed values are copied back to the Darts Q,UQdot, Udot buffers.

## 1.6.2. Dshell Models

### 1.6.2.1. Introduction

#### 1.6.2.1.1. Dshell Model Basics

##### 1.6.2.1.1.1. What are Dshell models?

Dshell simulations consist of a collection of component device models from model libraries assembled and connected together into a data flow to meet the required simulation behavior.

For example, a **thruster model**

- Models a device that applies a thruster force on a body
- is a C++ **class** derived from (and inherits all the properties of) an **actuator C++ base class**

Dshell models:

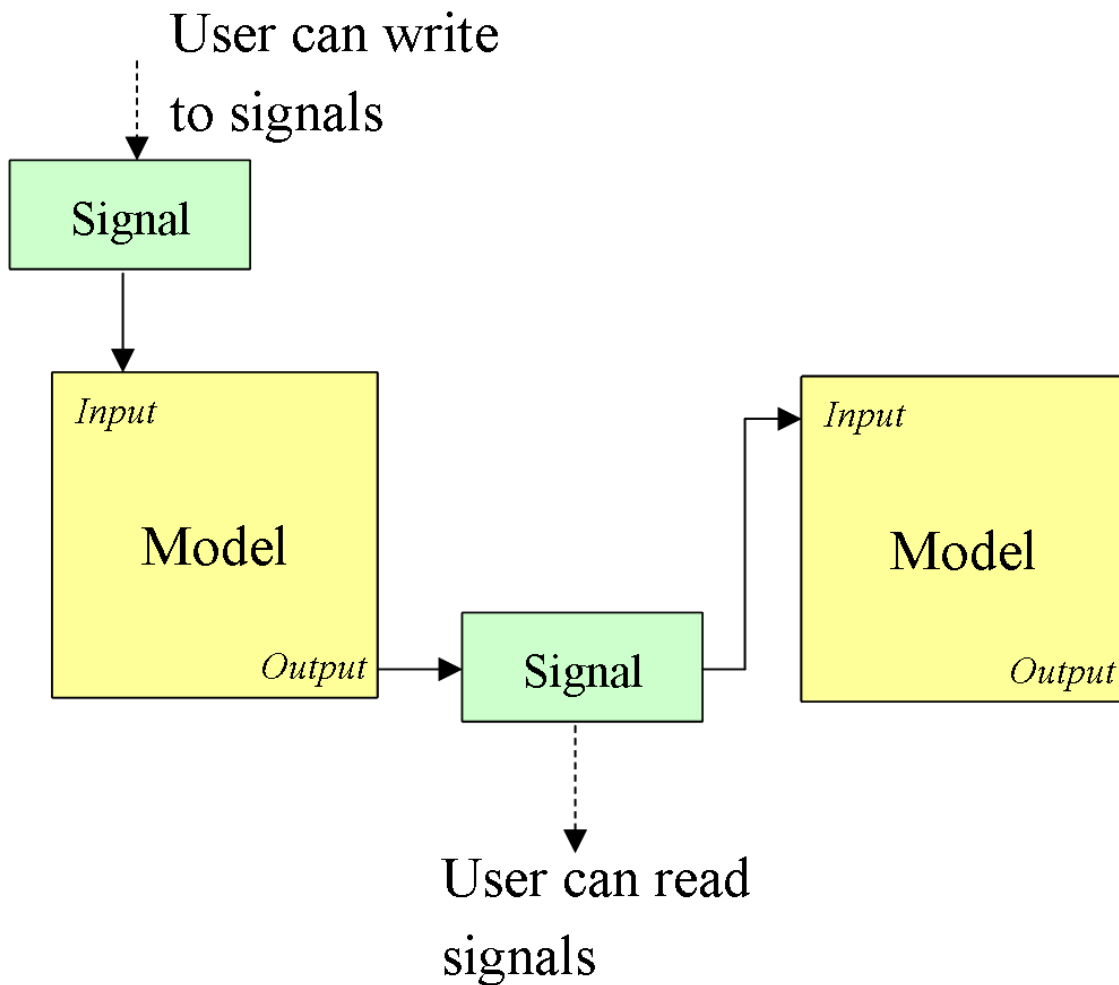
- **object-oriented design**
- Classes are organized into **hierarchies** involving related models, e.g. families of gravity models, thruster actuator models.

Each Dshell++ model:

- Has a standard interface for **parameter** and **state** data
- Has a standard interface for flow inputs and outputs which can be customized for each specific model.

Each Dshell model:

- Has user-defined **input and output ports**
- Data is shared between models by **signals**:
  - **Signals** tie a model's input port to another model's output port through a signal (basically a shared memory buffer).
  - Users can peek or poke the signal data through the Dvar interface.

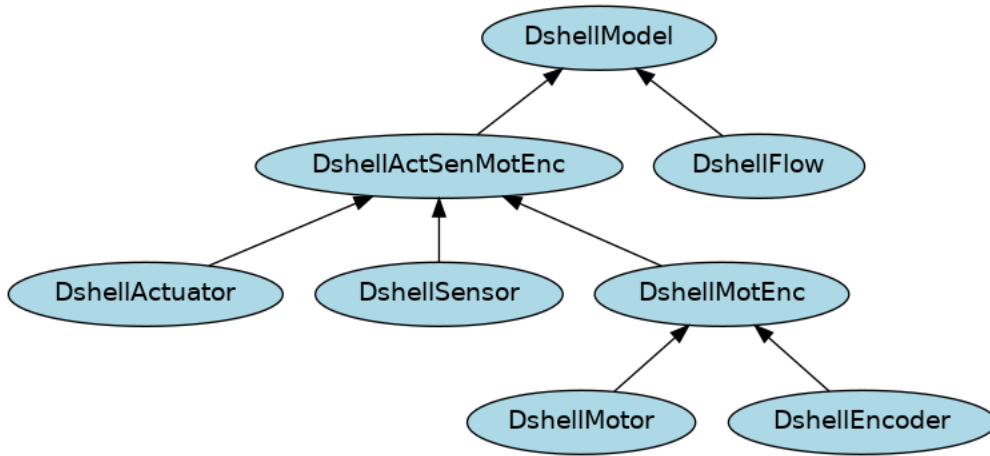


##### 1.6.2.1.1.2. What kinds of models are possible?

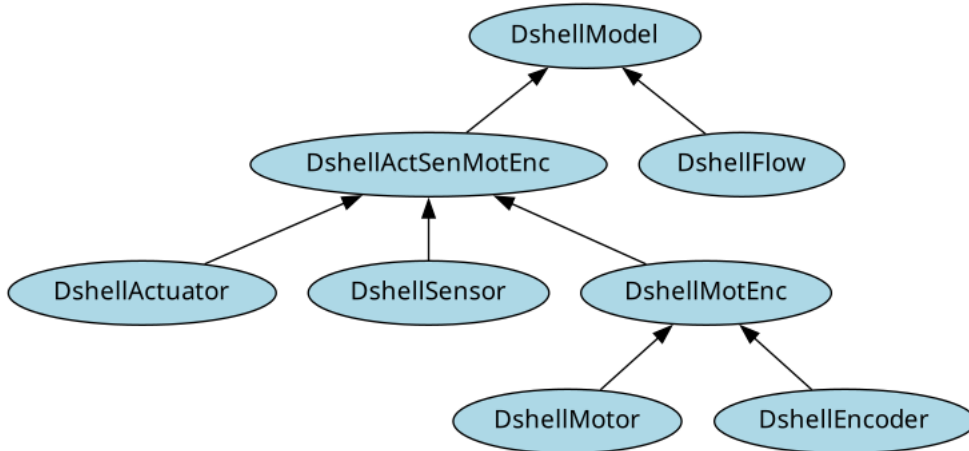
- **actuator** (`ActuatorModel` models apply forces to bodies)
- **motor** (`MotorModel` models apply forces to joints)

- **sensor** (SensorModel models read values from bodies)
- **encoder** (EncoderModel models read values from joints)
- **flow** (FlowModel models are for anything else)

How are these model classes related?



TBD: The above figure should have been autogenerated by graphviz (not working):



#### 1.6.2.1.1.3. Model Data Variables

- **Parameters**

Parameters are values that are set when the simulation starts.

Parameters may be changed at run-time by the user. Parameters can be checkpointed. A model's parameter variables are grouped into single structure (use the model's params() method to access a pointer to this structure).

- **Continuous States**

Continuous states are updated by the numerical integrator in DARTS and require the model developer to fill in a method for computing derivatives of these states. Continuous states are implemented as a C++ class object. Use the model's conststates() method to access a pointer to this object. Flow models do not support continuous states.

- **Discrete states**

Any "state" of the model that is not a continuous state (ie, does not require derivative evaluation for propagation) is a "discrete state".

Discrete states are initialized at start up and may be modified by both the model and the user during run-time.

A model state variables are grouped together into a single structure and can be used to share data among methods of the same I/O step within the same model. Use the model's states() method to access a pointer to the States structure. The difference between scratch and state variables is that state variables can be checkpointed.

- **Scratch**

A model's scratch variables are grouped together into a single structure and can be used to share data among methods of the same I/O step within the same model. Use the model's scratch() method to access a pointer to the States structure. The difference between scratch and state variables is that scratch variables are not checkpointed.

- **FlowIns**

These are inputs to the model.

A model's input variables are implemented as "pointers" so need to "tied" to a DshellSignal object. Use the model's flowIns() method to access a pointer to the structure of inputs.

- **FlowOuts**

These are outputs of the model.

A model's output variables are implemented as "pointers" so need to "tied" to a DshellSignal object. The flow outs are normally computed in the model's updateFlowOuts() method. Use the model's flowOuts() method to access a pointer to the structure of outputs.

#### 1.6.2.1.1.4. Model Methods

The following methods are called by Dshell during initialization to create the model:

- **constructor**

is called to instantiate (create) the model. Use it to set default values of states and parameters.

- **setup()**

is called once after all models have been instantiated and parameter values have been set. Use this method to compute derived parameters whose values depend on other parameters.

- **processParams()**

is called whenever parameters are updated. This function can be used to perform other initializations that are dependent on parameter values.

- **updateFlowOuts()**

is used to compute the flowOuts. updateFlowOuts() is called by Dshell depending on the value passed to the model's updateFlowOutsGranularity(flag) method. Valid flag values are:

- GRANULARITY\_PREDERIV : call updateFlowOuts() after every preDeriv() call
- GRANULARITY\_POSTDERIV : call updateFlowOuts() after every postDeriv() call
- GRANULARITY\_ENDINTEGRATION : call updateFlowOuts() after endIntegration step
- GRANULARITY\_ENDIO : call updateFlowOuts() at end of I/O step

The flags can be combined (e.g. GRANULARITY\_PREDERIV | GRANULARITY\_POSTDERIV )

For actuators, sensors, motors and encoders, the following methods are called by Dshell at each tick (I/O step) to advance the model's state:

- **startIoStep()**

is called at the beginning of every I/O step.

- **startIntegrationStep()**

is called at the beginning of every integration step. This is often where commands to actuator and motor models are handled.

- **preDeriv()**

is called by the DARTS numerical integrator immediately *before* the time derivatives of DARTS state vectors are computed. This method can be used to:

- Apply forces/torques within actuator models on the nodes they are attached to
- Apply force/torques within motor models to the hinges they are attached to.

The number of times this method is called per integration step depends on the numerical integrator used (four times for rk4, twice for euler).

- **postDeriv()**

is called by the DARTS numerical integrator immediately *after* time derivatives of DARTS state vectors are computed. This method can be used to:

- Access and use quantities that depend on derivatives of the dynamics state e.g. acceleration values
- Compute and use time derivative of continuous states for the models that have them. For example, an IMU model could use the computed node accelerations which have been computed by the dynamics engine to compute derived quantities such as the delta-V increments.

As with preDeriv(), the number of times this method is called per integration step depends on the numerical integrator used.

- **endIntegrationStep()**

is called at the end of every integration step.

- **endIoStep()**

is called at the end of every I/O step and often where sensor outputs are produced.

- **updateFlowOuts()**

is used to compute the flowOuts. updateFlowOuts() is called by Dshell depending on the value passed to the model's updateFlowOutsGranularity(flag) method. Valid flag values are:

- GRANULARITY\_PREDERIV : call updateFlowOuts() after every preDeriv() call
- GRANULARITY\_POSTDERIV : call updateFlowOuts() after every postDeriv() call
- GRANULARITY\_ENDINTEGRATION : call updateFlowOuts() after endIntegration step
- GRANULARITY\_ENDIO : call updateFlowOuts() at end of i/o step

The flags can be combined (e.g. GRANULARITY\_PREDERIV | GRANULARITY\_POSTDERIV )

- Flow models have a **tick()** method which is called at every I/O step().

A model can do anything it wants in these methods, but typically:

- Forces and torques from actuators and motors are applied in preDeriv(), because they affect the DARTS state.
- Derivatives of continuous states for a model are computed in postDeriv(), because they may depend on the DARTS state.
- The number of times preDeriv/postDeriv are called per integration step depends on the integration algorithm used (four times for 4th order Runge-Kutta and twice for Euler).



For models involved in continuous time dynamics, the figure illustrates the calling order of the various methods and the Darts dynamics engines. The figure shows an upstream model in the model data-flow, a downstream model, and the dynamics engine. The various methods in each model are annotated with an index where the ordering of the indices illustrates the calling order e.g. 1.3.2 is called before 1.4, etc.

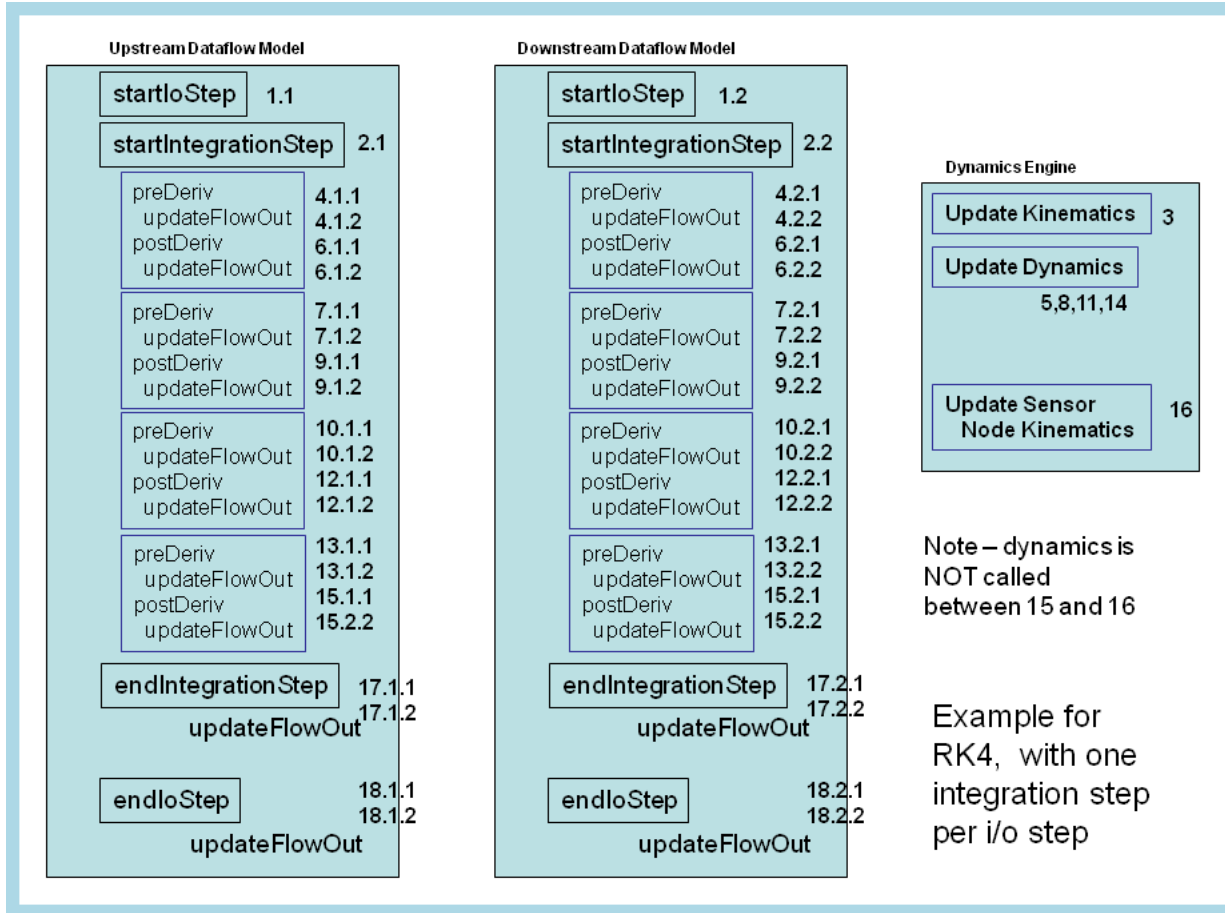


Figure 26. Dshell Model Calling Order

#### 1.6.2.1.2. Dshell model functionality

- Can model anything, including devices and processes
- May be connected to Darts backbone and knows how to interact with integrator

The system's DARTS multibody dynamics module is a key backbone layer supporting all the Dshell models. Dshell models can interact with the DARTS multibody dynamics library which computes the kinematics and dynamics state of the space vehicle. The Dshell models have explicit interfaces to the relevant nodes and hinges in the DARTS model that allow them to make direct function calls to get/set the needed data efficiently. The DARTS module is responsible for propagating the dynamics state of the system (e.g. spacecraft attitude and velocities, momentum, mobility slippage, vibration modes) using numerical integrators. The multibody dynamics states can be highly non-linear and coupled, and even though the physical assets (e.g. vehicles) may be distinct, their interactions must be handled together for the proper solution to the system's dynamics. As a consequence, the dynamics module is implemented as a unified model that includes contributions from all multibody components in the system. However, we have been careful to make sure this coupling does not adversely impact the overall modularity of the system. Towards this, Dshell models have been designed to have a restricted interface to the multibody model to avoid unnecessary interactions and coupling among the Dshell models.

- Continuous or discrete states

Model states can include both discrete and continuous states. A model interface file declares these externally visible characteristics for each model. Auto-code generators based on the Cheetah templating tool were written to read these interface files and generate C++ model code.

#### 1.6.2.1.3. Dshell Model C++ API Overview

- Basic Classes:

- **Model**

The DshellModel class is the base class for dynamic (actuator, sensor or motor) models and flow models. For example, a thruster device should be derived from the `ActuatorModel` class. To simplify the process, a user can create an ".mdl" file which is a template describing the model's states, inputs and outputs. Stub C++ code can then be autogenerated from the .mdl file.

- **ActuatorModel**

The DshellActuator class represents an actuator (e.g. thruster).

- **SensorModel**

The DshellSensor class represents a sensor (e.g. IMU) model.

- **MotorModel**

The DshellMotor class represent a motor which is attached to a joint/hinge between bodies and is used to articulate bodies.

- `EncoderModel`

The `DshellEncoder` class is attached to a joint/hinge between bodies and is used to determine coordinates for the hinge it is attached to for flight software.

- `FlowModel`

The `DshellFlow` class represents a "flow" model.

Flow models do not have ties to DARTS, and are used to model valves, device electronics, etc. using a data-flow paradigm. They do not have continuous states. Flow models have only the following method:

- **tick()** - called once per I/O step



*Note*

One flow model should call `advanceDynamics()` in its tick method. This will cause Dshell to do the DARTS computations. If no flow model makes this call, then Dshell will print out a warning and do it for you after all Flow models have had their tick method called.

#### 1.6.2.1.4. Sorting Models / Model Order

- Call `DshellObj.modelOrder()` in python to get the current model order.
- Use `DshellObj.sortModels()` to return a list of models ordered according to how the inputs and outputs are tied together.
- It is important that the models be sorted so that models which provide inputs for other models are run before the other models (so that the inputs to the other models correspond to the current time).



*Note*

To see a presentation on Modeling and Model Order (sorting), please see:

- [Modeling and Model Order \(April 2016 TIM\)](#)

#### 1.6.2.1.5. Basic Dshell Model Tutorial

##### 1.6.2.1.5.1. Example of Dshell Model Basic Commands

**Tutorial source:** `Dshell++/test/test_basic/script.py`

#### Dshell Model basic commands regtest

- ▶ [Click to see the Dshell++/test/test\\_Ndarts/test\\_models/script.py script](#)

#### 1.6.2.2. Creating Dshell Models

The steps to build a Dshell Model are:

1. Create a folder to hold the model.:
  - The folder should be named \*Models (e.g. GeneralModels, IdealModels, etc.). This folder name is also referred to as the "module" name.
  - You can put more than one model in the same folder.
2. Create a .mdl file to specify the inputs/outputs.

Tip: For examples of .mdl files, look in the `src/GeneralModels` and `src/IdealModels` modules.

For the syntax of the .mdl file, please see this guide: `models_mdl_file`
3. Create a Makefile.yam file.
4. Run Makefile.yam to auto-generate stub code and the Dshell interface code.
5. Edit the stub code to insert your custom code.
6. Re-run Makefile.yam.

##### 1.6.2.2.1. Creating an .mdl file

The .mdl file contains the model specifications.

The .mdl file has the following advantages:

- Automatic construction of data structures
- Model information is human readable
- Model information can be parsed/introspected
- Separates boilerplate from user-written model code
  - Autogenerated code simplifies infrastructure upgrades
  - Developer can concentrate on model-related code

This is best illustrated with an example. Let's say you want to build a Solar Panel Battery model with the following specs:

- Parameters (data initialized only once at start up)
  - A scaling factor to convert one simulation "tick" to hours.
  - Nominal power storage in watt-hours.
  - Battery charging efficiency.

- Inputs (data that changes at run-time)
  - Solar panel charging level in watts
  - Power load in watts
- Outputs (data computed by the model based on inputs and parameters)::
  - Current battery power level in watt-hours
- States (internal state data)::
  - Current battery power level in watt-hours

The Battery.mdl file would look something like this:

```
# File: BatteryModels/Battery.mdl
# Specify Model type.
# Valid values for "Kind": "flow", "actuator", "sensor", "motor" or "encoder"
[Type]
Name = Battery
Kind = flow
Keywords = {Resource model!Power} {Power model} {Battery} {Rover!Power}

# Summary and long description for .pod file
[Documentation]
Brief description = ""Battery model""
Long description = ""This model sums up the draining and charging volts/sec using a nominal value""

# class parameters
[params]

[[tickFactor]]
Type = double
Length = -
Brief description = ""scaling factor to convert ioStepSize() to hours/tick ""
Long description = """"

[[totalPower]]
Type = double
Length = -
Brief description = ""Nominal power storage watt-hours""
Long description = """"

[[efficiency]]
Type = double
Length = -
Brief description = ""battery charging efficiency (1.0 means 100% of input (src) power charges the battery)""
Long description = """"

# class states
[states]

[[powerLevel]]
Type = double
Length = -
Brief description = ""current power level (watt-hours)""
Long description = """"

# class flowIn variables
[flow_in]
[[src]]
Type = double
Length = -
Brief description = ""charge source from solar panel (watts)""
Long description = """"

[[load]]
Type = double
Length = -
Brief description = ""power load from components (watts)""
Long description = """"

# class flowOut variables
[flow_out]

[[powerLevel]]
Type = double
Length = -
Brief description = ""battery's power storage level watts-hr""
Long description = """"
```

For a more detailed description of the syntax of the .MDL files, please see this description:

single: Dshell Model; Model Description Language File (.mdl) Syntax

1.6.2.2.1.1. Dshell Model Description Language File (.mdl) Syntax  
The Dshell Model Description files (.mdl) describe the model.

The Model Description File (.mdl) contains information about the model grouped by keywords. The .mdl files are in Python ConfigObj format (see <http://wiki.python.org/moin/ConfigObj> for details). The file is read by a Python script and uses a number of Python specific conventions (such as how Python treats and declares character strings).

### 1.6.2.2.1.1.1. Model Description File Sections

Supported sections:

| Section         | Definition   |
|-----------------|--|
| [Type]          | Tells the processor what type of model it is going to be.  |
| [Documentation] | Allows the model developer to enter in some information on the intent of the model as well as a little background  |
| [scratch]       | Defines scratch/temporary variables  |
| [flow_in]       | Defines variables as model inputs. Model inputs can be instructions to the model (aka variable settings) or data the model needs to operate.   |
| [flow_out]      | Defines variables as model outputs   |
| [params]        | Defines variables as model parameters. Parameters are values set up values read in or set up by the configuration script at start up, but are not changeable by the model itself.  |
| [states]        | States are initialized at startup, and may be modified by both the model and the user during run-time. These are 'discrete' states.  |
| [cont_states]   | Continuous states are updated by the numerical integrator in Darts, and require the model developer to fill in a method for computing the derivatives of these states. Continuous states are either doubles, or arrays of doubles. |
| [structs]       | Defines C-style structures that can be used in other sections (such as 'scratch').   |
| [enums]         | Define the specific enumeration constants that will be available for the C++ code to use to use enum tokens instead raw integer values. See the GeneralModels/ExternalDisturbance model for an example.                            |
| [commands]      | Older models sometimes still use this older technique of providing inputs to models at run-time. Newer models use signals for similar functionality.   |
| [outputs]       | Older models sometimes still use this older technique of providing outputs from models at run-time. Newer models use signals for similar functionality.  |

Each key word has their own specific items that must be included, but there are some common elements.

#### 1.6.2.2.1.1.1.1. The 'Type' section

In the **[Type]** section, the developer can describe the nature of the model. A type section might look like this:

```
[Type]
Name = AsphericalGravity
Kind = actuator
Keywords = {Gravity!J2 term} {Gravity!J3 term}
```

The 'Name' field is the name of the model and all the C++ files associated with the model will use this for the name of the files.

#### The 'Kind' is the kind of model. Allowed kinds are

- 'actuator'
- 'sensor'
- 'motor'
- 'encoder'
- 'flow'

The 'Keywords' field allows the developer to add keywords that can be used to identify and sort the models. Multiple keywords can be added, as shown. The keywords are divided into a hierarchy using the exclamation symbol, as shown.

The 'Type' section also supports the 'BaseModel' field. If the model is to be derived from an existing model, that model is identified using the 'BaseModel' field. The format is:

“*BaseModel = Module/ModelName*

For examples, see the models in the 'EDLAeroModels' module.

#### 1.6.2.2.1.1.1.2. The 'Documentation' section

In the 'Documentation' section, the developer can provide documentation for the model. An example might look like this:

```
[Documentation]
Brief description = ""
Long description = ""
The AsphericalGravity actuator is a gravity model which inherits all properties
of the PointMassGravity model and adds aspherical compensation terms to it.
Currently, only the J2 and J3 zonal harmonics are supported. (See Wertz,
"Spacecraft Attitude Determination and Control")""
```

The 'Brief description' is generally a short description used in lists, etc. The 'Long description' can be much longer and more detailed.

#### 1.6.2.2.1.1.1.3. The data sections

The [scratch], [states], [cont\_states], [params], [flow\_in], and [flow\_out] sections have a similar format (excerpts from AsphericalGravity model):

INI

```
[params]

[[G]]
Type = double
Length = -
Brief description = ""Universal gravitational constant""

[scratch]

[[Accel]]
Type = double
Length = 6
Brief description = ""acceleration due to gravity""
Long description = """"

[states]

[[BodyPosition]]
Type = double
Length = 3
Brief description = ""body inertial position vector (J2000)""
Long description = ""position vector of the attracting body in an inertial frame (J2000)""
```

Each variable definition has the name of the variable in double-brackets (as shown).

The 'Type' is the data type of the variable. Known types are the typical C++ data classes such as: int, long, short, char, uint, ulong, ushort, uchar, double, float, string, etc. The names of previously defined 'structs' and 'enums' can also be used for the type.

Note that signals only support a subset of these data types as described in `signals`: int, bool, double, and string. Only vectors of 'int' and 'double' are currently supported.

The 'Length' field indicates the size of the data:

“ '1' means it is a scalar

'3' means it is a 3-vector

'\*' means it is vector of unspecified size

The description fields apply to the associated variable and can be used to provide more detailed documentation for the variable.

Thanks to the JSC team for providing an early version of this documentation for .MDL files.

#### 1.6.2.2.2. Makefile structure for model modules

The next step is create a Makefile (which should be named "Makefile.yam") to build the model.

Example of a Makefile for a module called "BatteryModels":

```
#####
# File: BatteryModels/Makefile.yam
# To run this makefile, type "make -f Makefile.yam all"
#####

# Add any additional (external) object files (*.o) or libraries (lib*.a or lib*.so) here
ADDT_OBJJS =

# Set MODEL_COMPILE_FLAG to any additional compilation flags that need to be passed
# on to the compilation process
export MODEL_COMPILE_FLAGS =

# -----
# Use the standard Makefile.yam for model libraries
# -----
ifndef YAM_ROOT
include ../../etc/SiteDefs/Makefile.yam.model
else
ifeq ($(YAM_ROOT)/etc/SiteDefs/Makefile.yam.model,$(wildcard $(YAM_ROOT)/etc/SiteDefs/Makefile.yam.model))
include $(YAM_ROOT)/etc/SiteDefs/Makefile.yam.model
endif
endif

# -----
# when building binary executables and shared libraries, list any extra
# libraries that must be linked in and any -L options needed to find them.
# -----
CFLAGS-libBatteryModels += -I/usr/include

# the following defines the import command to be added to the
# auto-generated Python file to pull in required modules
# PYTHON_IMPORTS := OLDMath_Py Dutils.Spice_Py
PYTHON_IMPORTS :=

# You can export any file to the etc/ folder
ETC_MODULE_LINKS :=
```

### 1.6.2.2.3. Generating C++ stub code

Use the following command to generate the C++ stub code:

```
gmake -f Makefile.yam links
```

The following files will be generated:

- Battery.cc, Battery.h  
Stub code. You need to edit this code.
- auto/BatteryIF.cc, auto/BatteryIF.h  
Dshell interface code. Do not edit these files.

### 1.6.2.2.4. Editing the auto-generated Dshell Model C++ file

You will need to edit the auto-generated files (Battery.cc, Battery.h) to customize the model. You normally need to do this:

- Set the default parameter values in the constructor `Battery::Battery()`. Make sure all strings (if any) are properly null terminated.
- Initialize local variables in the `Battery::setup()` method. The `setup()` method is called after Dshell has called the constructors of all models.
- If you allocate memory (e.g. call `malloc()` or `new`), you should clean up memory in the destructor `Battery::~~Battery()`.
- Whenever the user changes a parameter, Dshell notifies the model that a parameter has changed by calling the model's `processParams()` method.
- For Flow models, Dshell calls the model's `tick()` method at every I/O step so the code to compute the model's outputs should be placed here.

Here is the `BatteryModel.cc` filled out to compute the desired outputs:

```

/*
 * Battery.cc -- C++ source file
 *
 * User will always edit this file, to fill in code for methods
 */

#include "Battery.h"

/*
 * Battery - This is the constructor method, which is called
 * immediately after the model is allocated. Use it to set
 * default values for states and parameters, which will be
 * overridden by any values specified in the model file and
 * initial state file.
 */
Battery::Battery(std::string modelName) : BatteryIF(modelName)
{
    /*
     * INITIALIZE BatteryParams DATA STRUCTURE -- this sets default values
     * in case they are not specified in the model input file
     */
    params()->tickFactor = 1/3600; //scaling factor to convert ioStepSize() to hours/tick
    params()->totalPower = 69.1; // watts hr
    params()->efficiency = 100.0; // battery efficiency
    states()->powerLevel = params()->totalPower;
}

/*
 * ~Battery - This is the destructor method, which is called
 * before the model is deallocated. Use it to deallocate any
 * memory allocated by the model.
 */
Battery::~Battery()
{
}

/*
 * setup - This method is called to compute the values of
 * "derived" parameters, which are parameters whose
 * values depend on those of other parameters.
 *
 * This method is called only once, after the constructor
 * and after values for parameters from the model file
 * are set. Do NOT set default values for non-derived
 * parameters in this method - it will override values
 * provided in the model file.
 */
void
Battery::setup()
{
    states()->powerLevel = params()->totalPower;
}

/*
 * tick - This method is called once in each I/O step.
 */
void
Battery::tick()
{
    double pwr;
    double deltatime;

    deltatime = ioStepSize();
    deltatime *= params()->tickFactor;
    // determine charging or draining power to/and from the battery
    pwr = *(flowIns()->src) * params()->efficiency - *(flowIns()->load);

    states()->powerLevel += pwr*deltatime;
    if (states()->powerLevel > params()->totalPower)
        states()->powerLevel = params()->totalPower;
    if (states()->powerLevel < 0.0)
        states()->powerLevel = 0.0;

    *(flowOuts()->powerLevel) = states()->powerLevel;
}

/*
 * processParams - This method is called when:
 * -- a parameter is changed with the 'Dparam' Tcl command
 * -- once during initialization after the 'setup()' methods are called
 * -- when the C function 'BatteryParamsSet()' is called by simulator
 * Its purpose is to update the values of scratch variables
 * when a parameter changes.
 */
void
Battery::processParams()
{
}

```

### 1.6.2.2.5. Compiling the Model

Run the Makefile.yam to build the model library object files:

```
gmake -f Makefile.yam all
```

This will create two files:

1. BatteryModels\_Py.so is a loadable Python module
2. libBatteryModels.so contains the compiled C++ code

#### 1.6.2.2.6. Example Python Script

Here is an example model.py and Python script to exercise the Battery model:

tutorials/modelCreation

#### 1.6.2.3. Advanced Topic

- When you run a python script, how does it know about various models?
  - Auto-generate python `pyIndex` file in 'auto' directory
  - Each module `pyIndex` includes rules to load the module for specific model names (model 'A' is in module 'X')
  - All `pyIndex` files are linked into the `lib/ModulesTclIndex/` directory of your sandbox
  - When you run scripts in your sandbox
    - All 'pyIndex' files in `lib/ModulesTclIndex/` are processed (note: the name is for historical reasons; ModulesTclIndex has nothing to do with Tcl).
    - At runtime, when you request model 'A', it knows to import module 'X'
  - SUMMARY: Any model defined in a module in your sandbox will be available automatically from python; you do not need to do anything extra.
- Basics of model inheritance

Dshell Models can be derived from other models. As an example, let's say we want to extend the `{Dshellpp_Battery.mdl_script_uri}{Battery.mdl}` to add a new output to trigger an alarm if the battery level falls below a certain level.

The `.mdl` file for this new model would look like:

```
# File: BatteryModels/Battery2.mdl
# This model is derived from BatteryModels/Battery.mdl.
# All parameters, states, inputs, outputs, etc defined in Battery.mdl
# will be accessible from Battery2.
[Type]
Name = Battery2
BaseModel = BatteryModels/Battery

# add new parameters for this model
[params]
[[alarm_threshold]]
Type = double
Length = -
Brief description = ""Alarm will be triggered if power falls below this level (watt-hours) ""
Long description = """"""

# add new outputs
[flow_out]
[[alarm]]
Type = int
Length = -
Brief description = ""1 if alarm is set, 0 if alarm is off""
Long description = """"""
```

You can then auto-generate the `Battery2.cc` and `Battery2.h` stub code from the `.mdl` file with the command:

```
gmake -f Makefile.yam links
```

If you examine the auto-generated `Battery2.cc` file, you will see how the derived class methods call the base class methods. For example, this is the stub code for `Battery2::tick()`:

```
void Battery2::tick()
{
    Battery::tick(); // calls base class method
}
```

C++

You can modify this to add your own code to set the alarm:



```

void Battery2::tick()
{
    Battery::tick(); // calls base class method

    // Code to set alarm.
    // Battery::states()->powerLevel refers to the powerLevel state
    // defined in the base class.
    if (Battery::states()->powerLevel < params()->alarm_threshold)
        *(flowOuts()->alarm) = 1;
    else
        *(flowOuts()->alarm) = 0;
}

```

The command to compile the .cc code would be:

```
gmake -f Makefile.yam all
```

### 1.6.3. Dshell Signals

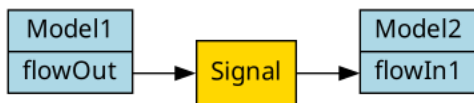
#### 1.6.3.1. The Purpose of Signals

Dshell signals are basically C++ data buffers that can be used to communicate data between various entities at runtime:

- Model-to-model



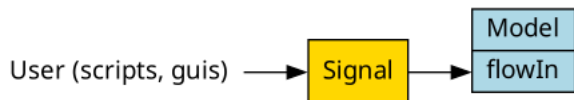
TBD: The above figure should have been autogenerated by graphviz (not working):



- User inputs into models (from runtime scripts or guis)



TBD: The above figure should have been autogenerated by graphviz (not working):



- User-accessible outputs from models



TBD: The above figure should have been autogenerated by graphviz (not working):



During the initialization of a simulation, all model inputs and outputs must be tied to signals.

#### 1.6.3.2. Types of Signals

Each Dshell signal (`Signal`) allocates and manages a memory buffer: The signal memory is accessible via a Dvar variable. So signal information can be retrieved or set using standard Dvar operations (see the DVar section).

Only a limited subset of all Dvar types are supported as signals. These include the following scalar values:

- int

- bool
- double
- string

Signals that are vectors are also supported for two types:

- vector of ints
- vector of doubles

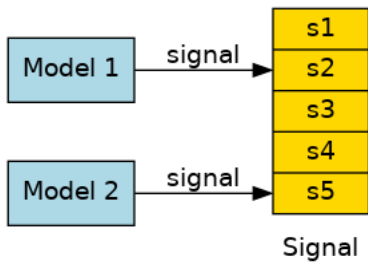


**Note**

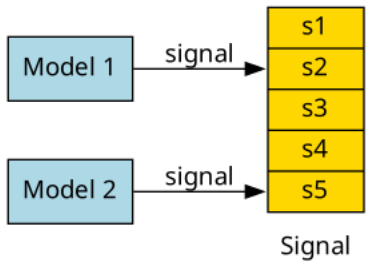
For simplicity, all scalar-valued signals are represented as vectors of length 1.

### 1.6.3.3. Signal Slices

A signal can be an array of data, and 'Signal Slices' provide a way to split out a part of the array as a separate signal. That way parts of a signal array can be sent to different models. Or data from several models can be combined into a larger signal array.



TBD: The above figure should have been autogenerated by graphviz (not working):



Signal slices can be created in several ways in python.

- In assemblies code, signal slices can be retrieved easily. To get a complete signal:

```
signal = self.getSignalSlice('signalName')
```

PYTHON

where 'self' is the assembly object. To get just a subset of the signal:

```
slice = self.getSignalSlice('signalName', [0,1])
```

PYTHON

where the range list is [ $\langle$ start-index $\rangle$ ,  $\langle$ end-index $\rangle$ ]. This will construct a subset of the signal that only has the first two entries.

- An alternate syntax is available when accessing signals via Dvar: Given a spec\_node that contains a signal named 'signalName':

```
signal = spec_node['signalName']
```

PYTHON

To get the subset (as above):

```
slice = spec_node['signalName(0-1)']
```

PYTHON

where the range is specified as part of the signal name as shown.

### 1.6.3.4. Advanced Signal Topics

#### 1.6.3.4.1. Signal Granularity

How often a signal is refreshed depends on the model that outputs data into the signal. The model granularity flag specifies when the DshellModel::updateFlowOuts() method is called to write data to the signals tied to the model's output ports. The following flags (which are bit flags which may be combined) are available:

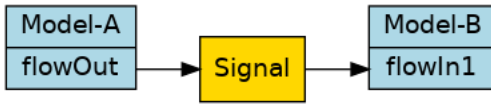
- GRANULARITY\_PREDERIV (update after every preDeriv() call )
- GRANULARITY\_POSTDERIV (update after every postDeriv() call )
- GRANULARITY\_ENDINTEGRATION ( update after endIntegration() step )
- GRANULARITY\_ENDIO ( update at end of i/o step )

A model usually sets the granularity using the `DshellModel updateFlowOutsGranularity<DartsModel::updateFlowOutsGranularity>` function during model initialization. Note: `FlowModel` models do not support setting granularity.

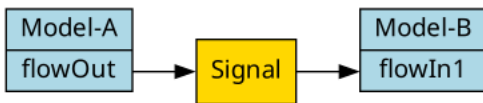
#### 1.6.3.4.2. Model Sorting

When we process models during integration, it is important to evaluate the models in the correct order so the inputs to a model are as current as possible.

Suppose Model-A generates an output which is connected to an input of Model-B via a signal.



TBD: The above figure should have been autogenerated by graphviz (not working):



Clearly, the computations that Model-B performs normally depend on the input it receives from Model-A. Therefore, Model-A should send out new data to the signals connected to its outputs before Model-B performs its computations. This implies that Model-A should appear before Model-B when model updates are run in the main simulation loop (such as `preDeriv()` calls).

Dshell uses a graph-sort technique to make sure that each model receiving data from another model is invoked after its 'up-stream' model. In cases where the dependencies are circular loop, the creator of the models must specify where to break the loop. The decision on where to break such as loop requires an analyst to evaluate the models involved and decide where the break should be done logically. The resulting decision is usually incorporated into the Assembly code.

#### 1.6.3.4.3. Signal Freshness

Some models may require a significant amount of computation. If the inputs to the model (signals) have not changed, there may be no need to re-perform the computations to update the models outputs.

For instance, a signal that initiates a motor command should not be repeated each time the model is executed. The model needs to recognize when it receives a new motor command (via a signal) and only deal with new motor commands once.

So signals have a mechanism that supports determining whether the signal value has changed since the last time the model accessed the signal. The concept is *signal freshness*.

In practice, *signal freshness* is simply an integer that is incremented each time the signal is updated. Each signal keeps its own freshness index. Each flowIn port of a model keeps track of the value of the freshness index for its associated signal and can therefore tell if the signal has changed. Some models take advantage of this.

For an example of a model that uses *signal freshness*, see the `SetBodyMass` model. It uses signal freshness to keep from updating the body mass unless the input signal for mass has really changed.

Note: There are some examples of accessing signal freshness near the end of the tutorial below.

### 1.6.4. Dshell Assemblies

#### 1.6.4.1. Motivation for Assemblies

When modeling a system, it is common to have multiple instances of a type of subsystem.

- For wheeled systems:

## Recurring subsystems in robotic ground vehicles



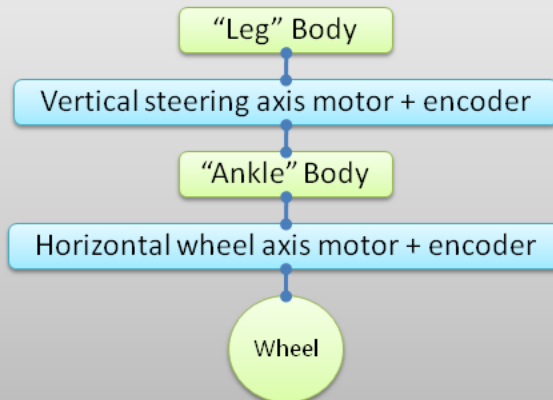
### Similar Wheel Subsystems:

- Identical topology
- Differing parameters

### Goal:

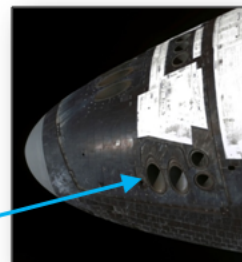
- Create reusable "assemblies" to model generic subsystems

### Generic Wheel Subsystem



- For spacecraft:

## Recurring subsystems in spacecraft



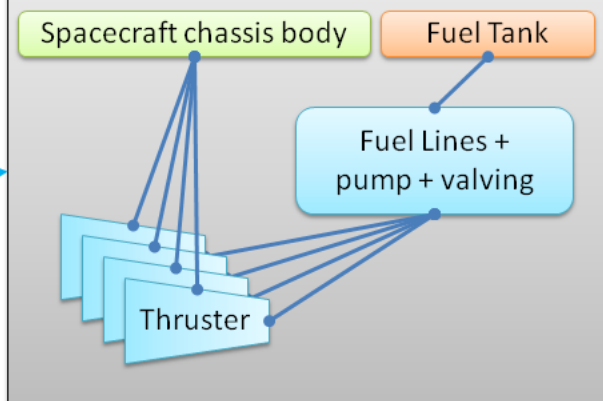
### Similar Thruster Subsystems:

- Identical topology
- Differing parameters

### Goal:

- Create reusable "assemblies" to model generic subsystems

### Generic Thruster Subsystem





Assemblies let us prepackage subsystems in a reusable way!

#### 1.6.4.2. Assembly Basics

- What are **Assemblies**?

**Assemblies** are hierarchical building blocks for organizing simulation components.

**Assemblies** are reusable containers for groupings of signals, bodies, models, and subassemblies.

- Purpose of Assemblies
  - Create models, signals, bodies, and subassemblies at runtime
  - Connect model flowIns/flowOuts with signals
  - Contain other assemblies as needed
  - Encourage loose coupling of models
    - Assemblies are more self-contained so that updates to one have less chance to affect others
    - Improves code maintainability
  - Encourage reusable code
    - Once an Assembly is written, it can be reused easily
    - Assemblies capture domain area expertise
    - Tests specific to the assembly can be written to assure correct functionality and improve maintainability
  - Hierarchical Assemblies encourage reuse
    - Assemblies for sub-components (eg, thrusters) can be reused in a variety of sub-systems ( single thrusters, thruster sets )
  - Share assemblies across different types of systems
  - Provide subsystem-level functionality

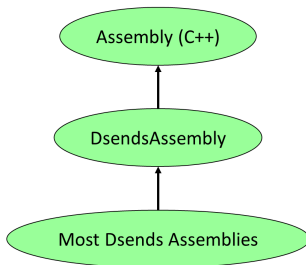
##### 1.6.4.2.1. Assembly Architecture



To see a presentation on Assembly Design and Creation, please see:

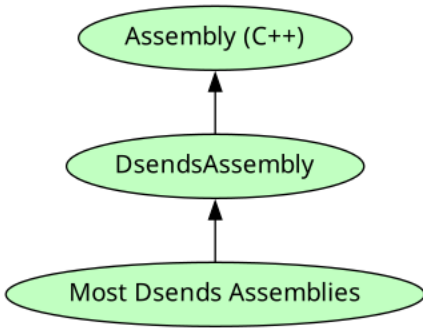
- <link:///extras/DLabDocs/src/Dshell++/doc/source/documents/Assembly-Design-and-Creation-2014-08-26.pdf>[Assembly Design and Creation (August 2014 TIM)].

- An assembly is meant to be a self-contained unit.
  - Creates all the models, signals, bodies and sub-assemblies it needs
- There can be more than one instance of an assembly type
  - e.g. multiple thruster assemblies
- All assemblies are derived from the C++ Assembly class



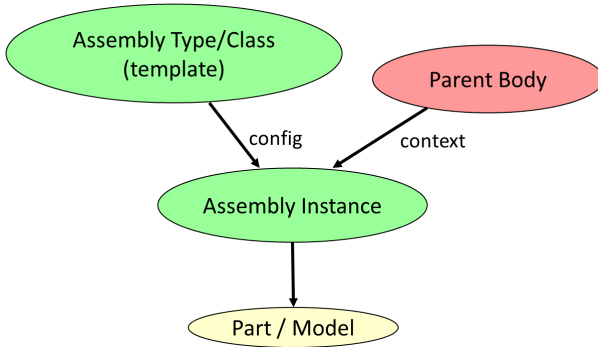
**TBD:** The above figure should have been autogenerated by graphviz (not working):

+



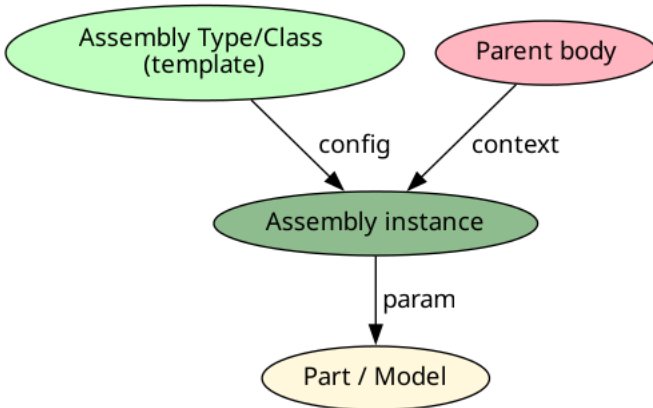
+ Assembly logic is easier to implement/maintain in Python Conceptually possible to make Assemblies out of all C++ \*\* Dartslab Assemblies are all Python

- Assemblies are hierarchical
  - Assemblies can contain other Assemblies
- Assemblies have params, config and context attributes
  - **config**: Specifies configuration properties for the assembly (eg. number of thrusters)
  - **params**: Configuration parameters (eg. body mass, geometry)
  - **context**: Context information needed to embed/connect the assembly with its parent (eg. parentBody)



TBD: The above figure should have been autogenerated by graphviz (not working):

+



- Assemblies often support variants that can be selected by configuration parameters
  - Trade-off with options vs derived classes

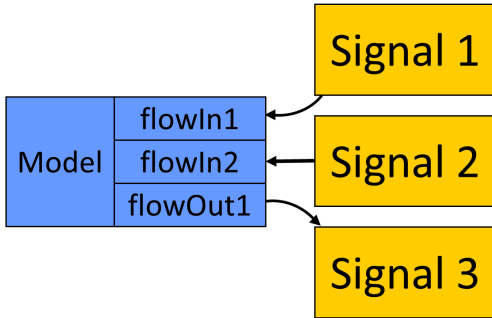
#### 1.6.4.2.2. Assembly Parameters

- Assembly parameters classes are based on the **paramTypes** base class
  - Defines required and optional parameter elements
  - Contains description and units information for the elements
  - May enforce type checking
  - May have extra class methods (eg. rover dimensions class) to derive additional data
  - Generic parameter instances as well as for specific hardware (eg. crossbow IMU, planet earth, Thruster parameters)

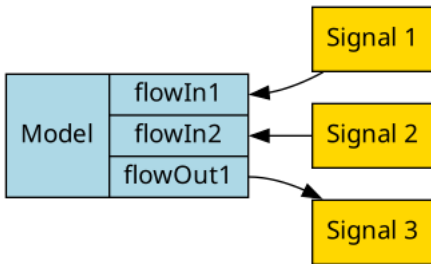
- Lazy binding of parameters

### 1.6.4.2.3. Assembly I/O (Signals)

- Assemblies define the models and signals they contain
- Assemblies create signals for models to communicate with each other
- Signals serve as the interface objects to inter-connect assemblies/models
  - Between unassociated assemblies (via global signals)
  - Between associated assemblies
    - parent/child/sibling assemblies
    - child assembly signals tied to signals in a parent assembly
- Assemblies connect model flowIns and flowOuts to signals



TBD: The above figure should have been autogenerated by graphviz (not working):



- Who creates/owns the interface signals?
  - The assembly that created it (or parent assembly)

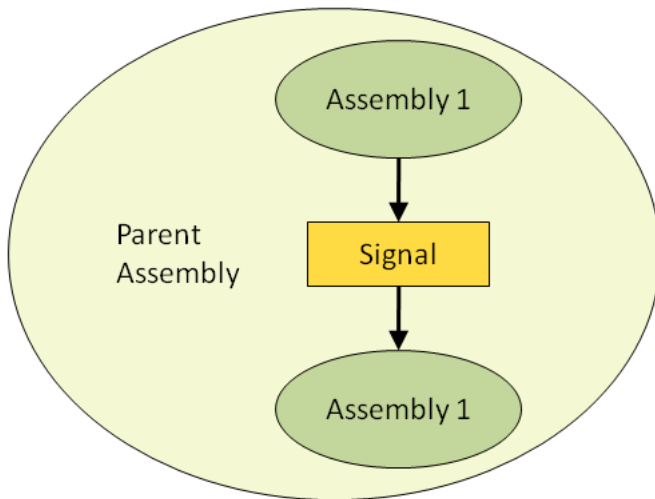


Figure 27. A signal that is used to communicate between child assemblies should be created by the parent assembly and passed down to each child assembly. See Signal Mapping for more details on this.

### 1.6.4.3. Stages of Assembly Construction



*Note*  
Examples from EDL/python/models/dyn/fuelTankAssembly.py (unless otherwise noted)

As an Assembly is being created, the following methods are called automatically in this order during initialization.

- addSignals()
- addModels()
- addAssemblies()

After initialization, it is up to the user-run script to call these functions:

- params() - To store the raw values of the parameters
- bindParams() - To process the parameters and apply the various parameter values to associated bodies and models.
- bindState() - To do final pre-run initialization

#### 1.6.4.3.1. Add signals

Add any signals needed by the Assembly models or by models of sub-assemblies

```
def addSignals(self):
    name = self.name()
    self.addSignal('FuelUsed', { 'length': self.maxlines, 'type': 'double',
                                'comment': 'Fuel line draws on ' + name + ' tank' })
    self.addSignal('FuelRemaining', { 'length': 1, 'type': 'double',
                                      'comment': 'Fuel remaining in ' + name + ' tank' })
```

PYTHON

For more information about signals, see the Dshell Signals section.

#### 1.6.4.3.2. Add models

Add models for the assembly

- Create needed bodies and any needed nodes
- Add models
- Define which signals connect to which flowIns and flowOuts

```
def addModels(self):
    name = self.name()

    # Add the body
    from Dshell import DartsCommon
    mbody = DartsCommon.DartsMbodyLookup()
    self.body = DartsCommon.DartsBody.asBody( mbody.body(context['body'], 0, True ))

    # Add the actuator node
    self.node = self.body.createActuatorNode(self.nodeName)

    # Add the fuel tank model
    self.addModel(name, 'actuator', {
        'MODEL_TIES': {
            'entry0': { 'ioName': 'fuel_burned',
                       'ioSlice': [0, self.maxlines-1],
                       'ioType': 'flowIn',
                       'signal': 'FuelUsed'},
            'entry1': { 'ioName': 'fuel_remaining',
                       'ioSlice': [0,0],
                       'ioType': 'flowOut',
                       'signal': 'FuelRemaining'},
        },
        'class': self.config()['model'],
        'node': self.node,
        'PARAMS': {},
    })
```

PYTHON

NOTES:

- The second argument ('actuator' here) tells the assembly what kind of model to add. Supported types of models include:
  - 'actuator'
  - 'motor'
  - 'flow'
  - 'sensor'
  - 'encoder'
 See Dshell Models section for more information on these model types
- The **MODEL\_TIES** entries tells the autocoder how to connect the model flowIns and flowOuts with the available signals.
- The **class** entry should resolve to the name of a model class. See Dshell Model name resolution section to understand how the model names are resolved at run-time.
- The **node** entry should resolve to a `DartsNode` object that the object will be attached to.

#### 1.6.4.3.3. Add subassemblies

Since assemblies are potentially hierarchical, any assembly can add subassemblies.

Example from ThrusterSetAssembly (abridged):



```
def addAssemblies(self):
    childTies = self.createChildTies1()
    context = { 'parentBody' : self.parentBody }
    for thrusterName in self.config()['model']:
        config = self.config()['model'][thrusterName]
        self.addSubAssembly( thrusterAssembly ( thrusterName,
                                                config, context,
                                                childTies ) )
```

## NOTES:

- The *context* is used to provide the parent body that the thruster set will be attached to.
- The *config* is used to provide configuration information about the assembly itself. In this case, it provides the name of the ThrusterSet.
- the 'childTies' argument to the 'addSubAssembly' function is important.

1.6.4.3.3.1. Signal Mapping  
Signal mapping

How do parent assemblies access signals from child assemblies?

As mentioned in the xef:Sphinx\_Dshellpp\_assembly\_io\_signals[Assembly I/O (Signals)] section, a parent assembly can instruct its child assemblies to use a signal that it (the parent assembly) provides in the place of a signal that the child assembly would normally create.

Suppose a child assembly 'B' creates a model with a flowout signal called 'b\_out'. Now suppose that this signal needs to be sent to a model in a peer assembly, 'C'. In model in assembly 'C', the flowin signal is called 'c\_in'. The normal way to accomplish this is for the parent assembly, 'A', to create a signal, 'a', and passes it down to both children so that they use it.

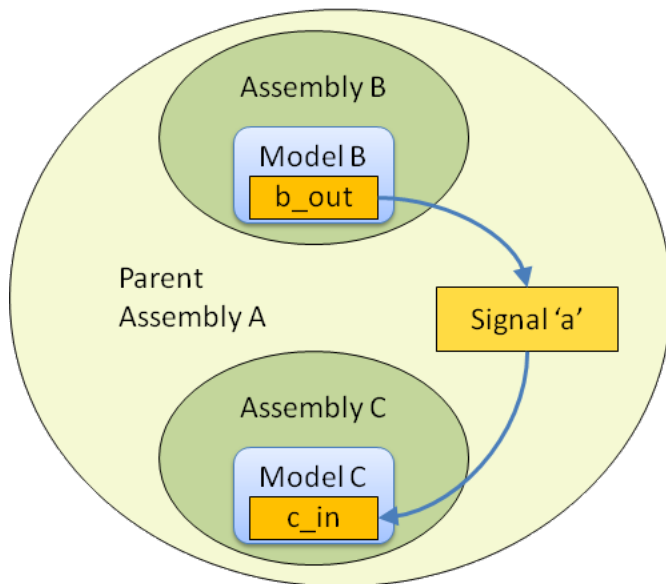


Figure 28. Assembly A creates a signal 'a' that it provides to child assemblies B and C so that they can communicate with each other.

In model 'B', the flowout is named 'b\_out'. In the assembly code that creates model 'B', it will create a signal to attach to that flowout. Its add signals code would look like this:

```
def addSignals(self):
    """ for assembly B """
    self.addSignal('model_b_output' , { 'length': 1, 'type': 'double', 'comment': '' })
```

Similarly for Assembly C:

```
def addSignals(self):
    """ for assembly C """
    self.addSignal('model_c_input' , { 'length': 1, 'type': 'double', 'comment': '' })
```

And model 'B' would instruct the model to connect this 'model\_b\_output' signal to the flowout of the model:

```
def addModels(self):
    """ for assembly B """
    self.addModel('ModelB', 'actuator', {
        'class' : 'ModelBClassName',
        'MODEL_TIES' : {
            'entry0' : { 'ioName': 'b_out',
                        'ioSlice': [0,0],
                        'ioType': 'flowOut',
                        'signal': 'model_b_output'},
        },
        'node' : nodeObject
    })
```

Similarly for Assembly C:

```
def addModels(self):
    """ for assembly C """
    self.addModel('ModelC', 'actuator', {
        'class': 'ModelCClassName',
        'MODEL_TIES': {
            'entry0': { 'ioName': 'c_in',
                       'ioSlice': [0,0],
                       'ioType': 'flowIn',
                       'signal': 'model_c_input'},
        },
        'node': nodeObject
    })
```

PYTHON

In order for the parent assembly to connect these two models in the desired way, several small pieces of code are necessary.

1. In the parent assembly 'addSignals' method, we create the necessary shared signal, named 'a':

```
def addSignals(self):
    """ for parent assembly A """
    self.addSignal('a', { 'length': 1, 'type': 'double', 'comment': 'shared signal' })
```

PYTHON

2. In the parent assembly 'addAssemblies section', when we create assembly B (with model B), we instruct assembly B to use the signal created by the parent (signal 'a') instead of the signal it would have created for its output. Similarly, we instruct model C to use the provided signal ('a') instead of the input signal model C would have created:

```
def addAssemblies(self):
    """ for parent assembly A """

    # Add assembly B
    signal_map = { 'model_b_output' : self.getSignalSlice('a') }
    feed_thru = []
    child_ties_b = self.updateTies(self.signalTies(), signal_map, feed_thru)
    self.addSubAssembly(AssemblyBClass( 'asm_B', config={}, context={}, child_ties_b))

    # Add assembly C
    signal_map = { 'model_c_input' : self.getSignalSlice('a') }
    feed_thru = []
    child_ties_c = self.updateTies(self.signalTies(), signal_map, feed_thru)
    self.addSubAssembly(AssemblyBClass( 'asm_C', config={}, context={}, child_ties_c))
```

PYTHON

When assemblies B and C are created as shown here, they do not create their own signals to connect to their model's flowins or flowouts; they use the ones provided by the parent instead.

The **feed\_thru** list is a list of signal names can be used to pass shared signals through this assembly so it can be used by assemblies that are children of the direct children assemblies (B and C here). It is a list of names of signals that the parent assembly has been provided when it was created by its parents. In other words the when a signal is shared, the parent creating the shared signal passes it to its children using the 'child\_ties' approach shown here. To pass the signal through assemblies B and C, they have to use the 'feed\_thru' list when they create their own children assemblies.



#### Note

The code snippets in this section are simplified and incomplete and are only for illustration purposes.

#### 1.6.4.3.4. Bind parameters

Collect parameters and update parameters in models and subassemblies

```
def bindParams(self):
    name = self.name()
    param_set = self.params()

    # Update model parameters
    self.modelParamSet(name, 'num_thrusters', self._numFuelLines)
    self.modelParamSet(name, 'starting_fuel', 'Tank.startingFuel', param_set)
    self.modelParamSet(name, 'empty_mass',
                       param_set['Body']['mass']() - param_set['Tank']['startingFuel']())

    inval = param_set['Body']['cmInertia']()
    invalflat = inval[0] + inval[1] + inval[2] # flatten matrix to a 9 vec
    self.modelParamSet(name, 'initial_inertia', invalflat)
    self.modelParamSet(name, 'empty_inertia', emptyInertiaSum )
    self.modelParamSet(name, 'initial_CM', [0,0,0])
    self.modelParamSet(name, 'empty_CM', 'Tank.emptyCM', param_set)

    # Set up parameters for the fuelTank body
    vec = SOA_Py.SOAVector3(param_set['Node']['bodyToNode']())
    vec.thisown = False
    self.node.setBodyToNode(vec)

    # Bind any parameters of subassemblies
    DsendsAssembly.bindParams(self)
```

PYTHON

NOTES:

- There are three aspects of binding parameters

- Updating the parameters for any models created by the assembly (lines 5-16)
- Updating the parameters for any bodies that the assembly created (lines 18-21)
- Instructing child assemblies to update their parameters (line 24)
  - In this example, there are no subassemblies so this could have been omitted.
  - It is necessary that parameters of all child assemblies be passed down to them by invoking the 'params()' function before executing the 'DsendsAssembly.bindParams(self)' function call.

#### 1.6.4.3.5. Set the initial state

The bindState() function is used to initialize models and signals belonging to the Assembly.

```
def bindState(self):
    name = self.name()
    model = self.config()['model']

    # model internal states
    if model in ['FuelTank',]:
        self.modelObj(name).specNode()['state.fuel_remaining'](0.0)
        self.modelObj(name).specNode()['state.sum_fuel_consumed'](0.0)

    # initialize signals
    self.signalSlice('FuelUsed').setToNominalValue()

    # initialize any subassemblies
    DsendsAssembly.bindState(self)
```

PYTHON

#### 1.6.4.4. Debugging Assembly design/coding problems

- Debugging assembly code development can be difficult
  - Transition from Python to C++ sometimes makes isolating problem difficult
  - 'print' is your friend!
- Debugging procedure
  - From your error messages, guess at the source of the problem
  - You may need to run with debugging:

```
> Dmain.py --file script.py --usegdb
```

SH

- Instrument main Assembly functions (addSignals(), addModels(), etc) by adding print messages at beginning and end of functions:

```
print '***** addModels', self.name()

[original code]

print '***** addModels done'
```

PYTHON

- Execute assembly code to isolate problem area
- Repeat process at lower levels until the C++ call is isolated
- At C++ call interface, examine inputs carefully for problems
- Other tips:
  - Use pformat to output data in useful ways

```
from pprint import pformat
print "-----"
print pformat(self.params()) print
"-----"
```

PYTHON

- Examine outputs for suspicious values

#### 1.6.4.5. Assembly Usage Tutorial

##### 1.6.4.5.1. Dshell Assembly Basics Tutorial

##### Dshell++ Assembly basic usage regtest

- ▶ [Click to see the Dshell++/test/test\\_Ndarts/test\\_assembly/script.py script](#)

#### 1.6.5. Dshell Parameter Classes

##### 1.6.5.1. Parameter System Basics

Simulation parameters are used to initialize the system at the start of the simulation. These parameters are set up by the user to build up spacecraft assemblies, set the parameters of the C++ models used in the simulation, control thresholds and triggers in the state machine, and generate auxiliary parameter sets for third-party simulation tools integrated into the simulation (such as MarsGram, Spice, etc).

In Dsends, this parameter information is managed by a set of parameter classes which provide functionality related to required and optional data, units, data types, and parameter source documentation.

Examples of parameters include:

- initial state
- spacecraft mass properties
- sensor or actuator properties
- GNC parameters
- Model modes

#### 1.6.5.2. Design Goals

- All parameters must be designed to have the same "look and feel" with regard to their specifications, instancing, unit systems, etc
- Parameters are provided by the user and may be
  - Required. The absence of a required parameter is an error
  - Optional. The absence of an optional parameter implies that the simulation system provides a default value
- The same parameter must not be multiply defined. i.e. a single defined value must "fan out" to all uses of the parameter
- Parameters for initializing third-party tools integrated into the simulation (e.g. GRAM atmosphere models) must be autogenerated from the parameter system.
- Parameters must be capable of being over-written in top-level scripts to support Monte-Carlo and Parametric Sweeps.

#### 1.6.5.3. Key Requirements

- The parameter classes must encapsulate the parameters into a hierarchy with each sub-class:
  - Augmenting the required parameters of the base class as needed.
  - Augmenting the optional parameters of the base class as needed.
  - Remapping those parameters that were optional for the base class but are to considered as required for the sub-class.
  - Providing pre-determined parameter values for the base class i.e. those that are not to be provided by the user.
  - Providing default values for the optional parameters that may be set by the users.
  - Defining and computing dependent parameters that are functions of the (independent) required and optional parameters in the class instance. Note that what is considered as a dependent parameter in one sub-class of a parent class may be an independent (required) parameter for another sub-class derived from the same parent class.

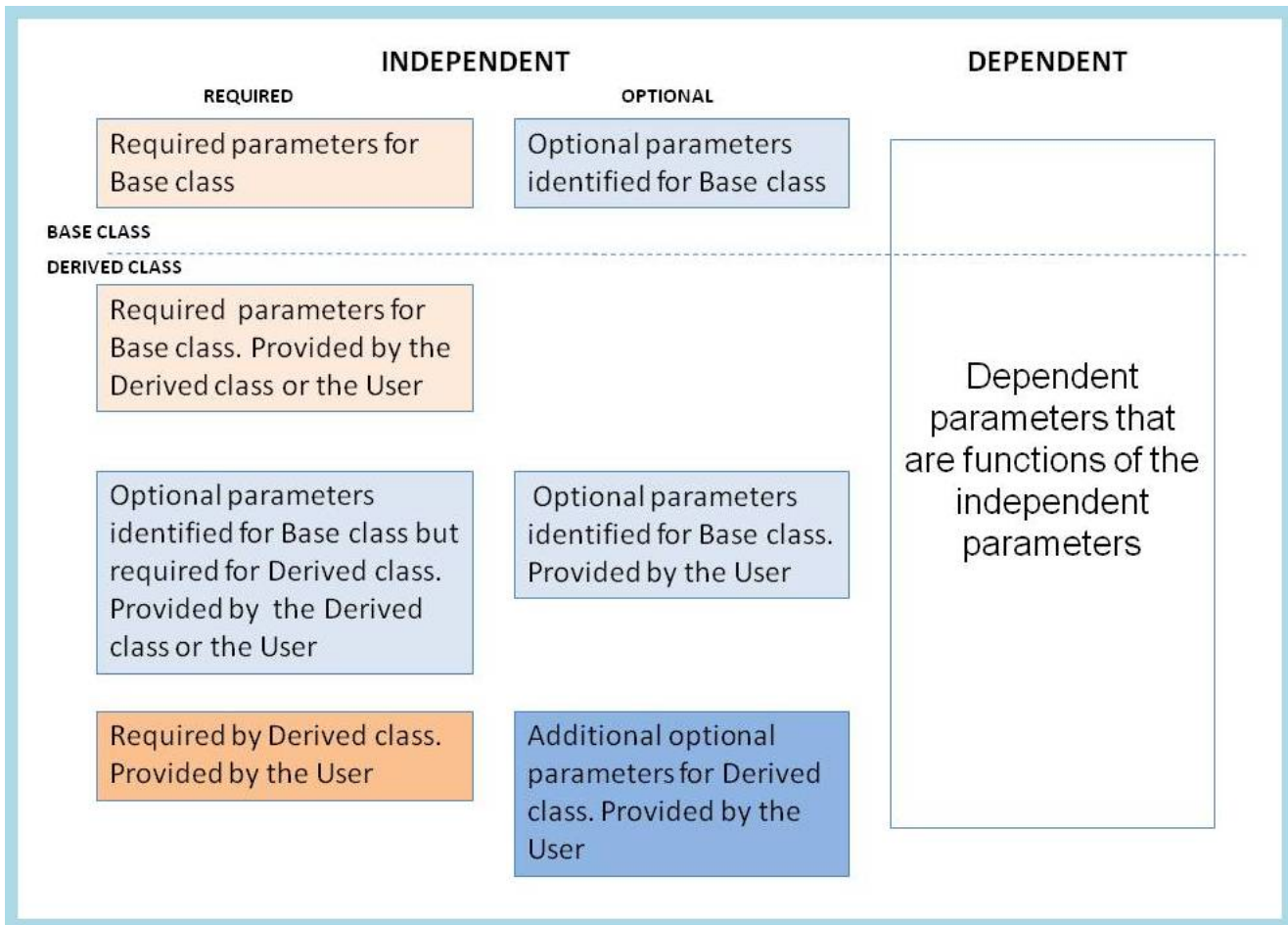


Figure 29. Parameter Class Hierarchy

- The parameter class must support units by:
  - Providing for the user to specify the units for each parameter field

- Converting the user-provided values into the unit specified in the parameter class definition.
- Provide a description of each parameter field in the parameter class definition.
- Provide a capability for the user to provide information on the source of:
  - each parameter field
  - the parameter class instance as a whole
- Provide python methods to support ancillary functionality associated with the parameter classes.
- Utilize the underlying Dshell provided Dvar C++ class capabilities to implement parameter storage and access.
- Provide to users, where possible, a python dictionary-style access to the parameter fields.

#### 1.6.5.4. Parameter Classes

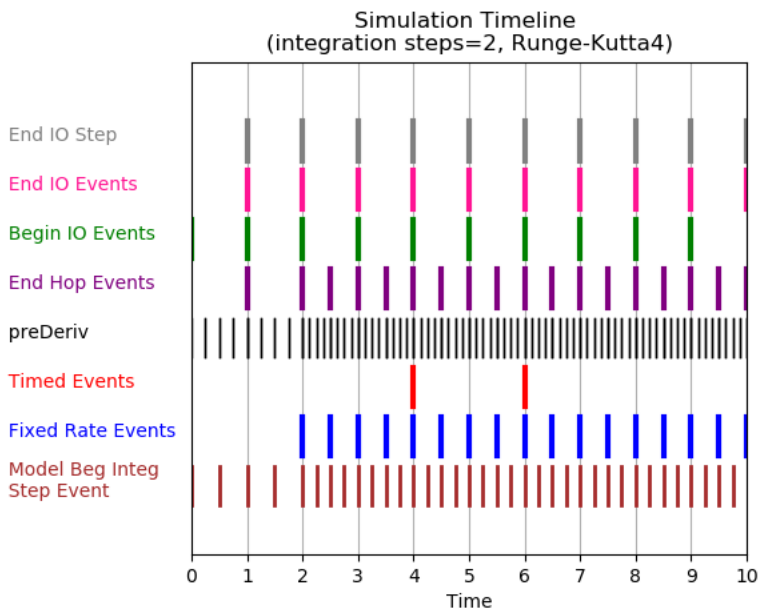
- `DshellCommon Parameter Reference<DshellCommonParamReference>`
- `DshellCommon base parameter class (Baseparam)<DshellCommonBaseParam>`

#### 1.6.6. Dshell++ Events

Events are an important part of the DARTS Dshell framework which enable many capabilities. DARTS/Dshell supports the following types of events (which will be described following sections):

- Events that occur at a regular or non-periodic intervals
- Events that occur at every system-wide BeginIOStep or EndIOStep
- Events that occur at every endHop
- Events that occur with specific model functions
- Step Validation Events / "Zero-Crossing" events

Here is a sample timeline of example events occurring during a simulation:



This timeline of events was created by this regtest:

- Combined events regtest

#### 1.6.6.1. Basic Event C++ API

Each event has the following primary features:

- A callback function that the event executes
- A trigger that defines when the callback function is executed

The C++ API for registering events is similar in all cases and follows the form:

```
void register<type>Event(conststd::string &name,
                        EventFunction* f,
                        EventCondition* condition,
                        void* cookie,
                        ...
                        <more optional arguments>);
```

where `<type>` corresponds to the type of event being registered and **name** is the name of the event (for various event operations); the event name must be unique. There may be some variation of these arguments for registering some types of events. The other arguments are as follows:

- **EventFunction** — The "callback" function that will be executed when the event is "fired". Its API is:

```
typedef void (EventFunction)(void *);
```

- **EventCondition** — The (optional) trigger function that must evaluate to true in order for the event callback function to be executed. Its API is:

```
typedef bool (EventCondition)(void *);
```

- **cookie** — The "client data" is passed to the callback function and event condition function.

Note that the API shown here is for the C++ implementation. The Python interfaces for these functions are very similar although there are some conveniences that the Python interface offers.



#### Note

The exact API for each type of event can be found in `Dshell++/Simulation.h`

#### 1.6.6.1.1. Event Utility Functions

There are also several other utility functions for dealing with events. To find an event by its string name:

```
CallbackEvent* findEvent(conststd::string &name);
```

To delete an event by its string name:

```
void deleteEvent(conststd::string &name);
```

To delete all events:

```
void clearEvents(void);
```

To get a list of all the names of registered events:

```
std::vector< std::vector<std::string> > listEvents(void);
```

#### 1.6.6.2. Time-based Events

These events occur at specific times in the simulation. These times can be at a specific time, some fixed rate (periodic), or at arbitrary intervals (self-rescheduling).

Here is an example using the Python API (this can also be done in C++):

```
def callback():
    print "event hit!"

# A SINGLE event
sim.registerTimedEvent("ev", 3.0, callback)

# SEVERAL events
sim.registerTimedEvent("ev1", [1.0, 2.0, 3.0], callback)
```

PYTHON

The final line here creates several one-time events for each of the items in the list (via extensions to the interface in the python interface). Note that the names of the events are appended with the execution times so the event names are unique.

For examples of timed events, see these regtests in `Dshell++/test`:

- Timed events regtest
- Combined events regtest

It is also possible to create a timed event that reschedules itself in an arbitrary way. See regtest `timed events regtest` for an example of this.



**TBD:** Double check that the `Dshellpp_test_Events_timed_regtest_uri[regtest]` does reschedule a event (used to be `timed2 regtest`)

#### 1.6.6.3. IOStep or endHop Events

Events can be registered to occur immediately after regular system-wide IO steps (at `BeginIOStep` or `EndIOStep`). These are separate from the normal model C++ `beginIOStep0/endIOStep0` functions which get executed at IO Step boundaries.

Python API (can also be done in C++)

```
def f(t):
    # t is the sim.time function. Call t to get the time
    print "callback at time={}".format( t() )

sim.registerEndIOStepEvent('event', f, t = sim.time)
```

Note that `registerEndIOStepEvent` could have been `registerBeginIOStepEvent` or `registerEndHopEvent` to associate the event callback with those system-wide functions.

For examples of IOStep/endHop events, see these regtests in Dshell++/test:

- Basic events regtest
- Combined events regtest



TBD: Double check that the Dshellpp\_test\_Events\_basic\_regtest\_uri[regtest] does does have endHop events (used to be io regtest)

#### 1.6.6.4. Model Events

Model events allow adding event callbacks that execute when a specific model's C++ function (such as `preDeriv`) is called. This does not require recompiling model code.

The callback function AND trigger function can be C++ or python. Implementing these callbacks and triggers in Python is useful for prototyping functions. These callback can be set to execute before or after corresponding the C++ function.

```
def f(t):
    # t is the sim.time function. Call t to get the time
    print "callback at time={}".format( t() )

Sim.registerPreDerivModelEvent('pdev', mdl, True, f)
```

where:

- `mdl` is the model object
- `True` here indicates that the callback occurs before the model's `preDeriv()` function.
- `f` is the callback function

Note that 'PreDeriv' here can be replaced with a name of some other normal model function.

For examples of model events, see these regtests in Dshell++/test:

- {Dshellpp\_test\_Events\_model\_regtest\_uri}[Model events regtest]
- Combined events regtest
- `test_events/script_model_events.py`
- `test_Ndarts/test_event/test_combined_event_timeline/script.py`

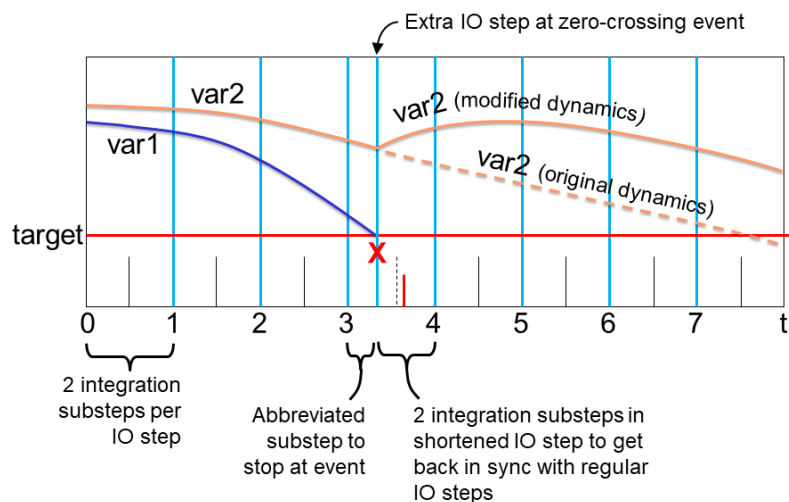


TBD: Double check that the Dshellpp\_test\_Events\_model\_regtest\_uri[regtest] exists

#### 1.6.6.5. Step Validation / Zero-Crossing Events

In realistic simulations, we often encounter situations where we need to stop simulation at the precise time that a condition is met (eg. something touching the ground). Step-Validation events allow carrying out of actions at the exact time when a condition is met whose timing cannot be predicted explicitly in advance.

Here is representation of what occurs in step validation events.



Here a step-validation event is used to determine exactly when specified condition is met, even if it is in the middle of an integration step. This is accomplished as follows:

- The user provides callback function that determines when the condition occurs even if it is in the middle of an integration step. The trigger function is usually when a DVar specNode ('var1' in the figure above) crosses some value ('target' in the figure above). If the condition occurs during in a substep, the substep is reverted (eg, the substep is NOT validated and is rejected, hence the name "Step Validation")
- When the trigger condition occurs, the simulation enters an iterative process to determine exactly when to end the hop (and interrupt the current IO step):
  - The current integration step is reverted.
  - The end of the next trial substep is set part way through the step.
  - A trial substep is taken: If the condition is still violated, the substep is reverted again and the trial time is reduced
  - This process iterates until condition is achieved (within epsilon). This terminates the hop at the zero-crossing time (effectively an implicit barrier is inserted).
  - Then events associated with the zero-crossing condition are handled, and a new hop is used to complete the interrupted hop.

A few notes on this process:

- We are only retrying substeps (not full IO steps).
- We do not touch I/O steps, FSM transitions or other data not captured in the full multibody or model state.
- If the trigger condition is met, the step is terminated early, and the next hop begins earlier than it otherwise would have. The end of the next hop syncs up with the normal I/O step cycle.
- It is possible that the user may have made qualitative changes to the system in the step-validation callback function. Therefore we need to reevaluate the full dynamics of the system at that point and start propagating again from that point. In the figure above, notice the changed behavior of 'var2' due to system changes during the step-validation event.
- By default, the iterative process of determining exactly when an event occurs uses a bisection algorithm but other types of iterative approaches can be implemented in C++.

The C++ API for Step-Validation Events is a little different:

```
void registerStepValidationEvent(const std::string &name,
                               const DVar::Leaf_T<double> &_x_var, # or vector
                               double _x_target,
                               bool _from_below,
                               double _eps = 1e-6,
                               EventFunction* f = NULL,
                               void* cookie = NULL);
```

where

- `_x_var` - is the DVar specNode to be monitored
- `_x_target` - the callback triggers when the value of `_x_var` reaches this value
- `_from_below` - the user indicates which direction the value of `_x_var` is approaching `_x_target`
- `_eps` - how accurately the value of `_x_var` needs to reach `_x_target` (the convergence routine iterates until this is reached)
- `f` - the event callback to be called when the event occurs
- `cookie` - data to be provided to the EventFunction `f` when it is invoked.

The Python API makes it a bit simpler to use. Note that it uses the bisection algorithm by default:

```
def cb(t):
    print "callback at time={}".format( t() )

sim.registerStepValidationEvent('zcev', spec, 0.0,
                               False, f=cb, t=sim.time)
```

This Step Validation event will trigger when the value of the DVar specNode 'spec' reaches '0.0'. Notice that the Python interface processes any extra arguments (such as 't' here) and bundles them into the 'cookie'. These arguments are eventually passed to the event function as a series of extra arguments. So the function `cb(t)` receives the callable function 't' when it is called.

For examples of step validation events, see these regtests in Dshell++/test:

- Step validation events regtest
- [test\\_step\\_validation/test\\_integration\\_rollback/script.py](#)



The C++ API does support step validation events using an arbitrary trigger function. Please see these files for details:

- Dshell++/Simulation.h
- Dshell++/CallbackStepValidation.h

### 1.6.7. MultiRate EndIOStep Execution

The DARTS Dshell framework supports having models whose `EndIOStep()` functions are called at arbitrary rates. There is easy-to-use support for fixed rate multirate models. There is also support for user-defined arbitrary multirate execution of a model's `EndIOStep` function.

Multi-rate schedule for a model can be done by two approaches using methods in `Dshell::Simulation`:

- Fixed Rate



```
void Simulation::registerMultiRateModel(Model &model, double fixed_rate_hz);
```

- User-provided callback function

```
void Simulation::registerMultiRateModel(Model &model, multirate_callback_t cb);
```

Both can be done externally (in a run script) or in a model's C++ setup() function.

All models can be configured to be multirate models (including flow models). This does not have to be done in the model's C++ code.



#### Note

The usual functions of a multirate model (such as StartIntegrationStep, preDeriv, postDeriv, etc) operate along with all models during ALL integration steps. What distinguishes multirate models is that their `EndIOStep()` function is called at the times their multirate registration specifies — and ONLY at those times — Specifically, not at the normal system-wide end of IO steps (unless the rate specified by the registration happens to coincide with the system-wide IO step times).

Therefore, if you want operations to happen at the times specified when the model is registered as a multirate model, the operations must be called in the models (overridden) `EndIOStep()` function.

### 1.6.7.1. Fixed Rate

Once the model is registered using the fixed rate API, the specified model will be called at the fixed rate specified and NOT at the usual simulation-wide IO-step schedule (unless the specified rate is commensurable with it). Note that the fixed rate does not need to be related to the current IO Step size. No further actions are needed to enable fixed-rate multirate support.

### 1.6.7.2. User-provided callback function

If more flexibility is needed, the model time trigger points can be arbitrarily specified by calling the second version of `registerMultiRateModel()`, and providing a callback function (of type `multirate_callback_t`). This callback function returns the time trigger points.

For example, a the multirate callback function for a mode can be an override of model base class function `getNextMultiRateIOTime()` or an arbitrary function can be defined with an API like this:

```
MultirateStatus nextStepTime(Time::TimeSpec &desired_time);
```

then this function can registered using the simulation's function

```
sim.registerMultiRateModel(<model>, nextStepTime);
```

The callback function must set the desired time for the next IO step callback for the specific model `<model>` and return the appropriate **MultirateStatus** (see `Dshell++/Model.h` for details):

- NOT\_MULTIRATE
- MULTIRATE\_HAVE\_FUTURE\_TRIGGER
- MULTIRATE\_NO\_FUTURE\_TRIGGER

Note that registering a model as a **Fixed Rate** multirate model simply uses the rate provided and registers the Model base class function `Model::getNextMultiRateIOTime()` to handle the fixed rate callbacks.

For examples of how to set up more complicated `multirate_callback_t` functions, please see the C++ files in:

- `Dshell++/test/test_multirate/models/`

### 1.6.7.3. Multi-rate timing

A note on overhead for multirate support: Our tests indicate that using the multirate functionality adds less than 1% overhead.

## 1.7. Regression Tests

### 1.7.1. Dshell++ Regression Tests

#### 1.7.1.1. A few useful regression tests for reference

| Link to regression test            | Description  |
|------------------------------------|--|
| <a href="#">test_Assembly</a>      | Simple assembly tests  |
| <a href="#">test_Model</a>         | Simple model tests   |
| <a href="#">test_Model2</a>        | Use signal splitter/joiner models and verify signal connectivity |
| <a href="#">test_Model7</a>        | Test model sorting constraints                                   |
| <a href="#">test_Model8</a>        | Using model meta data without creating a model instance          |
| <a href="#">test_Signals</a>       | Demonstrate basic signal operations                              |
| <a href="#">test_Events_io</a>     | Demonstrate using IO events                                      |
| <a href="#">test_Events_model</a>  | Demonstrate using model function events                          |
| <a href="#">test_Events_timed2</a> | Demonstrate using timed events (including self-rescheduling)     |

| Link to regression test     | Description   |
|-----------------------------|---|
| test_Events_timed           | Demonstrate using timed events                      |
| test_Events_add_remove_find | Demonstrate adding, removing, and finding events    |
| test_Events_combined        | Plot a timeline of various combined events          |
| test_Events_step_validation | Basic step validation / zero-crossing event example |

## 2. DshellCommon

### 2.1. Background

#### 2.1.1. Reference & Source material

- [DshellCommon Doxygen docs](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/DshellCommon/html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/DshellCommon/html)
- [Team talks](https://dlabportal.jpl.nasa.gov/resources/darts-lab-talks/) (https://dlabportal.jpl.nasa.gov/resources/darts-lab-talks/)
- [2019, 2020 course slides](https://dlabportal.jpl.nasa.gov/documentation/coursetutorial-links/) (https://dlabportal.jpl.nasa.gov/documentation/coursetutorial-links/)
- [Sphinx docs](https://dartslab.jpl.nasa.gov/dlabdocs/) (https://dartslab.jpl.nasa.gov/dlabdocs/)
- [Jupyter notebooks](https://dlabnotebooks.jpl.nasa.gov/hub/login) (https://dlabnotebooks.jpl.nasa.gov/hub/login)
- [DARTS Lab QA site](https://dartslab.jpl.nasa.gov/qa/) (https://dartslab.jpl.nasa.gov/qa/)

### 2.2. Design

#### 2.2.1. InputDict: replacement for anonymous dictionaries

InputDict are utility classes that allow us to organize data in a single object. They are similar to python built-in dataclasses and C++ struct. They can act as substitutes for python dictionaries when we know what fields to expect in the dictionary.

##### 2.2.1.1. Motivation

Let's consider we have a function `print_house(...)` that takes a python dictionary (`info`), which contains necessary information for the function to work:

```
def print_house(info):  
    name = info["name"]  
    homeworld = info["homeworld"]  
    major = info.get("major", False)  
    print(f"{'Great' if major else ''} House {name} of {homeworld}")
```

PYTHON

By inspection of the source of the function, we see that `info` must have the fields `"name"` and `"homeworld"`. Additionally, the field `"major"` can optionally be given. Moreover, we could predict that `"name"` and `"homeworld"` must be strings, while `"major"` must be a boolean. Users would need to know this information to properly use `print_house(...)`, and to do so, they must also inspect the source code of the function.

While reading the code is simple in this example, this might not always be the case. Having to find and read the source of a function before using it is a tedious and error-prone process, especially when the functions span many lines. Ideally, we would like to be able to fully understand how to use a function from its signature and docstring.

##### Simple use case

We can use an InputDict to solve this problem without even having to change the source code of the function:

```
from DshellCommon.input_dict.input_dict import InputDict  
  
class HouseInformation(InputDict):  
    name: str  
    homeworld: str  
    major: bool = False  
  
def print_house(info: HouseInformation):  
    name = info["name"]  
    homeworld = info["homeworld"]  
    major = info.get("major", False)  
    print(f"{'Great' if major else ''} House {name} of {homeworld}")
```

PYTHON

As we can see, we created a new class `HouseInformation` that inherits from `InputDict`. We declared three fields, `name`, `homeworld`, and `major`, and their types: `str`, `str`, and `bool`. Moreover, we provided a default value for `major` (`False`). Finally, we added a type hint to the signature of `print_house(...)` declaring that the input `info` should be an instance of `HouseInformation`.

Now, when users want to use `print_house(...)`, they only need to inspect the signature and discover that the input should be a `HouseInformation`. Then, they can inspect `HouseInformation` and easily see a list of the appropriate fields and their types.

Given that the source of the function has not changed, both a dictionary and `HouseInformation` are valid and equivalent inputs to the function:

```
print_house({  
    "name": "Atreides",  
    "homeworld": "Caladan",  
    "major": True,  
})  
# Great House Atreides of Caladan  
  
print_house(HouseInformation(  
    name = "Atreides",  
    homeworld = "Caladan",  
    major = True,  
))  
# Great House Atreides of Caladan
```

PYTHON

##### 2.2.1.2. Advantages of InputDict over anonymous dictionaries

- As explained in the previous section, using `InputDict` allows users to quickly understand what information is required by a piece of code (a function, class...). This means faster development with errors.
- Using a modern code editor (such as VSCode with Python intellisense) will show all necessary information as pop-up windows, thus removing the need to navigate to the source code of the class: `\[INSERT IMAGE basic_popup]`
- Activating syntax highlighting for incorrect types in your code editor will allow you to catch errors before even running the code: `\[INSERT IMAGES error_highlight]`
- At runtime, validation will be performed on the inputs. This allows you to contain all input validation on the `InputDict` and thus write a clearer, more concise function. Input validation consists on:
  - checking the types of the inputs. For example, setting `"homeworld"` to be an integer will raise an exception.
  - any other more complex and arbitrary validation. For instance, we might create an `InputDict` that checks that the first letter of the input `"homeworld"` is in upper case.
  - More info on Validation in section Validation.
- `InputDict` subclasses might be created from a python dictionary or text files using `parse` and `parse_file`. See Parsing functions for details.
- `InputDict` can easily be transformed into options for commands in the CLI. See DictLinker for more details.
- `InputDict` subclasses can be autogenerated from Models. See ModelLinker
- `InputDict` can act as elegant wrappers for DVar Branches. See DVarInputDict.

## How to use InputDict

In the Simple use case section, we saw how one might implement a simple `InputDict`. In this section, we are going to expand on this and explore all possibilities for `InputDict`.

### Accessing the fields of an InputDict

In the previous section, we saw that we could use array-like syntax to access the values stored in an `InputDict`:

```
info = HouseInformation(name="Atreides", homeworld="Caladan")
assert info["homeworld"] == "Caladan"
info["homeworld"] = "Arrakis"
```

PYTHON

However, attribute-like access is also supported and acts exactly the same as the array-like operations:

```
info = HouseInformation(name="Atreides", homeworld="Caladan")
assert info.homeworld == "Caladan"
info.homeworld = "Arrakis"
```

PYTHON

New code should always aim to use attribute-like access (`info.homeworld`), as this preserves type hints. Array-like access is supported mainly to allow using `InputDict` with legacy code that was written with python dictionaries in mind and thus uses array-like access (`info["homeworld"]`).

### Dict-like behaviour

`InputDict` inherits from the python abstract class `MutableMapping`. This means that not only we can get and set fields, but also use other operators and methods such as `keys`, `items`, `values`, `get`, `__iter__` ...

The `__del__` method is also supported, but its usage is heavily discouraged in new code. Deleting a field in an `InputDict` will make it so trying to access the value raises a `KeyError`; similar to regular dictionaries. For new code, the recommended approach is to use `None` and declare the type as `Optional`. This explicitly signals to users that the field might become a `None`, while deleting a field can be done silently and thus cause problems on the long term:

Usage of `__del__` for dict-like compatibility:

```
class HouseInformation(InputDict):
    homeworld: str

info = HouseInformation(homeworld="Caladan")
del info["homeworld"]
assert "homeworld" not in info
assert "homeworld" not in info.keys()
info["homeworld"] # <- raises a KeyError
```

PYTHON

Recommended approach for new code:

```
from Dutils.typing import Optional

class HouseInformation(InputDict):
    homeworld: Optional[str]

info = HouseInformation(homeworld="Caladan")
info["homeworld"] = None
assert "homeworld" in info # Note that the `in` operator returns True now!
assert info["homeworld"] is None
```

PYTHON

### Required and optional fields

In Simple use case we saw examples of required fields and a field with a default value:

```
class HouseInformation(InputDict):
    name: str
    homeworld: str
    major: bool = False
```

PYTHON

To specify optional fields with no default we need to use the function `Dfield`, which allows us to provide additional information for each of the fields of an `InputDict`:

```
from DshellCommon.input_dict.input_dict import Dfield

class HouseInformation(InputDict):
    name: str
    homeworld: str
    major: bool = False
    title: str = Dfield(required=False)
```

PYTHON

When we create an `InputDict` we must always provide values for required fields. We might also provide values for optional fields. If we do not provide values for these, defaults will be used. If the field is optional without default, then the field will not contain any value; similarly to calling `__del__` on the field, accessing the attribute will raise a `KeyError`:

```
info = HouseInformation(name="Atreides", homeworld="Caladan")

assert "title" not in info
assert "title" not in info.keys()
info["title"] # <- raises a KeyError
info.title    # <- raises a KeyError

info.title = "Duke"
assert "title" in info
info.title    # <- does not raise a KeyError
```

PYTHON

Much like how the usage of `__del__` was discouraged for new code, we also discourage using optional fields without default. To obtain similar but more explicit behaviour (i.e. safer), use `Optional` types and `None` as the default:

```
from Dutils.typing import Optional

class HouseInformation(InputDict):
    name: str
    homeworld: str
    major: bool = False
    title: Optional[str] = None
```

PYTHON

The `Dfield` function can also be used to provide a default value by specifying `default`. Alternatively, a `default_factory` can be provided. This must be a callable object (function) that takes no arguments and returns a default value for the field:

```
from DshellCommon.input_dict.input_dict import Dfield

default_world = "Terra"

class HouseInformation(InputDict):
    homeworld: str = Dfield(default_factory= lambda: default_world)
    major: bool = Dfield(default=False)

print(HouseInformation())
# HouseInformation(homeworld='Terra', major=False)

default_world = "Ceres"
print(HouseInformation())
# HouseInformation(homeworld='Ceres', major=False)
```

PYTHON

Note that default values are always deepcopied before being assigned. This means that having mutable types (lists, dictionaries) as default is valid and each instance will have their own copy.

#### Description of each field

For documentation purposes, it's useful to provide descriptions for each of the fields of an `InputDict`. Descriptions are not only useful as documentation for the users, but they might also be used by other systems that use `InputDict` (for example, autogenerated CLI options will use the description as the help message for the option). There are two ways to set the description for a field: through the use of `Dfield(..., description=...)` or by writing "docstrings" in the numpy format. These two snippets are equivalent:

#### Using `Dfield`

```
class HouseInformation(InputDict):
    name: str = Dfield(description="The common name of the House")
    homeworld: str = Dfield(description="The planet seat of power of the House")
    major: bool = Dfield(default=False, description="Whether this is a House Major")
```

PYTHON

using the "docstring"

```
class HouseInformation(InputDict):
    """Contains information about a Dune House.

    Parameters
    -----
    name : str
        The common name of the House
    homeworld : str
        The planet seat of power of the House
    major : bool
        Whether this is a House Major
    """
    name: str
    homeworld: str
    major: bool = False
```

While the first option is more compact, the second option is easier to read and allows IDEs to display the information on hover: \[INSERT IMAGE docstring]

### 2.2.1.3. Inheritance

`InputDict` can inherit from one or more `InputDict`. The resultant class will have all the fields of the parent `InputDict`, plus the ones defined in the new class body. Moreover, fields can be specified again to override their definition.

```
class HouseInformation(InputDict):
    name: str
    homeworld: str
    major: bool = False

class LegionCommander(InputDict):
    number_of_legions: int

class MajorHouseInformation(HouseInformation, LegionCommander):
    major: bool = True
    vassal_house: str

import pprint
pprint.pprint(MajorHouseInformation(
    name="Harkonnen",
    homeworld="Giedi Prime",
    number_of_legions=10,
    vassal_house="Rabban",
), sort_dicts=False)
# MajorHouseInformation({'number_of_legions': 10,
# 'name': 'Harkonnen',
# 'homeworld': 'Giedi Prime',
# 'major': True,
# 'vassal_house': 'Rabban'})
```

Note that fields are "inherited" following python's class inheritance scheme. In this case, `LegionCommander` fields are set first, then `HouseInformation` fields, and finally those defined in `MajorHouseInformation`. Fields that are set later override those defined before.

### 2.2.1.4. Validation

Input validation is one of the core features of `InputDict`. Any `InputDict` class will use the types given in the class definition to check whether inputs are of the correct type:

```
class HouseInformation(InputDict):
    name: str
    major: bool

HouseInformation(name=42, major=True)
# raises a TypeError because 42 is not a string

HouseInformation(name="Corrino", major="test")
# raises a TypeError because "test" is not a boolean
```

#### 2.2.1.4.1. Advanced types

In addition to regular types, the greater part of the python's "typing" module is supported. Users are encouraged to read the python documentation on [type hinting](https://docs.python.org/3/library/typing.html) (<https://docs.python.org/3/library/typing.html>).

Users are also **strongly** encouraged to read the [DARTS-specific typing documentation](https://dartsgitlab-internal.jpl.nasa.gov/darts-lab/jpl-internal/all-access/development/core-modules/dshellenv/-/wikis/DshellEnv-documentation#user-content-typing)

(<https://dartsgitlab-internal.jpl.nasa.gov/darts-lab/jpl-internal/all-access/development/core-modules/dshellenv/-/wikis/DshellEnv-documentation#user-content-typing>) (in `DshellEnv`), which covers both python typing and the special extensions to typing done in DARTS.

The `Dutils.typing` documentation shows an example of a function with several type annotations, which is used to illustrate the range of possibilities of typing. The following snippet shows an `InputDict` with the same type annotations and demonstrates that validation with these types can also be performed in `InputDict`:

```

from Dutils.typing import (
    Optional,
    Union,
    Literal,
    List,
    Tuple,
    Set,
    Dict,
    Sequence,
    Iterable,
    Mapping,
    Any,
)

class MyType:
    ...

ListOfAgencies = List[Literal["NASA", "ESA", "JAXA", "Roscosmos"]]

class TypesTest(InputDict):
    a_type: MyType
    b_union: Union[str, bool]
    c_optional: Optional[str]
    d_literal: Literal["foo", True, 42]
    e_list: List[str]
    f_tuple: Tuple[int, ...]
    g_tuple_fixed: Tuple[int, str, float]
    h_set: Set[str]
    i_dict: Dict[str, int]
    j_sequence: Sequence[float]
    k_iterable: Iterable[int]
    l_mapping: Mapping[str, int]
    m_combinations: Union[List[Union[str, float]], Optional[Set[int]]]
    n_alias: ListOfAgencies
    o_any: Any

```

#### 2.2.1.4.2. Annotated types

As explained in the [DARTS-specific typing documentation](https://dartsgitlab-internal.jp1.nasa.gov/darts-lab/jpl-internal/all-access/development/core-modules/dshellenv/-wikis/DshellEnv-documentation#user-content-typing)

(<https://dartsgitlab-internal.jp1.nasa.gov/darts-lab/jpl-internal/all-access/development/core-modules/dshellenv/-wikis/DshellEnv-documentation#user-content-typing>) (in DshellEnv), we use `Annotated` and `FieldCheck` to add additional constraints to certain types. This extended types can also be used in `InputDict` :

```

from DshellCommon.input_dict.input_dict import InputDict
from Dutils.typing.field_check import FieldCheck
from Dutils.typing import Annotated

class PositiveCheck(FieldCheck):

    def check(self, value):
        return 0 < value

PositiveInt = Annotated[int, PositiveCheck()]

class HouseInformation(InputDict):
    number_of_legions: PositiveInt

HouseInformation(number_of_legions = 1000) # ok

HouseInformation(number_of_legions = -1)
# TypeError: Value -1 failed check PositiveCheck

```

The `Dutils.typing` module is filled with built-in `FieldCheck` classes and type alias that are useful for `InputDict` classes. Reading the docs for this module is (again) encouraged ([link](#)).

#### 2.2.1.4.3. Other validation with FieldCheck

While using `Annotated` with `FieldCheck` allows you to perform any validation possible on the value, sometimes we do not want to "contaminate" the type declaration with the checks. To add checks to a field without using its type we can use the `validators` keyword argument in `Dfield` :

```

from DshellCommon.input_dict.input_dict import InputDict
from Dutils.typing.field_check import FieldCheck

class PositiveCheck(FieldCheck):

    def check(self, value):
        return 0 < value

class LegionCommander(InputDict):
    number_of_legions: int = Dfield(validators=[PositiveCheck()])

LegionCommander(number_of_legions = 42) # ok

LegionCommander(number_of_legions = -1)
# RuntimeError: number_of_legions failed field check: Value -1 failed check PositiveCheck

```

In the previous snippet, we are specifying the parameter `validators` to be a list that contains an instance of this `PositiveCheck` class. Even though the input `-1` met the type requirement (it is an integer), it failed the validator `PositiveCheck()` .

Type checking is always performed before validation, so you may assume that the type is "correct" in your validator function definition.

#### 2.2.1.4.4. Unsafe functionality

Validation (and type checking) can be disabled. This is discouraged, as most of the time one can better define the type of a field or its validation function instead of simply turning validation off. However, for completion purposes, the different ways of disabling validation are explained here.

##### 2.2.1.4.4.1. Class-wide

One can disable validation and type checking for all instances of a class and their sub-classes by providing the keyword arguments `type_check` and `validate` on the class signature:

```
from DshellCommon.input_dict.input_dict import InputDict, Dfield, FieldCheck

class PositiveCheck(FieldCheck):
    def check(self, value):
        return 0 < value

class Unsafe(InputDict, type_check="never", validate="never"):
    my_unsafe_int: int = Dfield(validators=[PositiveCheck()])

my_unsafe = Unsafe(my_unsafe_int = "test")
my_unsafe = Unsafe(my_unsafe_int = -42)
my_unsafe.my_unsafe_int = "test"
my_unsafe.my_unsafe_int = -42

class Unsafe(InputDict, type_check="on_init", validate="on_init"):
    my_unsafe_int: int = Dfield(validators=[PositiveCheck()])

my_unsafe = Unsafe(my_unsafe_int = 1)
my_unsafe.my_unsafe_int = "test"
my_unsafe.my_unsafe_int = -42
```

PYTHON

As we can see, setting `type_check` or `validate` to "never" will allow you to always bypass these checks both on initialization and when setting attributes. If `type_check` or `validate` are set to "on\_init", however, then the checks are only performed when the class is constructed, but not when setting attributes. By default, `type_check` or `validate` are "always".

##### 2.2.1.4.4.2. Field-specific

If you want to disable type checking for a single field in a class, add `type_check=False` to `Dfield`:

```
from DshellCommon.input_dict.input_dict import InputDict, Dfield

class Safe(InputDict):
    my_unsafe_int: int = Dfield(type_check=False)

Safe(my_unsafe_int="test")
```

PYTHON

If you want to disable validation for a field, then simply avoid providing validators for that field.

##### 2.2.1.4.4.3. Instance-specific

If you want to turn off type checks/validation only for one instance, then you can use the `unsafe_init` and `unsafe_set_field` to temporarily bypass checks:

```
from DshellCommon.input_dict.input_dict import InputDict, Dfield, FieldCheck

class PositiveCheck(FieldCheck):
    def check(self, value):
        return 0 < value

class Safe(InputDict):
    my_safe_int: int = Dfield(validators=[PositiveCheck()])

Safe.unsafe_init(bypass_type_check=True, bypass_validate=True, my_safe_int="test")
Safe.unsafe_init(bypass_validate=True, my_safe_int=-4)

my_safe = Safe(my_safe_int=32)
my_safe.unsafe_set_field("my_safe_int", "test", bypass_type_check=True, bypass_validate=True)
my_safe.unsafe_set_field("my_safe_int", -3, bypass_validate=True)
```

PYTHON

#### 2.2.1.5. Parsing functions

All `InputDict` subclasses provide two convenience functions to parse raw data into an `InputDict`; `parse` and `parse_file`. These parser functions will attempt to convert input data into an `InputDict` object. `parse` will take dictionary whose keys correspond to the fields of the `InputDict`, or it will take keyword arguments where the keywords are the fields of the `InputDict`:



```

from DshellCommon.input_dict.input_dict import InputDict
from Dutils.typing import List
from Dutils.typing import Annotated
from Dutils.typing.Dtyping import PositiveInt
from Dutils.typing.Dtyping import Time
from Dutils.typing.Dtyping import LengthArray
from Dutils.typing.field_check import SizeCheck

class HouseInformation(InputDict):
    major: bool
    members: List[str]
    choam_shares: PositiveInt
    years_since_formation: Time
    homeworld_position: Annotated[LengthArray, SizeCheck(3)]

unparsed_data = {
    "major": 1,
    "members": ("Feyd", "Vladimir"),
    "choam_shares": 143.7,
    "years_since_formation": "1000 years",
    "homeworld_position": ["1", "2", "3km"]
}

import pprint
pprint.pprint(HouseInformation.parse(unparsed_data), sort_dicts=False)
# HouseInformation({'major': True,
# 'members': ['Feyd', 'Vladimir'],
# 'choam_shares': 143,
# 'years_since_formation': array(1000.) * yr,
# 'homeworld_position': array([1., 2., 3.] * km)})

```

There are two things to mention about the parsing function:

- The parsing function will silently attempt to transform the data to fit the given types. The previous example shows three examples of this:
  - The "major" entry was transformed from an integer `1` to a boolean `True`.
  - The "members" entry was transformed from a tuple to a list.
  - The "choam\_shares" was transformed from a float `143.7` to an integer `143` by ignoring the decimals: **some information loss is possible**.
- Strings can be converted to numerics. Moreover, entries with types that accept units will parse the units from the strings:
  - For single values with units, the notation "[NUMERIC] [UNIT]" can be used.
  - For arrays with units, the unit must given in the **last** entry of the array. These are possible notations: "[NUMERIC\_1], "[NUMERIC\_2][UNIT]", ("NUMERIC\_1", "[NUMERIC\_2][UNIT]")

The function `parse_file` acts similarly to `parse`. Instead of taking a dictionary or the values, however, it takes the path to a file that contains the data. Internally, it works by loading the file's contents to a dictionary and then using the `parse` function, so both functions should act equivalently:

```

# unparsed_harkonen_file.yaml
major: 1
members:
  - Feyd
  - Vladimir
choam_shares: 143.7
years_since_formation: 1000 years
homeworld_position: [1, 2, 3km]

```

```

import pprint
pprint.pprint(HouseInformation.parse_file("unparsed_harkonen_file.yaml"), sort_dicts=False)
# HouseInformation({'major': True,
# 'members': ['Feyd', 'Vladimir'],
# 'choam_shares': 143,
# 'years_since_formation': array(1000.) * yr,
# 'homeworld_position': array([1., 2., 3.] * km)})

```

### 2.2.1.6. DictLinker

DictLinker, similarly to the older ParamLinker, is capable of autogenerating options for `Dclick` commands and later building `InputDict` from user inputs on the Command Line Interface or through config files.

Note that the generated command will have a rich help message, so you can always do:

```
srun my_command --help
```

to get an idea on how to use the command.

#### 2.2.1.6.1. Single linker

We can create the options for a `Dclick` command by calling the `.cli` method on an `InputDict` class. Then, within the command function definition, we must call `Dclick.add_options(COMMAND_NAME, Dclick.cli.commands[COMMAND_NAME], kwargs)`. This will store the values (`kwargs`) passed to the command `COMMAND_NAME` at the options dictionary under the key `COMMAND_NAME`. Finally, we can construct the `InputDict` with the class method `from_linker`. This is best illustrated with an example:

```

from DshellCommon.input_dict.input_dict import InputDict
from Dutils.typing import Annotated
from Dutils.typing.Dtyping import Velocity
from Dutils.typing.Dtyping import LengthArray
from Dutils.typing.field_check import SizeCheck

import Dutils.Dclick as Dclick
import click

class Ornithopter(InputDict):
    max_speed: Velocity
    destination: Annotated[LengthArray, SizeCheck(3)]

class Pilot(InputDict):
    age: int

@Dclick.cli.command()
@click.pass_context
@Ornithopter.cli
@Pilot.cli
def ornithopter(ctx, **kwargs):
    """
    Options to pass to a flying ornithopter assembly.
    """
    Dclick.add_options("ornithopter", Dclick.cli.commands["ornithopter"], kwargs)

# This imitates inputting a CLI command in the shape:
# srun ornithopter --max-speed 120m/s --destination 1 2 3km --age 32
cfgobj, ctxobj = Dclick.cli([
    "ornithopter",
    "--max-speed", "120m/s",
    "--destination", "1", "2", "3km",
    "--age", "32",
], standalone_mode=False)

# Usually, one does not input the command directly, so the following is used:
# cfgobj, ctxobj = Dclick.cli(standalone_mode=False)

ornithopter_input_dict = Ornithopter.from_linker(cfgobj["ornithopter"])
pilot_input_dict = Pilot.from_linker(cfgobj["ornithopter"])

import pprint
pprint.pprint(ornithopter_input_dict, sort_dicts = False)
# Ornithopter(max_speed=array(120.) * m/s, destination=array([1., 2., 3.] * km)
pprint.pprint(pilot_input_dict, sort_dicts = False)
# Pilot(age=32)

```

### 2.2.1.6.2. Customizing the behaviour of DictLinker

DictLinker will automatically make several "decisions" when transforming an `InputDict` to click options. Most of these decisions can be overridden or otherwise altered by passing arguments to the `cli` method:

- The `prefix` input allows you to add a prefix to all options for the `InputDict`
- The `exclude` input is a list of strings that will tell DictLinker to ignore creating options for certain fields. Note that if these fields are required, we will have to input them manually. This is discussed later in this section.
- If `remove_defaults` is `True`, then the `InputDict` defaults are not transformed to click option defaults.
- `cli_hints` is a dictionary that maps field names to objects of the type `CliHint` (from `"DshellCommon.input_dict.dict_linker"`). These allow you finer control over each option by using different keyword arguments (which you can see in `DshellCommon/python/input_dict/dict_linker.py`). Some useful keywords are:
  - `default` used to override the field default with another one.
  - `nargs` allows you to define how many arguments this option takes. This is useful for fields that do not prescribe a number of arguments themselves (`List[int]` takes any number of arguments, for example), as `click` needs to know the number of arguments beforehand.
  - `help` allows you to customize the help string of the option. By default, this is the description of the field in the `InputDict` (if available).

Finally, when we call `from_linker`, we can pass the `ext_fields` argument. This is a dictionary that allows you to manually add fields that were not specified through the CLI. This is useful for required fields that were not included as options but that must be provided for `InputDict` to be created.

Let's see an example:

```

from DshellCommon.input_dict.input_dict import InputDict
from Dutils.typing.Dtyping import Velocity
from Dutils.typing import List
from DshellCommon.input_dict.dict_linker import CliHint

import Dutils.Dclick as Dclick
import click

class Ornithopter(InputDict):
    max_speed: Velocity
    crashes: List[int]

class Pilot(InputDict):
    age: int
    name: str

orni_hints = {
    "max_speed": CliHint(default=100),
    "crashes": CliHint(nargs=2),
}

@Dclick.cli.command()
@click.pass_context
@Ornithopter.cli(cli_hints=orni_hints)
@Pilot.cli(prefix="pilot", exclude=["name"])
def ornithopter(ctx, **kwargs):
    """
    Options to pass to a flying ornithopter assembly.
    """
    Dclick.add_options("ornithopter", Dclick.cli.commands["ornithopter"], kwargs)

# This imitates inputting a CLI command in the shape:
# srun ornithopter --crashes 3054 3056 --pilot-age 32
cfgobj, ctxobj = Dclick.cli([
    "ornithopter",
    "--crashes", "3054", "3056",
    "--pilot-age", "32",
], standalone_mode=False)

# Usually, one does not input the command directly, so the following is used:
# cfgobj, ctxobj = Dclick.cli(standalone_mode=False)

external_fields_pilot = {"name": "Duncan"}

ornithoper_input_dict = Ornithopter.from_linker(cfgobj["ornithopter"])
pilot_input_dict = Pilot.from_linker(cfgobj["ornithopter"], ext_fields=external_fields_pilot)

import pprint
pprint.pprint(ornithoper_input_dict, sort_dicts = False)
# Ornithopter(max_speed=100.0, crashes=[3054, 3056])
pprint.pprint(pilot_input_dict, sort_dicts = False)
# Pilot(age=32, name='Duncan')

```

### 2.2.1.6.3. Using `.cli` multiple times

Sometimes we want to create multiple `InputDict` of the same class from different commands. This leads us to use `.cli` on different commands, possibly with different customizations (`prefix`, `cli_hints` ...). Because of this, when we use `from_linker`, the method does not know which customization options should be used. To solve this, we need to pass the command from which to we want to create the `InputDict` to `from_linker`:

```

from DshellCommon.input_dict.input_dict import InputDict
from Dutils.typing.Dtyping import Velocity
from DshellCommon.input_dict.dict_linker import CliHint

import Dutils.Dclick as Dclick
import click

class Ornithopter(InputDict):
    max_speed: Velocity

class Pilot(InputDict):
    age: int

orni_hints = {"max_speed": CliHint(default=100)}

@Dclick.cli.command()
@click.pass_context
@Ornithopter.cli(cli_hints=orni_hints)
@Pilot.cli(prefix="pilot")
def ornithopter(ctx, **kwargs):
    """
    Options to pass to a flying ornithopter assembly.
    """
    Dclick.add_options("ornithopter", Dclick.cli.commands["ornithopter"], kwargs)

@Dclick.cli.command()
@click.pass_context
@Ornithopter.cli
def elite(ctx, **kwargs):
    """
    Options to pass to an elite ornithopter assembly.
    """
    Dclick.add_options("elite", Dclick.cli.commands["elite"], kwargs)

# This imitates inputing a CLI command in the shape:
# srun ornithopter --pilot-age 32 elite --max-speed 150m/s
cfgobj, ctxobj = Dclick.cli([
    "ornithopter",
    "--pilot-age", "32",
    "elite",
    "--max-speed", "150m/s",
], standalone_mode=False)

# Usually, one does not input the command directly, so the following is used:
# cfgobj, ctxobj = Dclick.cli(standalone_mode=False)

ornithoper_input_dict = Ornithopter.from_linker(cfgobj["ornithopter"], command=Dclick.cli.commands["ornithopter"])
pilot_input_dict = Pilot.from_linker(cfgobj["ornithopter"])

elite_orni_input_dict = Ornithopter.from_linker(cfgobj["elite"], command=Dclick.cli.commands["elite"])

import pprint
pprint.pprint(ornithoper_input_dict, sort_dicts = False)
# Ornithopter(max_speed=100.0)
pprint.pprint(pilot_input_dict, sort_dicts = False)
# Pilot(age=32)
pprint.pprint(elite_orni_input_dict, sort_dicts = False)
# Ornithopter(max_speed=array(150.) * m/s)

```

In the previous example, we used `Ornithopter.cli` twice. For the `ornithopter` command we provided a default `max_speed`, while for the `elite` command we did not. Therefore, `max_speed` is a required option in the `elite` command, but it is not for `ornithopter`. When we build the `InputDict` with `from_linker`, we need to specify the command with `command=Dclick.cli.commands[COMMAND_NAME]`. Note that this was unnecessary for the `Pilot.from_linker` method, as only one `.cli` was ever called.

#### 2.2.1.6.4. Specifying the units of an argument

In the previous examples, we saw how we can specify the units for fields that accept units by adding the corresponding unit symbol in the CLI:

```
srun ornithopter --max-speed 120m/s --destination 1 2 3km
```

As we can see, for options that take multiple arguments, the units must be specified only on the last argument. Note that there cannot be any separation (spaces) between the numerical value and the symbol.

An alternative notation is possible:

```
srun ornithopter --max-speed 120 --units-max-speed m/s --destination 1 2 3 --units-destination km
```

The two commands above behave equivalently. These styles can be mixed, but not used at the same time for the same option (... `--max-speed 120m/s --units-max-speed km/s` is illegal, for example).

#### 2.2.1.6.5. Using configuration files

Configuration files can be used to simulate running commands on the CLI. This works as expected with `Dclick` options generated from `InputDict.cli`. While CLI will have option names in the format `--[NAME]-[NAME]`, the keys of a config file are in the format `[NAME]_[NAME]`. The config file and command:

```
# cfg.yml
ornithopter:
  max_speed: 120 m/s
  destination: [1, 2, 3]
  units_destination: km
  age: 32
```

YAML

```
srunch --in-cfg cfg.yml ornithopter
```

are equivalent to:

```
srunch ornithopter --max-speed 120m/s --destination 1 2 3 --units-destination km --age 32
```

### 2.2.1.7. ModelLinker

Models are defined through `.mdl` files and loaded into DARTs on initialization. These models are identified by a unique name in the format "[MODULE].[MODEL]", for example "DshellCommonModels.Accelerometer". Each model can define a series of parameters, with unique names, types, descriptions... Creating `InputDict` classes from these Model definitions is useful to expose these parameters to users or generate CLI commands.

When transforming Models to `InputDict`, we can use two functions defined in "DshellCommon.input\_dict.model\_linker": `model_linker_source` and `model_linker`. The first method will generate the source code of an `InputDict` equivalent to the model as a **string**. This means you can write this source on a `.py` file and it will be loaded as a simple `InputDict`. This method is therefore **static**: once you generate the source code and paste it in a file, changing the Model will not update the `InputDict` definition, as these are not linked.

The second method, on the other hand, returns the `InputDict` class directly. This class can be treated as any other type in python: one can construct instances from it, or use it as the parent class of other `InputDict`. Because this class is generated on-the-fly when `model_linker` is called, it will keep itself updated with changes in the models automatically.

Whether you use the source code or evaluated version of the function depends on your preferences. The source code version is clearer, as you can inspect the result and assert that the translation happened as desired. Moreover, users and IDE will be able to inspect the fields and their types, thus making its usage simpler and safer. However, it will not be updated with changes in `.mdl`.

The evaluated version is more opaque, but is the fastest to implement and will keep itself updated. For rapid prototyping of new models, the evaluated version is recommended, while it might be worth using the source code option for more settled models.

#### 2.2.1.7.1. model\_linker\_source

The following script will write the source code for an `InputDict` that represents the fields of "DshellCommonModels.Accelerometer" on a new python file:

```
from DshellCommon.input_dict.model_linker import model_linker_source
acc_class_source, _ = model_linker_source("DshellCommonModels.Accelerometer")
with open("my_accel.py", "w") as f: f.write(acc_class_source)
```

PYTHON

Once the `my_accel.py` file has been created, we can import `Accelerometer` (the generated class) from it. Then, we can use it as if it were any other `InputDict` class (because it is):

```
from my_accel import Accelerometer
accel = Accelerometer(vwn=2.3, vrw=1.2, ...)
class MyExtendedAccelerometer(Accelerometer):
    extra_field: int
extended_accel = MyExtendedAccelerometer(vwn=2.3, vrw=1.2, ..., extra_field=2)
```

PYTHON

#### 2.2.1.7.2. model\_linker

Its usage is very similar to `model_linker_source`, except that the return object is the class object directly:

```
Accelerometer = model_linker("DshellCommonModels.Accelerometer")
accel = Accelerometer(vwn=2.3, vrw=1.2, ...)
class MyExtendedAccelerometer(Accelerometer):
    extra_field: int
extended_accel = MyExtendedAccelerometer(vwn=2.3, vrw=1.2, ..., extra_field=2)
```

PYTHON

#### 2.2.1.7.3. Customizing the behaviour of DictLinker

Both `model_linker_source` and `model_linker` can be called with optional arguments to customize their behaviour:

- The `prefix` input allows you to add a prefix to all fields of the generated `InputDict`
- The `exclude` input is a list of strings that will tell ModelLinker to ignore creating fields for certain parameters of the Model.
- `mdl_param_hints` is a dictionary that maps param names to objects of the type `MdlParamHint` (from "DshellCommon.input\_dict.model\_linker"). These allow you finer control over each field by using different keyword arguments (which you can see in `DshellCommon/python/input_dict/model_linker.py`). Some useful keywords are:
  - `default` is used to override the param default with another one.
  - `quantity` to establish the quantity for a parameter (e.g. "Length").

- `description` to provide the description of the field, instead of obtaining it from the Model.

### 2.2.1.8. DVarInputDict

`DVarInputDict` is a subclass of `InputDict` that maintains a `DVar` branch harmonized with themselves. Changing fields of the `DVarInputDict` python object will cause changes in the underlying `DVar` branch. This makes handling `DVar` objects simpler, as users only need to worry about the python-side `DVarInputDict`, which acts very similarly to a regular `InputDict`. One can access the `DVar` branch with `.specNode()`:

```
from DshellCommon.input_dict.dvar_input_dict import DVarInputDict
from Dutils.typing import List
from DVar.DVar_Py import Container

# This is usually done by the SimulationExecutive, but it is done here
# manually for the purpose of testing and demonstration
dvc = Container()
DVarInputDict.register_dvar_container(dvc)

# Declare DVarInputDict exactly the same as an InputDict
class Foo(DVarInputDict):
    foo: int
    baz: List[int]
    bar: str = "test"

foo = Foo(foo=44, baz=[1, 2, 5])

foo.specNode().dumpValues()
# .dvarInputDictCommonBranch.Foo...
# {'bar': 'test', 'baz': [1, 2, 5], 'foo': 44}

foo.baz = [1, 9]
foo.specNode().dumpValues()
# .dvarInputDictCommonBranch.Foo...
# {'bar': 'test', 'baz': [1, 9], 'foo': 44}

# This is usually done by the SimulationExecutive, but it is done here
# manually for the purpose of testing and demonstration
DVarInputDict.close_class()
```

PYTHON

#### 2.2.1.8.1. Using the DVar branch

By default, all `DVar` branches associated with `DVarInputDict` objects are stored in the `DVarInputDict.common_branch` `DVar` branch. However, if one wants to place this branch at a different position, one has to use the `addSpecNodeTo` in `DVarInputDict`. If one wants to return the branch to the storage `DVarInputDict.common_branch`, then the `detachSpecNode` must be called. To check whether a branch is in use (i.e. it is no longer in `DVarInputDict.common_branch`) one can use `isSpecNodeInUse`.

#### 2.2.1.8.2. Copy on get attribute

When we retrieve a mutable attribute (lists, dictionaries...) from a `DVarInputDict`, this attribute is deep copied before being returned. This means that changes to the returned value will not affect the `DVarInputDict`. To update the `DVarInputDict`, one has to re-set the attribute. This is done so that users cannot modify values of the `DVarInputDict` without this object "noticing" and thus updating the `DVar` branch:

```
from DshellCommon.input_dict.dvar_input_dict import DVarInputDict
from Dutils.typing import List
from DVar.DVar_Py import Container

# This is usually done by the SimulationExecutive, but it is done here
# manually for the purpose of testing and demonstration
dvc = Container()
DVarInputDict.register_dvar_container(dvc)

# Declare DVarInputDict exactly the same as an InputDict
class Foo(DVarInputDict):
    foo: int
    baz: List[int]
    bar: str = "test"

foo = Foo(foo=44, baz=[1, 2, 5])

# Will not do anything to the DVarInputDict
foo.baz.append(999)

print(foo.baz) # [1, 2, 5]
foo.specNode().dumpValues()
# .dvarInputDictCommonBranch.Foo...
# {'bar': 'test', 'baz': [1, 2, 5], 'foo': 44}

my_baz = foo.baz
my_baz.append(999)
foo.baz = my_baz

print(foo.baz) # [1, 2, 5, 999]
foo.specNode().dumpValues()
# .dvarInputDictCommonBranch.Foo...
# {'bar': 'test', 'baz': [1, 2, 5, 999], 'foo': 44}

# This is usually done by the SimulationExecutive, but it is done here
# manually for the purpose of testing and demonstration
DVarInputDict.close_class()
```

PYTHON

#### 2.2.1.8.3. Fields with units

Values with units are transformed when they are passed to DVar objects. From the given unit, they are transformed to the system units. Moreover, the unit and quantity attributes are specified in the DVar leaf:

```
PYTHON
from DshellCommon.input_dict.dvar_input_dict import DVarInputDict
from Dutils.typing.Dtyping import Length
from Dutils.typing.Dtyping import LengthArray
from DVar.DVar_Py import Container
from quantities import km

# This is usually done by the SimulationExecutive, but it is done here
# manually for the purpose of testing and demonstration
dvc = Container()
DVarInputDict.register_dvar_container(dvc)

# Declare DVarInputDict exactly the same as an InputDict
class Foo(DVarInputDict):
    tam: Length
    bar: Length
    kap: LengthArray

foo = Foo(tam=44, bar=12*km, kap=[1, 2]*km )

print(foo)
# Foo(tam=44, bar=array(12.) * km, kap=array([1., 2.] * km)

foo.specNode().dumpValues()
# .dvarInputDictCommonBranch.Foo...
# {'bar': 12000.0, 'kap': [1000.0, 2000.0], 'tam': 44.0}

foo.specNode()["tam"].dump()
# name = tam
# type = double
# ...
# units = m
# quantityType = Length

foo.specNode()["bar"].dump()
# name = bar
# type = double
# ...
# units = m
# quantityType = Length

foo.specNode()["kap"].dump()
# name = kap
# type = doubleVector
# ...
# units = m
# quantityType = Length

# This is usually done by the SimulationExecutive, but it is done here
# manually for the purpose of testing and demonstration
DVarInputDict.close_class()
```

#### 2.2.1.8.4. DVarInputDict with None, empty list/tuple or deleted attribute

When a value is set to `None` or to an empty `list` or `tuple`, the DVar branch will not generate a Leaf for that attribute. Similarly, if a field is deleted or if it's a non-initialized optional field without default, no Leaf will be generated:

```
PYTHON
from DshellCommon.input_dict.dvar_input_dict import DVarInputDict
from DshellCommon.input_dict.input_dict import Dfield
from Dutils.typing import Optional, List
from DVar.DVar_Py import Container
from quantities import km

# This is usually done by the SimulationExecutive, but it is done here
# manually for the purpose of testing and demonstration
dvc = Container()
DVarInputDict.register_dvar_container(dvc)

# Declare DVarInputDict exactly the same as an InputDict
class Foo(DVarInputDict):
    tam: Optional[int]
    bar: List[int]
    baz: int
    kal: int = Dfield(required=False)

foo = Foo(tam=None, bar=[], baz=2)

foo.specNode().dumpValues()
# .dvarInputDictCommonBranch.Foo...
# {'baz': 2}

del foo["baz"]
foo.specNode().dumpValues()
# .dvarInputDictCommonBranch.Foo...
# {}

# This is usually done by the SimulationExecutive, but it is done here
# manually for the purpose of testing and demonstration
DVarInputDict.close_class()
```

### 2.2.1.8.5. Nested DVarInputDict

It is possible for `DVarInputDict` to contain other `DVarInputDict` within themselves. This means that a `DVarInputDict` attribute can be another `DVarInputDict`, a list of `DVarInputDict`, or a dictionary where the values are `DVarInputDict`. When this happens, the branch of the contained `DVarInputDict` will be a child of the branch of the container `DVarInputDict`:

```
from DshellCommon.input_dict.dvar_input_dict import DVarInputDict
from Dutils.typing import Dict, List
from DVar.DVar_Py import Container

# This is usually done by the SimulationExecutive, but it is done here
# manually for the purpose of testing and demonstration
dvc = Container()
DVarInputDict.register_dvar_container(dvc)

class Bar(DVarInputDict):
    name: str

# Declare DVarInputDict exactly the same as an InputDict
class Foo(DVarInputDict):
    tam: Bar
    kaz: List[Bar]
    par: Dict[str, Bar]

foo = Foo(
    tam=Bar(name="foo_tam"),
    kaz=[Bar(name="foo_kaz_0"), Bar(name="foo_kaz_1")],
    par={
        "child_a": Bar(name="foo_par_a"),
        "child_b": Bar(name="foo_par_b"),
    })

foo.specNode().dumpValues()
# .dvarInputDictCommonBranch.Foo_[]
# {'baz': 2}

foo.tam.specNode().dumpValues()
# .dvarInputDictCommonBranch.Foo_[]tam
# {'name': 'foo_tam'}

foo.kaz[0].specNode().dumpValues()
# .dvarInputDictCommonBranch.Foo_[]kaz.0
# {'name': 'foo_kaz_0'}

foo.par["child_a"].specNode().dumpValues()
# .dvarInputDictCommonBranch.Foo_[]par.child_a
# {'name': 'foo_par_a'}

# This is usually done by the SimulationExecutive, but it is done here
# manually for the purpose of testing and demonstration
DVarInputDict.close_class()
```

PYTHON

In the previous example, the spec string `".dvarInputDictCommonBranch.Foo_[]parchild_a"` shows that the branch corresponding to the `Bar` `DVarInputDict` in `foo.par["child_a"]` is a sub-branch of `foo's` branch `".dvarInputDictCommonBranch.Foo_[]"`.

### 2.2.1.8.6. Exclusivity

A `DVarInputDict` can only be used within another `DVarInputDict` when its branch is living in the `".dvarInputDictCommonBranch"` branch. This means that, as soon as the underlying branch is being used elsewhere (such as being part of an assembly `DVar` tree or already contained within another `DVarInputDict`), we cannot make this `DVarInputDict` a subelement of another `DVarInputDict`.

If one wants to move a `DVarInputDict` object from one `DVarInputDict` object to another, then the current container `DVarInputDict` has to stop "containing it":

```
from DshellCommon.input_dict.dvar_input_dict import DVarInputDict
from DVar.DVar_Py import Container

# This is usually done by the SimulationExecutive, but it is done here
# manually for the purpose of testing and demonstration
dvc = Container()
DVarInputDict.register_dvar_container(dvc)

class Bar(DVarInputDict):
    name: str

class SimpleBarContainer(DVarInputDict):
    my_bar: Bar

cont_1 = SimpleBarContainer(my_bar = Bar(name="my_bar"))

# This will cause an error because it would mean that cont_1.my_bar
# is stored both in cont_1 and cont_2
# cont_2 = SimpleBarContainer(my_bar=cont_1.my_bar)

my_bar = cont_1.my_bar
cont_1.my_bar = Bar(name="other_bar")
cont_2 = SimpleBarContainer(my_bar=my_bar)

# This is usually done by the SimulationExecutive, but it is done here
# manually for the purpose of testing and demonstration
DVarInputDict.close_class()
```

PYTHON



In this example, we replaced `cont_1.my_bar` with a new `Bar`. This made the old value (stored temporarily in `my_bar`) independent and thus able again of being a sub-element of a different `DVarInputDict` (`cont_2`).

#### 2.2.1.8.6.1. Updating sub-DVarInputDict

`DVarInputDict` objects are not deep-copied when an attribute is retrieved from a parent `DVarInputDict`. This allows modifying nested `DVarInputDict` in place. Continuing from the example in "Nested DVarInputDict":

```
foo.tam.name          = "new_foo_tam"
foo.kaz[0].name       = "new_foo_kaz_0"
foo.par["child_a"].name = "new_foo_par_a"
foo.specNode().dumpValues()
# {'kaz': {'0': {'name': 'new_foo_kaz_0'}, '1': {'name': 'foo_kaz_1'}},
#  'par': {'child_a': {'name': 'new_foo_par_a'},
#          'child_b': {'name': 'foo_par_b'}},
#  'tam': {'name': 'new_foo_tam'}}

# This is usually done by the SimulationExecutive, but it is done here
# manually for the purpose of testing and demonstration
DVarInputDict.close_class()
```

PYTHON

#### 2.2.1.8.7. Words of caution

Users should be careful when storing references of the underlying `Leaf` and `Branch` objects in a `DVarInputDict`. Certain operations can cause these objects to be destroyed and thus potentially cause segmentation faults:

```
from DshellCommon.input_dict.dvar_input_dict import DVarInputDict
from Dutils.typing import Dict
from DVar.DVar_Py import Container

# This usually done by the SimulationExecutive, but it is done here
# manually for the purpose of testing and demonstration
dvc = Container()
DVarInputDict.register_dvar_container(dvc)

class Foo(DVarInputDict):
    bar: Dict[str, int]

foo = Foo(bar = {"a": 0, "b": 1, "c": 2})

bar_a_leaf = foo.specNode()["bar"]["a"]
print(bar_a_leaf) # 0

foo.bar = {"b": 1, "c": 2}
# Foo[0].bar.a no longer exists, thus the following throws an error
# print(bar_a_leaf) # Segmentation fault

# This usually done by the SimulationExecutive, but it is done here
# manually for the purpose of testing and demonstration
DVarInputDict.close_class()
```

PYTHON

Users can prevent this from happening by defining the field types so that the "structure" of the DVar never needs to change. Things to avoid to keep the "structure" of a DVar unchangeable:

- Fields that accept values that parse into different Leaf objects. If a type is `Union[str, int]`, then both `"foo"` and `2` are acceptable but they each require a different kind of DVar Leaf (`StringLeaf`, `IntLeaf`) thus we would need to destroy one to create the other if the input types change.
- Fields that accept lists or arrays of different sizes. If the type is `List[int]`, then both `[1, 2]` and `[1, 2, 3]` are valid inputs. However, to accomodate variable-sized lists, we need to destroy and recreate the Leaf, as DVar Leaf have fixed dimensions.
- Dictionaries with arbitrary fields. Changing from `{"a": 0, "b": 1}` to `{"b": 1, "c": 2}` would require destroying the Leaf corresponding to "a" and creating a new one in "c".

Finally, note that the harmonization does not (currently) occur both ways: updating `DVarInputDict` will update the DVar branch, but updating the branch will not update the values stored in the `DVarInputDict`. Therefore, users are advised to not modify the Branch directly to prevent data mismatch.

#### 2.2.1.9. BaseDParam

`BaseDParam` are a subclass of `DVarInputDict` that allows setting the source for the value of each field. This source will then be used as the "source" attribute of the corresponding `Leaf` and `Branch` of the DVar tree.

This class provides all the functionality that old-style `BaseParam` (`"DshellCommon.params.BaseParam"`) did.

The source can be specified class-wide (with the `source` attribute), or field-specific (with the `sources`. `[FIELD NAME]` attribute). Specific sources will take priority over class-wide ones. Both types of sources can be provided on class initialization through `source` keyword (a string) or `sources` (a dict whose keys are field names):

```

from DshellCommon.input_dict.base_dparam import BaseDParam
from DshellCommon.input_dict.dvar_input_dict import DVarInputDict
from DVar.DVar_Py import Container

# This usually done by the SimulationExecutive, but it is done here
# manually for the purpose of testing and demonstration
dvc = Container()
DVarInputDict.register_dvar_container(dvc)

class Foo(BaseDParam):
    bar: int
    baz: int

foo = Foo(source="General source", bar=5, baz=12)
print(foo.specNode()["bar"].getAttribute("source")) # General source
print(foo.specNode()["baz"].getAttribute("source")) # General source

foo.source = "New general source"
print(foo.specNode()["bar"].getAttribute("source")) # New general source
print(foo.specNode()["baz"].getAttribute("source")) # New general source

foo = Foo(sources={"bar": "bar source"}, bar=5, baz=12)
print(foo.specNode()["bar"].getAttribute("source")) # bar source
print(foo.specNode()["baz"].getAttribute("source")) #

foo.source = "General source"
print(foo.specNode()["bar"].getAttribute("source")) # bar source
print(foo.specNode()["baz"].getAttribute("source")) # General source

foo.sources.bar = "new bar source"
print(foo.specNode()["bar"].getAttribute("source")) # new bar source
print(foo.specNode()["baz"].getAttribute("source")) # General source

foo.sources.bar = ""
print(foo.specNode()["bar"].getAttribute("source")) # General source
print(foo.specNode()["baz"].getAttribute("source")) # General source

foo.source = ""
print(foo.specNode()["bar"].getAttribute("source")) #
print(foo.specNode()["baz"].getAttribute("source")) #

# This usually done by the SimulationExecutive, but it is done here
# manually for the purpose of testing and demonstration
DVarInputDict.close_class()

```

### 2.2.1.9.1. Default sources

You can provide the source for a default value by creating the `Sources` subclass in your `BaseDParam`. Note that this subclass must inherit from `DshellCommon.input_dict.base_dparam.Sources`. Then, in this class definition, you can specify the default source of any field. You do not need to declare the source of all fields. When the object is created, if the default value is used, then the default source will also be used. Otherwise, the default source will be ignored:

```

from DshellCommon.input_dict.base_dparam import BaseDParam
from DshellCommon.input_dict.base_dparam import Sources
from DshellCommon.input_dict.dvar_input_dict import DVarInputDict
from DVar.DVar_Py import Container

# This usually done by the SimulationExecutive, but it is done here
# manually for the purpose of testing and demonstration
dvc = Container()
DVarInputDict.register_dvar_container(dvc)

class Foo(BaseDParam):
    bar: int = 5
    baz: int

    class Sources(Sources):
        bar: str = "Default bar source"

foo = Foo(baz=12)
print(foo.specNode()["bar"].getAttribute("source")) # Default bar source
print(foo.specNode()["baz"].getAttribute("source")) #

foo.source = "General source"
print(foo.specNode()["bar"].getAttribute("source")) # Default bar source
print(foo.specNode()["baz"].getAttribute("source")) # General source

foo = Foo(bar=10, baz=12)
print(foo.specNode()["bar"].getAttribute("source")) #
print(foo.specNode()["baz"].getAttribute("source")) #

# This usually done by the SimulationExecutive, but it is done here
# manually for the purpose of testing and demonstration
DVarInputDict.close_class()

```

## 2.2.2. DAssembly

### 2.2.2.1. Introduction

Go to Cheatsheet: Transforming old Assembly to DAssembly if you want to translate an existing `Assembly`. Note that the rest of this wiki is worth reading to gain extra insight.

DAssembly objects are collections of models, signals, bodies, frames... that work together to create some functionality in the simulation. A DAssembly can represent a vehicle, a sensor, a motor, a planet, a force, and many other possibilities.

DAssembly are thus high-level objects designed to facilitate simulation-building for users. Developers of DAssembly should think of ways to implement the desired functionality through lower level objects (models, signals, bodies, frames...). Then, these lower-level objects can be encapsulated within thematic DAssembly classes, which handle the creation and configuration of these objects.

DAssembly objects usually take user inputs to configure how the lower-level objects are created and configured. For example, a spacecraft DAssembly could take the the mass of the spacecraft as one of its inputs. By making DAssembly classes customizable, we increase reusability and thus maintainability.

This wiki covers how to define and use DAssembly objects.

### 2.2.2.2. DAssembly input definition

Inputs to DAssembly classes are organized into three "groups": parameter inputs ("params"), configuration inputs ("config"), and context inputs ("context"). There is very little functional distinction between these groups; whether an input should be a "param" or a "config" is up to the developer.

In general, however, "config" entries are used for inputs that change how or what lower-bodies are built. For example, through a "config" entry we might change the name of a node created by the DAssembly, or we might register additional frames. "context" entries, on the other hand, are usually related to how the DAssembly relates to other objects in the simulation. For example, a DAssembly that creates a gravity force on a body might take the object representing this body as a "context" entry. Finally, "param" inputs are used usually used to parametrize models or models. A DAssembly that creates a PID-controller model might take as input the Kp, Ki, and Kd parameters so that it can use this parameters to configure the PID model.

Every "config", "context", and "param" must have a name and a type. The name is a unique string identifier that should be written in `snake_case`. The type is represented through a python type annotation and is enforced at runtime. For example, an input might be given the type `List[int]`, and thus require that all user inputs for that field be lists of integers. "Extended" DARTS-flavoured typing is also supported, as described in this [wiki](https://dartsgitlab-internal.jpl.nasa.gov/darts-lab/jpl-internal/all-access/development/core-modules/dshellenv/-/wikis/docFiles/typing) (https://dartsgitlab-internal.jpl.nasa.gov/darts-lab/jpl-internal/all-access/development/core-modules/dshellenv/-/wikis/docFiles/typing). This means you can set arbitrary constraints for the inputs.

In terms of syntax, this information is written in the form of specially-named inner classes within the DAssembly class. These inner classes must also inherit from a special class, and are as a consequence a subclass of `DVarInputDict`. To understand the details of how `InputDict` and `DVarInputDict` work, see this [wiki](https://dartsgitlab-internal.jpl.nasa.gov/darts-lab/jpl-internal/all-access/development/simulation-framework/dshellcommon/-/wikis/InputDict-replacement-to-anonymous-dictionaries) (https://dartsgitlab-internal.jpl.nasa.gov/darts-lab/jpl-internal/all-access/development/simulation-framework/dshellcommon/-/wikis/InputDict-replacement-to-anonymous-dictionaries).

The following example shows how to declare a simple `DAssembly` class that implements no functionality except accepting user inputs:

```
from DshellCommon.assemblies.DAssembly import DAssembly
from Dutils.typing import Iterable
from Dutils.typing import Mapping
from Dutils.typing.Dtyping import FilePath

class MyDAssembly(DAssembly):

    class Config(DAssembly.Config):
        foo: Iterable[FilePath] = []
        bar: Mapping[str, int] = {}

    config: Config

    class Params(DAssembly.Params):
        distance: Length
        weights: MassArray
        factors: List[RealArray] = []

    params: Params

    class Context(DAssembly.Context):
        ...

    context: Context

    def __init__(
        self,
        parent: Union[DAssembly, AssemblyLegacy],
        name: str,
        config: Optional[Config] = None,
        context: Optional[Context] = None,
        signal_ties: Union[SignalSliceMap, Mapping[str, SignalSlice]] = {},
        params: Optional[Params] = None,
        description: str = "",
        tag: str = "",
    ):
        super().__init__(parent, name, config, context, signal_ties, params, description, tag)
```

PYTHON

In the above snippet we have declared the three inner classes `Params`, `Config`, and `Context`, which inherit from `DAssembly.Params`, `DAssembly.Config`, and `DAssembly.Context`. The `Config` and `Params` classes declare fields, while `Context` is empty (does not define any fields). Because of this, we can avoid defining the class and the runtime behaviour will be the same:

```

from DshellCommon.assemblies.DAssembly import DAssembly
from Dutils.typing import Iterable
from Dutils.typing import Mapping
from Dutils.typing.Dtyping import FilePath

class MyDAssembly(DAssembly):

    class Config(DAssembly.Config):
        foo: Iterable[FilePath] = []
        bar: Mapping[str, int] = {}

    config: Config

    class Params(DAssembly.Params):
        distance: Length
        weights: MassArray
        factors: List[RealArray] = []

    params: Params

    def __init__(
        self,
        parent: Union[DAssembly, AssemblyLegacy],
        name: str,
        config: Optional[Config] = None,
        context: Optional[DAssembly.Context] = None,
        signal_ties: Union[SignalSliceMap, Mapping[str, SignalSlice]] = {},
        params: Optional[Params] = None,
        description: str = "",
        tag: str = "",
    ):
        super().__init__(parent, name, config, context, signal_ties, params, description, tag)

```

Also note that we have made the `config` and `context` optional and have given them a default value of `None` in the `__init__` method. The convention dictates that a `None` value for `config` and `context` shall be equivalent to passing `MyDAssembly.Config()` or `MyDAssembly.Context()`. This means that for `None` to be supported, the `Config` or `Context` classes must be either empty or have only fields that have defaults. If these classes have a required field, and we attempt to create them without providing any information, we will get an error. Because of this, if `Config` or `Context` have at least a required field, then it's recommended that you do not mark them as optional:

```

def __init__(
    self,
    parent: Union[DAssembly, AssemblyLegacy],
    name: str,
    config: Config,
    context: DAssembly.Context,
    signal_ties: Union[SignalSliceMap, Mapping[str, SignalSlice]] = {},
    params: Optional[Params] = None,
    description: str = "",
    tag: str = "",
):
    super().__init__(parent, name, config, context, signal_ties, params, description, tag)

```

If you are defining a `DAssembly` that is a subclass of a different `DAssembly` (e.g. `TargetDAssembly` inherits from `TargetBaseDAssembly`), then the inner classes should also inherit from the superclass inner classes:

```

class MySubDAssembly(MyDAssembly):

    class Config(MyDAssembly.Config):
        bar: Mapping[str, float] = {"default": 42.0}
        extra: int = -1

    config: Config

    def __init__(
        self,
        parent: Union[DAssembly, AssemblyLegacy],
        name: str,
        config: Optional[Config] = None,
        context: Optional[MyDAssembly.Context] = None,
        signal_ties: Union[SignalSliceMap, Mapping[str, SignalSlice]] = {},
        params: Optional[MyDAssembly.Params] = None,
        description: str = "",
        tag: str = "",
    ):
        super().__init__(parent, name, config, context, signal_ties, params, description, tag)

```

The above class `MySubDAssembly` inherits from `MyDAssembly`. Similarly, `Config` inherits from `MyDAssembly.Config`. Any fields defined in the inner subclass will either override previous fields (such as the case of `bar` in the above snippet), or be added (such as the case of `extra`). `MySubDAssembly.Config` will therefore have the fields `foo`, `bar`, and `extra`, where `bar` has the type and default `Mapping[str, float]` and `{"default": 42.0}`. Since we have not defined either `Params` or `Context`, then these will be the same as the superclass inner classes (`MyDAssembly.Params`, and `MyDAssembly.Context`).

### 2.2.2.3. DAssembly functionality methods

Lower-level objects (models, bodies, frames, signals...) are created and configured in the methods `addSignals`, `addPendingTies`, `addModels`, `addAssemblies`, and `setModelBreaks`, `bindParams`, `bindStates`.

The following are descriptions of each method:

- `addSignals` :
- `addPendingTies` :
- `addModels` :
- `addAssemblies` :
- `setModelBreaks` :
- `bindParams` :
- `bindStates` :

In the above methods, you will need to access the values of user inputs (`config`, `context`, and `params`). To do so, use the attributes `config`, `context`, and `params`. The following snippet shows three examples of doing this. In the `addModels` method, we are accessing the values stored in `config`. In `bindParams`, the `specNode` stored in `params` is used in `modelParamSet` :

```
class MyDAssembly(DAssembly):
    class Config(DAssembly.Config):
        foo: Iterable[FilePath] = []
        bar: Mapping[str, int] = {}

    config: Config

    class Params(DAssembly.Params):
        distance: Length
        weights: MassArray
        factors: List[RealArray] = []

    params: Params

    def __init__(...): ...

    def addModels(self):
        for file_path in self.config.foo:
            ...

        for bar_k, var_v in self.config.bar.items():
            ...

    def bindParams(self):
        self.modelParamSet("MyModel", "Distance", "distance", self.params.specNode())
```

#### 2.2.2.4. Using DAssembly

To instantiate a `DAssembly`, the input data must be passed through the inner classes:

```
sim = SimulationExecutiveNdarts(...)

my_assembly = MyDAssembly(
    sim.topAssembly(),
    "MyAssemblyName",
    config = MyDAssembly.Config(
        bar = {"hello": 42}
    ),
    # context = MyDAssembly.Context(), <- This line is not needed, as the default None is equivalent to MyDAssembly.Context()
    params = MyDAssembly.Params(
        distance = 12*km,
        weights = [1*kg, 2*gr],
    ),
)
# On creation, the following are called: addSignals, addPendingTies, addModels, addAssemblies, and setModelBreaks.

sim.setupDynSolver() # This calls bindParams for all assemblies
sim.bindState()      # This calls bindState for all assemblies
sim.resetState(0.0)
sim.lockObject()

...

sim.close()
```

#### 2.2.2.5. Cheatsheet: Transforming old Assembly to DAssembly

##### 2.2.2.5.1. Basic pointers

- Instead of inheriting from `Assembly` in `Dshell.Dshell_Py`, inherit from `DAssembly` in `DshellCommon.assemblies.DAssembly`
- BaseParam classes that are transformed should be named `__DParam` (from `TargetParam` to `TargetDParam`)
- Assembly classes that are transformed should be named `__DAssembly` (from `TargetAssembly` to `TargetDAssembly`)
- If a `BaseDParam` and `DAssembly` are closely related, the `BaseDParam` should live in the `DAssembly` module (both `TargetDParam` and `TargetDAssembly` should live in `TargetDAssembly.py`)
- There is no need to call `.register()` for `DAssembly`
- Docstrings are heavily encouraged, both on the `DAssembly` class and its inner classes.

##### 2.2.2.5.2. Transforming BaseParam classes to BaseDParam

`DataField` objects have been deprecated in favour of the typing system explained in this [wiki](#)

(<https://dartsgitlab-internal.jpl.nasa.gov/darts-lab/jpl-internal/all-access/development/core-modules/dshellenv/-wikis/docFiles/typing>). `BaseParam` classes now inherit from `BaseDParam`, which are `InputDict` and thus follow their syntax (see the `InputDict` [wiki here](#) (<https://dartsgitlab-internal.jpl.nasa.gov/darts-lab/jpl-internal/all-access/development/simulation-framework/dshellcommon/-wikis/InputDict-replacement-to-anonymous-dictionaries>)).

To convert existing `BaseParam` classes to the new format (for example `TargetParam` which lives in `DshellCommon.params.TargetParam`), use the `param-conv.py` script:

```
srun param-conv.py TargetParam --class-module DshellCommon.params.TargetParam > TargetDParam.py
```

This will store the generated class source code in `TargetDParam.py`. Note that if the class you transform inherits from a different `Param` class, then you will also have to transform the super class.

The transformation will change all field names to snake\_case format and will make all nonrequired fields without default `Optional` type with default value `None`. **This transformation is not complete:** it merely adapts the `DataField`'s. Other functionality (such as custom `__init` behaviour) must be implemented by hand (for example, by implementing the `pre_type_check` abstract function).

### 2.2.2.5.3. Transforming param field declarations:

```
GRAVITY = "Gravity"
TARGET = "Target"

class GravityAssembly(Assembly):
    _param_types = {
        GRAVITY: GravityParam,
        TARGET: TargetParam,
    }
    _requiredParamFields = Assembly._requiredParamFields + [GRAVITY, ]
    _optionalParamFields = Assembly._optionalParamFields + [TARGET, ]
```

PYTHON

to

```
from Dutils.typing import Mapping

class GravityDAssembly(DAssembly):

    class Params(DAssembly.Params):
        """
        Parameters
        -----
        gravity: GravityDParam
            Description for the gravity parameter
        target: Mapping[str, TargetDParam]
            Descriptions can be
            multi-line

            and even have double line-breaks
        """
        gravity: GravityDParam
        target: Mapping[str, TargetDParam] = {}

    params: Params
```

PYTHON

In the code above:

- Instead of using the class variables `_param_types`, `_requiredParamFields`, `_optionalParamFields`, we define the inner class `Params` which inherits from the super class (`DAssembly`) inner class (`DAssembly.Params`).
- The field names are written (in snake\_case) within the inner class in the format: "name: type".
- Types are defined through the [typing system](#) (<https://dartsgitlab-internal.jpl.nasa.gov/darts-lab/jpl-internal/all-access/development/core-modules/dshellenv/-wikis/docFiles/typing>). In the old system, if `TARGET` took a dictionary of `TargetParam` (for example: `{"Earth": earthTarget, "Mars": marsTarget}`), then the type was simply `TargetParam`. Now, the fact that a dictionary can be accepted must be made explicit: `Mapping[str, TargetDParam]` (`Mapping` means anything dict-like).
- Default values can be given with the notation "name: type = default". For example, `target` was given the default `{}` (which will be deepcopied when used). A field will be considered required if it doesn't have a default value.
- The description for each field can be provided through the docstring in the format shown above.

### 2.2.2.5.4. Transforming config and context field declarations

```
SPIKE_KERNELS = "spiceKernels"

class GravityAssembly(Assembly):

    _requiredConfigFields = Assembly._requiredConfigFields + []
    _optionalConfigFields = Assembly._optionalConfigFields + [SPIKE_KERNELS]

    _requiredContextFields = Assembly._requiredContextFields + []
    _optionalContextFields = Assembly._optionalContextFields + []
```

PYTHON

to

```

from Dutils.typing import Union, List
from Dutils.typing.Dtyping import FilePath

class GravityDAssembly(DAssembly):

    class Config(DAssembly.Config):
        """
        Parameters
        -----
        spice_kernels: Union[FilePath, List[FilePath]]
            Description for the spice_kernels config
        """
        spice_kernels: Union[FilePath, List[FilePath]] = []

    config: Config

# Context need not be defined, as we don't need to add any field

```

We see:

- Config and context fields are defined the same as params fields: through inner classes ( Config and Context )
- The types need not be BaseDParam : they can be arbitrarily complex types (as defined in the [typing wiki](https://dartsgitlab-internal.jpl.nasa.gov/darts-lab/jpl-internal/all-access/development/core-modules/dshellenv/-wikis/docFiles/typing) (https://dartsgitlab-internal.jpl.nasa.gov/darts-lab/jpl-internal/all-access/development/core-modules/dshellenv/-wikis/docFiles/typing)). For example, the spice\_kernels fields accepts either a single file path (a string that points to a valid file), or a list of file paths.
- If you don't need to declare extra fields with respect to the super class, you do not need to specify the class (above, Context is not defined, which is interpreted as "use the superclass' Context'"). This is true for Params, Config, and Context . Note that DAssembly.Params , DAssembly.Config, and DAssembly.Context don't have any field.

#### 2.2.2.5.5. Constructor

The constructor **should** always be defined and type-hinted:

```

from Dutils.typing import Union
from Dutils.typing import Mapping
from Dutils.typing import Optional
from Dshell.Dshell_Py import SignalSlice, SignalSliceMap
from DshellCommon.assemblies.DAssembly import DAssembly
from DshellCommon.assemblies.AssemblyLegacy import AssemblyLegacy

class GravityDAssembly(DAssembly):

    ...

    def __init__(
        self,
        parent: Union[DAssembly, AssemblyLegacy],
        name: str,
        config: "GravityDAssembly.Config",
        context: "GravityDAssembly.Context",
        signal_ties: Union[SignalSliceMap, Mapping[str, SignalSlice]] = {},
        params: Optional["GravityDAssembly.Params"] = None,
        description: str = "",
        tag: str = "",
    ):
        super().__init__(parent, name, ...)

```

If Config or Context are empty or only have optional fields, these can be set to None and will be transformed to `_DAssembly.Config()` or `_DAssembly.Context()` :

```

def __init__(
    self,
    parent: Union[DAssembly, AssemblyLegacy],
    name: str,
    config: Optional["GravityDAssembly.Config"] = None,
    context: Optional["GravityDAssembly.Context"] = None,
    signal_ties: Union[SignalSliceMap, Mapping[str, SignalSlice]] = {},
    params: Optional["GravityDAssembly.Params"] = None,
    description: str = "",
    tag: str = "",
):
    super().__init__(parent, name, ...)

```

#### 2.2.2.5.6. Accessing data in params, config, or context

While most methods remains the same in the DAssembly class w.r.t Assembly , accessing the input data is done differently. Key aspects:

- Instead of `asm.paramObj("the_key")` do `asm.params.the_key` .
- `asm.getConfigValue("the_key")` and `asm.getContextValue("the_key")` are discouraged and should be replaced by `asm.config.the_key` , `asm.context.the_key` .
- Doing `asm.params()` no longer returns the params DVar branch. Instead, do `asm.params.specNode()` or `asm.paramsSpecNode()` . Similarly, doing `asm.config.specNode()` , `asm.configSpecNode()` , `asm.context.specNode()` , or `asm.contextSpecNode()` is supported.
- `asm.config()` and `asm.context()` are not supported, do `asm.config` or `asm.context` .
- By default, all params , config, and context objects are read-only. If you need to modify them within the DAssembly class, do `asm.params.unsafe_set_field("the_key", the_value, bypass_read_only=True)` (in this example params can be replaced with config or context ) .
- The values within BaseDParam classes should be accessed as attributes: `asm.params.gravity.apply_torque` is preferred over `asm.params.gravity["apply_torque"]` .

### 2.2.2.5.7. Constructing a DAssembly object

You must use the inner classes for the `config`, `context`, and `params` :

```
asm = GravityDAssembly(  
    sim.topAssembly(),  
    "MyGravity",  
    config = GravityDAssembly.Config(spice_kernels = ["home/..."]),  
    params = GravityDAssembly.Params(  
        gravity = GravityDParam(apply_torque=True, ...),  
        target = {  
            "Earth": TargetDParam(mass=42 * kg, ...),  
            "Mars": TargetDParam(mass=24 * kg, ...),  
        }  
    ),  
)
```

PYTHON

### 2.2.2.5.8. Parameters, config, and context that are used in multiple assemblies

In certain cases, especially in vehicle assemblies, parameters may be passed from a parent to a child. In this case, one must remember to detach the `specNode` from the parent class before it is passed to the child class. Otherwise, they will end up with errors like the following:

```
ValueError: Cannot create Branch from DVarInputDict asm2_param(v=3) because that DVarInputDict's  
specNode is already present in another branch (current spec: .Dshell.Assemblies.asm1.param.p2.p1.  
This might be because it is already nested (used) within another DVarInputDict. If so, Remove  
this DVarInputDict from the other DVarInputDict before proceeding.
```

Here is a simple, standalone example where this is done incorrectly:

```
from DshellCommon.assemblies.DAssembly import DAssembly  
from DshellCommon.input_dict.base_dparam import BaseDParam  
from Dutils.typing import Dict  
from DshellCommon.SimulationExecutiveNdarts import SimulationExecutiveNdarts  
  
class asm2_param(BaseDParam):  
    v : int  
  
class Asm2(DAssembly):  
    class Params(DAssembly.Params):  
        p2 : asm2_param  
  
class Asm1(DAssembly):  
    class Params(DAssembly.Params):  
        p2 : Dict[str, asm2_param]  
        params : Params  
  
    def addAssemblies(self):  
        for k,p in self.params.p2.items():  
            Asm2(self, k, params=Asm2.Params(p2=p))  
            super().addAssemblies()  
  
sim = SimulationExecutiveNdarts()  
p1 = asm2_param(v=3)  
p2 = asm2_param(v=4)  
params = Asm1.Params(p2={"p1":p1, "p2":p2})  
asm = sim.topAssembly()  
  
my_asm = Asm1(asm, "asm1", params=params)
```

PYTHON

To fix it, simply add a call to `detachSpecNode()` on the parameter being passed from `Asm1` to `Asm2`, like this:



```

from DshellCommon.assemblies.DAssembly import DAssembly
from DshellCommon.input_dict.base_dparam import BaseDParam
from Dutils.typing import Dict
from DshellCommon.SimulationExecutiveNdarts import SimulationExecutiveNdarts

class asm2_param(BaseDParam):
    v : int

class Asm2(DAssembly):
    class Params(DAssembly.Params):
        p : asm2_param

class Asm1(DAssembly):
    class Params(DAssembly.Params):
        p : Dict[str, asm2_param]
    params : Params

    def addAssemblies(self):
        for k,p in self.params.p.items():
            p.detachSpecNode()
            Asm2(self, k, params=Asm2.Params(p=p))
            super().addAssemblies()

sim = SimulationExecutiveNdarts()
p1 = asm2_param(v=3)
p2 = asm2_param(v=4)
params = Asm1.Params(p={"asm2_a":p1, "asm2_b":p2})
asm = sim.topAssembly()

asm1 = Asm1(asm, "asm1", params=params)
asm2_a = asm1.assemblyList()[0]
asm2_b = asm1.assemblyList()[1]

# Modify the param via the asm1 specNode
asm1.specNode()["asm2_a"]["param"]["p"]["v"](5)

# Modify a param via the Python interface
asm1.params.p["asm2_b"].v = 6

# Print out the final results in the specNode
print(asm1.specNode())

```

[Here](https://dartsgitlab-internal.jpl.nasa.gov/darts-lab/jpl-internal/all-access/development/simulation-framework/dshellcommon/-/issues/38) (https://dartsgitlab-internal.jpl.nasa.gov/darts-lab/jpl-internal/all-access/development/simulation-framework/dshellcommon/-/issues/38) is the issue where these examples were pulled from.

#### 2.2.2.5.9. Examples

The following classes have an Assembly and DAssembly version: TargetBaseDAssembly, TargetDAssembly, TargetBallDAssembly, TargetSpiceDAssembly, TargetSpiceFrameDAssembly, GravityBaseDAssembly (in GravityModels), NBodyGravityDAssembly (in GravityModels).

The following tests illustrate usage:

- DshellCommon/test/test\_Ndarts/test\_TargetSpiceDAssembly
- DshellCommon/test/test\_Ndarts/test\_TargetDAssemblies
- GravityModels/test/test\_NBodyGravityDAssembly
- AeroModels/test/test\_AeroCmSensor\_DAssembly

#### 2.2.2.5.10. Gotchas

Errors regarding passing non-pickleable values from `config`, `context`, or `params` are typically due to making calls to them in the assembly class like `self.config().my_value` rather than `self.config.my_value`. Calling `self.config()` rather than `self.config` attempts to make a copy of the config class via `deepcopy`, and, if values stored in it are not pickleable, then an error is thrown.

Errors with text such as `Cannot create Branch from DVarInputDict` are typically due to sharing `DVarInputDict`'s`, i.e., sharing ``param`s`, ``config`s`, or ``context`s` across assemblies without calling the ``detachSpecNode` method. See above for more details.

## 2.3. Usage

### 2.3.1. Module Usage

#### 2.3.1.1. Parameters

##### 2.3.1.1.1. modelLinker

`modelLinker` is a static method of the `BaseParam` class that helps automatically populate data fields in a parameter class using the parameter data in a given model class. The model class is specified via a string as the first argument to the method.

The data fields names in the parameter class will have the same names as the parameter names in the model class, plus an optional prefix (default is blank) prepended to the beginning of each name that is specified via a keyword argument to `modelLinker`. The remaining keyword arguments of `modelLinker` are:

- `exclude` - Any model parameter whose name matches an element in this list (of type string) will be skipped.
- `mdl_hints` - This dictionary is used to add to/overwrite the information included from the model. The keys of this dictionary are strings that match the names of the parameters in the model, and the values are dictionaries whose key/value pairs specify the data to be added / overwritten.

`modelLinker` returns a dictionary whose keys are the data field names and whose values are the data field classes. This dictionary's values are meant to be added directly to the parameter list of the parameter class being created. The examples that follow elaborate.

**Example** Suppose we wanted to create a control parameter that includes a frequency parameter and all the parameters of the `GeneralModels.SinglePID` model except the `maxOutput` parameter. Moreover, we want to add/modify the information coming from the model so that the defaults for the proportional, derivative, and integral gains are 1.0. This can be accomplished as follows:

```
# "mdl_hints" are used to add information or modify existing information
mdlParamData = {"kp": {"default_value": 1.0}, "kd": {"default_value": 1.0}, "ki": {"default_value": 1.0}}

# Exclude is used to exclude information from modelLinker
exclude = ["maxOutput"]

class ControlParam(BaseParam):
    _params = {
        "frequency" : FloatDataField(default_value=100, description="Controller frequency [Hz]"),
        **BaseParam.modelLinker("GeneralModels.SinglePID", exclude=exclude, mdlParamData=mdlParamData)
    }
```

PYTHON

The `mdl_hints` keyword was used here to add/modify the default value of the proportional, derivative, and integral gains. Furthermore, the `exclude` keyword was used to skip the `maxOutput` parameter of `GeneralModels.SinglePID`. As specified earlier, `modelLinker` outputs a dictionary; hence, the `**` Python operator was used to unpack this dictionary into the `_params` dictionary.

The full version of this example and other examples of `modelLinker` can be found in the `DshellCommon` module under `test/test_basic/test_model_linker`. In addition, [this team talk](https://dartslab.jpl.nasa.gov/technotes/Talks/2021-11-4-leake-param-linker.mp4) (<https://dartslab.jpl.nasa.gov/technotes/Talks/2021-11-4-leake-param-linker.mp4>) discusses `modelLinker` and how to use it.

`modelLinker` automatically sets a dictionary as the `mdlSource` variable of the associated data fields of the parameter class that can be used to set model parameters at run time. For example,

```
exclude = ["maxOutput"]
pidModel.setModelLinkerParams(controlParamObj, exclude=exclude)
```

PYTHON

would set the model parameters of a `GeneralModels.SinglePID` model (called `pidModel` in this example) using a `ControlParam` instance (called `controlParamObj` in this example). Again, the `exclude` keyword can be used to indicate model parameters that are excluded from the parameter class. It is important that anything excluded when calling `modelLinker` be included in the `exclude` list when calling `setModelLinkerParams` as `setModelLinkerParams` checks that all model parameters except those in the `exclude` list have been set. If this is not the case, then `setModelLinkerParams` will throw an error. For a similar example written out in full, see `test/test_basic/test_model_linker/setParams`.

### 2.3.1.1.2. ParamLinker

`ParamLinker` is a class in the `BaseParam` module. In addition, the `BaseParam` class (and all derived classes) provide methods that create and utilize the `ParamLinker` class. The latter is the more common way to use `ParamLinker`.

`ParamLinker` is designed to help automatically populate the options of a `click` command (i.e., it is designed to be used with `Dclick`) using the data fields in a parameter class. In addition, `ParamLinker` is used to create instances of parameter classes given the corresponding sections of the `Dclick` options dictionary. `ParamLinker` is typically utilized via the `cli` and `createParam` methods of any parameter class derived from the `BaseParam` class.

`cli` is a decorator that decorates the corresponding function with `click` options that correspond to the data fields of the associated class. The keywords of this class are similar to `modelLinker`:

- `prefix` - Used to prepend a prefix to the names of the `click` options.
- `exclude` - Any parameter whose name matches an element in this list (of type string) will be skipped.
- `mdl_hints` - This dictionary is used to add to /overwrite the information included from the model. The keys of this dictionary are strings that match the names of the parameters in the model, and the values are dictionaries whose key/value pairs specify the data to be added / overwritten.
- `remove_defaults` - This boolean is used to remove all defaults from the options added to the click command.

**Example** Suppose we wanted to create a `Dclick` control command using the parameter class that we generated in the `modelLinker` example. Moreover, suppose we want to add an additional option called `--is-enabled`:

```
@Dclick.cli.command()
@click.pass_context
@click.option("--is-enabled", type=bool, default=True, show_default=True, help="Enable accelerometer assembly.")
@ControlParam.cli()
def control(ctx, **kwargs):
    """
    Options to pass to the control assembly.
    """

    Dclick.add_options("control", Dclick.cli.commands["control"], kwargs)
}
```

PYTHON

Here, the `cli` method of the `ControlParam` parameter class was used to automatically populate the `Dclick` command `control` with options that correspond to the data fields of the `ControlParam` class.

Suppose later on we want to take the `Dclick` options dictionary, let's call it `cfgobj`, and use it to create an instance of `ControlParam`. This can be done using the `createParam` method like so:

```
controlParamObj = ControlParam.createParam(cfgobj["control"])
```

PYTHON

The full version of this example can be found in and other examples of `ParamLinker` can be found in `DshellCommon` module under `test/test_basic/test_model_linker`. Other examples of `ParamLinker` can be found in the `DshellCommon` module under `test/test_basic/test_param_linker`. In addition, [this team talk](https://dartslab.jpl.nasa.gov/technotes/Talks/2021-11-4-leake-param-linker.mp4) (<https://dartslab.jpl.nasa.gov/technotes/Talks/2021-11-4-leake-param-linker.mp4>) discusses `ParamLinker` and how to use it.

#### 2.3.1.1.2.1. Unit specification

If a parameter class has a data field with a quantity, then the option `--param-units` will automatically be added the associated `Dclick` options. The `--param-units` option is a multi-option, meaning it can be specified more than once, and it takes in two strings. The first of these strings specifies the data field for which units are being specified, and the second string is the units. For example, to set the units of the data field `radius` to feet, one would use `--param-units radius ft`. Since `--param-units` stores all values passed to it, if a value is specified more than once, e.g., `radius` is specified in meters in a configuration file and then in feet on the command line, the most recent version will be used.

Note that whenever a `--param-units` value is specified, the value of that parameter must be specified in the same place as well. For example, you cannot specify the `--param-units` for `radius` in a config file, without also specifying the value for `radius` in the same config file. The reason it is implemented this way is to avoid accidental unit conversions by the user. For example, if `radius` had a default value of 2, and that default value is supposed to be in meters, and the user specified the `--param-units` for `radius` to be `ft` in a config file with no associated value, then the sim will get a value of 2 feet, which is likely incorrect. Thus, users must specify a value for the parameter they are specifying `--param-units` for in the same place they specify `--param-units`. However, if units are specified along with a value, that value can be changed downstream without having to specify units again. For example, suppose we again specify that `radius` should have units of `ft` in a config file, and we set the value of `radius` to be 1 in that config file. Then, later on in the CLI we specify that `radius` has value 2, without giving any specification for the units. This will not throw any errors, and the value passed from `Dclick` to the sim will be 2 ft, i.e., using the units specified in the config file.

During the `createParam` call, any units specified via the `--param-units` option are applied to the associated data fields.

A full team talk on this topic can be found [here](https://dartslab.jpl.nasa.gov/technotes/Talks/2022-04-21-sprint-param-units.mp4) (<https://dartslab.jpl.nasa.gov/technotes/Talks/2022-04-21-sprint-param-units.mp4>).

## 2.4. Software

## 2.5. Raw documentation



TBD: Need scrubbing before integration.

### 2.5.1. DshellCommon: Add an optional reference frame path/uuid field to NodeParam class



TBD: Needs scrubbing. Notes brought over from [issue](https://dartsgitlab-internal.jpl.nasa.gov/darts-lab/jpl-internal/all-access/development/simulation-framework/dshellcommon/-/issues/7) (<https://dartsgitlab-internal.jpl.nasa.gov/darts-lab/jpl-internal/all-access/development/simulation-framework/dshellcommon/-/issues/7>)

Currently, the `NodeParam` class is used to specify the pose of a body node wrt its parent body frame. However, there are times when we need to specify the pose in a different frame (eg. for sensors). This has come up for the CADRE project where they would like to specify frame poses wrt designated frames independent of our body frames. Having to compute the actual body relative pose and pass it in can be ugly.

When this is desired, one way to solve this is to pass in information about the reference frame (eg. via a frame uuid, or a string path) that the class can use to look up the reference frame. This can be used to internally compute the actual body relative pose to set the `body2node` transform for the node.

Since the reference frame is expected to be set by the external user, using a string identifier such as the frame `name` may be preferable over having to specify its `uuid`. However, the frame name itself may not be unique, so we may need to use something like an abbreviated path such as `A.B...C` where `A`, `B` etc are frame names used to narrow down the context for finding the frame. For instance the chassis frame on `rvr3` may be specified via `rvr3.chassis` etc to avoid conflict with the chassis frames on other vehicles.

### 2.5.2. DshellCommon: Create MarsAssembly etc assemblies specific to known planetary bodies



TBD: Needs scrubbing. Notes brought over from [issue](https://dartsgitlab-internal.jpl.nasa.gov/darts-lab/jpl-internal/all-access/development/simulation-framework/dshellcommon/-/issues/2#note_7093) ([https://dartsgitlab-internal.jpl.nasa.gov/darts-lab/jpl-internal/all-access/development/simulation-framework/dshellcommon/-/issues/2#note\\_7093](https://dartsgitlab-internal.jpl.nasa.gov/darts-lab/jpl-internal/all-access/development/simulation-framework/dshellcommon/-/issues/2#note_7093)).

Key design elements of the proposed implementation for this are:

- All python modules will live in `DshellCommon/python/assemblies/known_targets`
- Two abstract classes are created: `KnownTargetAssembly` (subclass of `TargetSpiceAssembly`) and `KnownTargetParams` (subclass of `TargetSpiceParam`). They declare class attributes that are to be defined by subclasses.
- For each target, subclasses of `KnownTargetAssembly` and `KnownTargetParams` should be declared (e.g. `MarsTargetAssembly` and `MarsTargetParams`).
- `KnownTargetParams` implements `init_subclass`. This method parses the class attributes and changes the defaults of the data fields contained in `KnownTargetParams`. This way, instantiating an instance of a subclass of `KnownTargetParams` without any input will have enough default information to be a complete `TargetSpiceParam` (while also allowing for default overriding).
- `KnownTargetParams` also implements `init`, which adds the source of the default values.
- `KnownTargetAssembly` marks the `TARGET` param as optional (it is required in `TargetSpiceAssembly`). The `init` method is implemented to pass a default instance of the appropriate `KnownTargetParams` to `TargetSpiceAssembly`. `init` in case none is provided by the user.
- `KnownTargetAssembly`. `init` also adds a default `KnownTopoConfig` to the config if this is available for that known target.
- All implemented targets provide basic information (mass, mu, radius...). Depending on the availability of data, some also provide spherical harmonics coefficients and known topos.
- `DshellCommon/doc/source/guide/targets.rst` has been updated to reflect the existence of these `KnownTargetAssembly`.

This is an example of the declaration of Mars-specific classes, which would live in a single module `MarsTargetAssembly.py`:

```

class MarsTargetParam(KnownTargetParam):

    _default_params = {
        NAME: "Mars",
        MU: MARS_MU_DEFAULT,
        MASS: 0.64169e24 *kg,
        ROTATION_RATE: (2*pi *rad) / (24.6229 *hr),
        RADIUS_EQUATOR: 3396.2 *km,
        RADIUS_POLE: 3376.2 *km,
        SPH_HARM_FILE: os.path.join(os.environ["YAM_ROOT"], "etc", "GravityModelData", "MRO", "MRO120F.tab"),
        BODY_ID: 499,
    }

    _default_params_source = {
        MU: MARS_MU_DEFAULT_SOURCE,
        MASS: 'https://nssdc.gsfc.nasa.gov/planetary/factsheet/marsfact.html',
        ROTATION_RATE: 'https://nssdc.gsfc.nasa.gov/planetary/factsheet/marsfact.html',
        RADIUS_EQUATOR: 'https://nssdc.gsfc.nasa.gov/planetary/factsheet/marsfact.html',
        RADIUS_POLE: 'https://nssdc.gsfc.nasa.gov/planetary/factsheet/marsfact.html',
        SPH_HARM_FILE: '''Konopliv, A. S., Park, R. S., Rivoldini, A., Baland, R. M., Le Maistre, S., Van Hoolst, T., ... & Dehant, V. (2020). Detection of the Chandler wobble of Mars from orbiting spacecraft. Geophysical Research Letters, 47(21), e2020GL090568.'''
        BODY_ID: "https://naif.jpl.nasa.gov/pub/naif/toolkit_docs/C/req/naif_ids.html",
    }

class MarsTargetAssembly(KnownTargetAssembly):

    _param_types = {
        **KnownTargetAssembly._param_types,
        TARGET: MarsTargetParam,
    }

    _default_topos_config = "SphericalMars"

```

To instantiate the target assembly (with all default params and config), the user would simply use:

```

mars = MarsTargetAssembly(sim, params={EPOCH: epoch})

```

Alternatively, if they want to override one parameter (but keep the rest of the defaults), they may:

```

mars = MarsTargetAssembly(
    sim,
    params={
        TARGET: MarsTargetParam(mu=42)
        EPOCH: epoch
    })

```

or override the default topos:

```

mars = MarsTargetAssembly(
    sim,
    params={EPOCH: epoch},
    config={TOPOS : {"zero" : TopoAnalyticDemConfig(topo="zero")}}
)

```

The following known targets are available:

- CallistoTargetAssembly
- DeimosTargetAssembly
- EarthTargetAssembly (KnownTopos ID: EarthWGS84, Spherical Harmonics: GGM03C)
- EnceladusTargetAssembly (KnownTopos ID: SphericalEnceladus)
- EuropaTargetAssembly (KnownTopos ID: SphericalEuropa)
- GanymedeTargetAssembly
- IoTargetAssembly
- JupiterTargetAssembly (KnownTopos ID: Jupiter; Spherical Harmonics: jupiter)
- MarsTargetAssembly (KnownTopos ID: SphericalMars, Spherical Harmonics: MRO120F)
- MercuryTargetAssembly (Spherical Harmonics: MESS160A)
- MoonTargetAssembly (KnownTopos ID: SphericalMoon, Spherical Harmonics: GL0900D)
- NeptuneTargetAssembly (KnownTopos ID: Neptune, Spherical Harmonics: neptune)
- PhobosTargetAssembly (KnownTopos ID: SphericalPhobos)
- PlutoTargetAssembly
- SaturnTargetAssembly (Spherical Harmonics: saturn)
- SunTargetAssembly (KnownTopos ID: SphericalSun)
- TitanTargetAssembly (KnownTopos ID: Titan)

- UranusTargetAssembly (Spherical Harmonics: uranus)
- VenusTargetAssembly (KnownTopos ID: Venus, Spherical Harmonics: MGNP180U)

### 2.5.3. DshellCommon: New Object-Oriented way to handle Assemblies (design requirements)



**TBD:** Needs scrubbing. Notes brought over from [issue](https://darts.gitlab-internal.jpl.nasa.gov/darts-lab/jpl-internal/all-access/development/simulation-framework/dshellcommon/-/issues/19) (<https://darts.gitlab-internal.jpl.nasa.gov/darts-lab/jpl-internal/all-access/development/simulation-framework/dshellcommon/-/issues/19>).

Design notes for the creation of the `InputDict` and `DAssembly` classes.

Currently, instantiating new assemblies is a slow, error-prone job when you are not familiar with their format and required inputs.

- The IDE cannot give any hints with respect to the inputs, as all Assembly constructors have the same signature.
- Information about how to create/use these assemblies is spread over video tutorials, notebooks, tests, docs, gitlab issues and wikis, Q&As, or the source code. Having to navigate so many resources is slow.
- Current approach is data-oriented (dicts within dicts within dicts full of “magic strings”), which is anti-pythonic and it is easy to make mistakes (easy to misspell something or forget a parameter)
- Sometimes, when creating two different assemblies, the same information needs to be input twice, which leads to errors / inconsistencies / longer syntax.

The use of Param classes was a step in the direction of using objects instead of anonymous dictionaries, while still retaining backward compatibility. These classes should serve as inspiration for further steps in this direction.

Requirements for solutions:

- The IDE should show as much information as possible
  - Methods must be type-hinted ([see](https://docs.python.org/3/library/typing.html) (<https://docs.python.org/3/library/typing.html>))
  - Method arguments shall be explicitly defined. Whenever possible, avoid the use of dictionaries to group inputs and `*arg` and `**kwargs` arguments.
- Methods should be mostly self-documenting through docstrings and type-hints.
- Avoid making the user input the same information twice
- Completely (runtime) backward compatible. It is acceptable (even desirable) for the IDE to highlight any old syntax as “incorrect”, but at runtime old code must work as expected.

TODO: attach link(s) to presentation(s) on this issue

#### 2.5.3.1. Assembly constructors

Assembly constructors should be one of the main objectives of this refactor. Currently, all Assembly constructors take the shape:

```
def __init__(
    self,
    parent_asm,
    name,
    config,
    context,
    signal_ties=None,
    params=None,
    description="",
    tag="",
):
    """ Initializer """
```

The relevant information is hidden within the dictionaries `config`, `context`, and `params`. By simple inspection, it is not possible to determine the required content of each of these dictionaries. Moreover, IDEs will not be able to show any useful information.

##### 2.5.3.1.1. Constructors follow up

This comment aims to introduce a way to instantiate Assembly that allows us to retain the current structure for the `__init__`, yet enhances it so that it is clearer through type hints and docstrings. This solution is fully backward compatible.

Each Assembly subclass would include three inner classes: `Config`, `Context`, and `Params`. These classes would declare the content (and its type) of each of the corresponding dictionaries. These inner classes are python `dataclass`es (see [python docs](https://docs.python.org/3/library/dataclasses.html#module-dataclasses) (<https://docs.python.org/3/library/dataclasses.html#module-dataclasses>)) for ease of declaration. These classes can then be transformed into current-style dictionaries, so that the constructor can treat them as they did before.

The following implementation shows only a `Config` implementation, but the other two dictionaries would be equivalent:

```

from dataclasses import dataclass, asdict
from typing import List, Union, Optional, Any

class Assembly:
    """Dummy Assembly class"""

    @staticmethod
    def dataclass_to_dict(dataclass) -> dict[str, Any]:
        dict_ = asdict(dataclass)

        # None values are removed from the dictionary
        for key in list(dict_.keys()):
            if dict_[key] is None:
                dict_.pop(key)

        return dict_

class TargetAssembly(Assembly):

    @dataclass
    class Config:
        """Config of the TargetAssembly class.

        Parameters
        -----
        topo: str
            A test string argument
        foo: List[int]
            A test list of int argument
        mass: Optional[float]
            A test optional float argument
        """
        topo: str
        foo: List[int]
        mass: Optional[float] = None

    def __init__(
        self,
        config: Union[Config, dict[str, Any]],
    ) -> None:
        # Config objects are turned to dictionaries -> backward compatibility
        if isinstance(config, type(self).Config):
            config = type(self).dataclass_to_dict(config)

        self.config = config

```

The dummy class `TargetAssembly` declares the inner dataclass `Config`. This class can have a docstring as detailed as desirable so that the use of every field is clear to any user. The fields of this `Config` (`topo`, `foo`, `mass`) are declared below, along with their types.

Optional fields (`mass`) are differentiated from required fields (`topo`, `foo`) because optional fields indicate the default value = `None`. When the object is transformed into a dictionary, any entries with a value equal to `None` are removed from the dictionary. This is done in line with current behavior, which identifies whether or not a user specified a field by determining if the relevant key is present in the dictionary.

In the constructor of the assembly, we see that `config` takes either a `dict[str, Any]` (in line with current behavior) or an object of type `Config`. However, if a `Config` object is passed, the initializer will quickly convert it to its equivalent dictionary with the `Assembly.dataclass_to_dict` function.

Using a regular dictionary or a `Config` object should render exactly the same behavior:


```

OO_style_target = TargetAssembly(
    config = TargetAssembly.Config(
        topo = "test",
        foo= [1,2]
    )
)

Data_style_target = TargetAssembly(
    config = {
        "topo": "test",
        "foo": [1,2]
    }
)

assert OO_style_target.config == Data_style_target.config

```

However, using the Object-Oriented syntax would create more helpful IDE messages:  `config_init`

 `config_init_ide_error`

 `config_init_ide_missing`

And even fail at runtime if not all required inputs are given:

```

TargetAssembly.Config(topo = "test")
TypeError: __init__() missing 1 required positional argument: 'foo'

```

The new `Config`, `Context`, and `Param` classes could be leveraged to automate the declaration of the required and optional config, context, and param. To do so, the parent `Assembly` class should implement the `__init_subclass__` method. Modify the previous code to include the following:

```
from dataclasses import dataclass, asdict, fields, MISSING
from typing import List, Union, Optional, Any

class Assembly:
    _requiredConfigFields = []
    _optionalConfigFields = []

    _config_type = {}

    def __init_subclass__(cls) -> None:
        for input_dict in ("Config",):
            if not hasattr(cls, input_dict):
                continue

            for field in fields(getattr(cls, input_dict)):
                if field.default is MISSING and field.default_factory is MISSING:
                    cls._optionalConfigFields.append(field.name)
                else:
                    cls._requiredConfigFields.append(field.name)

            # Retrieve original type from Optionals
            # Optional[float] -> float
            if (
                hasattr(field.type, "__origin__")
                and hasattr(field.type, "__args__")
                and field.type.__origin__ is Union
                and len(field.type.__args__) == 2
                and type(None) in field.type.__args__
            ):
                typ = list(field.type.__args__)
                typ.remove(type(None))
                cls._config_type[field.name] = typ[0]
            else:
                cls._config_type[field.name] = field.type
```

With this addition, all subclasses of `Assembly` would automatically populate the `_requiredConfigFields`, `_optionalConfigFields`, and `_config_type` class attributes:

```
>>> print(TargetAssembly._requiredConfigFields)
['mass', 'foo']
>>> print(TargetAssembly._optionalConfigFields)
['topo']
>>> print(TargetAssembly._config_type)
{'topo': <class 'str'>, 'mass': <class 'float'>, 'foo': typing.List[int]}
```

### 2.5.3.1.3. More on Config, Context inheritance

`Config`, `Context`, and `Params` objects may inherit from the equivalent parent class' objects.

For example, the subclass `TargetSpiceAssembly` of `TargetAssembly`, could be written as:

```
class TargetSpiceAssembly(TargetAssembly):
    @dataclass
    class Config(TargetAssembly.Config):
        """
        ...
        """
        bodyID: Optional[int] = None

    def __init__(
        self,
        config: Union[Config, dict[str, Any]],
    ) -> None:
        TargetAssembly.__init__(self, config)
```

and then instantiated as:

```
>>> 00_style_target = TargetSpiceAssembly(
...     config = TargetSpiceAssembly.Config(
...         topo = "test",
...         foo= [1,2],
...         bodyID= 500
...     )
... )

>>> print(00_style_target.config)
{'topo': 'test', 'foo': [1, 2], 'bodyID': 500}

>>> print(TargetSpiceAssembly._config_type)
{'topo': <class 'str'>, 'foo': typing.List[int], 'mass': <class 'float'>, 'bodyID': <class 'int'>}
```

However, a major problem is that one cannot define required fields in subclasses if the parent class has optional fields. In the previous example, declaring `bodyID` as a required field would result in an error. A solution for this issue is available in python 3.10 (using `kw_only`).

EDIT: A solution for python < 3.10 is possible, although it's not very elegant:

```
from dataclasses import dataclass, field, fields
from typing import TYPE_CHECKING

class _NOT_DEFINED_TYPE:
    pass
NOT_DEFINED = _NOT_DEFINED_TYPE()

REQUIRED_FIELD = field if TYPE_CHECKING else lambda *args, **kw: field(default=NOT_DEFINED, *args, **kw)

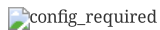
@dataclass
class InputConfig:
    def __post_init__(self) -> None:
        for field in fields(self):
            if getattr(self, field.name) is NOT_DEFINED:
                raise TypeError(f"Missing required argument: '{field.name}'")

@dataclass
class Parent(InputConfig):
    foo: int
    bar: str = "SAP"

@dataclass
class Child(Parent):
    baz: float = REQUIRED_FIELD()
```

In the previous syntax, `Parent` defines an optional argument `bar`. The `Child` class would like to define a required field `baz`, but this is usually not possible. Instead, we give a "fake" default `REQUIRED_FIELD()`. On instantiation, if this parameter is not defined, an error will be raised.

The use of `TYPE_CHECKING` in `REQUIRED_FIELD` is used to trick the IDE into believing that the argument does NOT have a default, and thus it will show it as a required field:

 config\_required

#### 2.5.3.1.4. Config, context docstrings creation

It is possible to provide extra information for `dataclass` fields by using the `field` method. In particular, the `metadata` argument is of interest since it can hold arbitrary information, which we can leverage. For example, we may choose to declare the description of each field within `metadata`, instead of in the docstring. This can help automate docstrings for subclasses.

For example, let's consider the previous class and subclass:

```
class TargetAssembly(Assembly):

    @dataclass
    class Config:
        """Config of the TargetAssembly class.

        Parameters
        -----
        topo: str
            A test string argument
        foo: List[int]
            A test list of int argument
        mass: Optional[float]
            A test optional float argument
        """
        topo: str
        foo: List[int]
        mass: Optional[float] = None

    ...

class TargetSpiceAssembly(TargetAssembly):

    @dataclass
    class Config(TargetAssembly.Config):
        """Config of the TargetAssembly class.

        Parameters
        -----
        topo: str
            A test string argument
        foo: List[int]
            A test list of int argument
        mass: Optional[float]
            A test optional float argument
        bodyID: Optional[int]
            A test optional int argument
        """
        bodyID: Optional[int] = None

    ...
```

As we can see, to create the docstring for `TargetSpiceAssembly.Config` we had to repeat the entire `TargetAssembly.Config` docstring. This is tedious and can lead to inconsistent definitions when changes are made in some classes (but forgotten in others).



Instead, we can provide the description as a `metadata` entry:

```
class TargetAssembly(Assembly):

    @dataclass
    class Config:
        """Config of the TargetAssembly class.
        """
        topo: str = field(metadata={"description": "A test string argument"})
        foo: List[int] = field(metadata={"description": "A test list of int argument"})
        mass: Optional[float] = field(default=None, metadata={"description": "A test optional float argument"})

        ...

class TargetSpiceAssembly(TargetAssembly):

    @dataclass
    class Config(TargetAssembly.Config):
        """Config of the TargetSpiceAssembly class.
        """
        bodyID: Optional[int] = field(default=None, metadata={"description": "A test optional int argument"})

        ...
```

and then let a script write the `Parameters` part of the docstrings for us:

```
>>> def autogen_args_docstring(cls):
...     result = "Parameters\n-----\n"
...     for field in fields(cls):
...         result += f"{field.name}: TYPE" # TODO: get type
...         if field.default is MISSING and field.default_factory is MISSING:
...             result += ", optional"
...         result += f"\n    {field.metadata.get('description', '_desc_')}\n"
...     return result

>>> print(autogen_args_docstring(TargetSpiceAssembly.Config))
Parameters
-----
topo: TYPE, optional
    A test string argument
foo: TYPE, optional
    A test list of int argument
mass: TYPE
    A test optional float argument
bodyID: TYPE
    A test optional int argument
```

This script could be deployed automatically on `svn` (`git`) commit, or alternatively called manually by developers.

#### 2.5.3.1.5. Variable names

The issue of variable names may also find a solution through the `metadata` input.

Input dictionaries to Assemblies are filled with argument names with inconsistent naming conventions: we can find arguments in `UpperCamelCase` (`Target`), `lowerCamelCase` (`bodyID`), and `snake_case` (`frame_assembly`). Ideally, the user would be presented with a consistent convention. Given that python uses `snake_case` for variable and attribute names, I would argue that this convention should be retained.

However, the issue of backward compatibility remains. Even though we would like to rename an input to `"topo"`, the assembly code expects that input to be accessible in the dictionary as `"Topo"`. Moreover, other users have historically used `"Topo"` or used the constant `TOPO= "Topo"`. An `"alias"` entry to the `metadata` of a field could resolve this.

Consider the following implementation

```

from dataclasses import dataclass, asdict, fields, field
from typing import List, Union, Optional, Any

TOPO = "Topo"

class Assembly:

    @staticmethod
    def dataclass_to_dict(dataclass) -> dict[str, Any]:
        dict_ = asdict(dataclass)

        for field in fields(dataclass):
            for alias in field.metadata.get("alias", []):
                dict_[alias] = dict_[field.name]

        # None values are not included
        for key in list(dict_.keys()):
            if dict_[key] is None:
                dict_.pop(key)

        return dict_

class TargetAssembly(Assembly):

    @dataclass
    class Config:
        """Config dictionary of the TargetAssembly class.
        """
        topo: str = field(metadata={"alias": [TOPO]})
        mass: Optional[float] = field(default=None)
        foo: Optional[List[int]] = field(default=None)

    def __init__(
        self,
        config: Union[Config, dict[str, Any]],
    ) -> None:
        if isinstance(config, type(self).Config):
            config = Assembly.dataclass_to_dict(config)

        self.config = config

```

The field `topo` in `TargetAssembly.Config` has a new `metadata` entry: `"alias"`. This entry maps to a list of alias for the field `topo`. In this case, this only includes `TOPO`. The relevant functionality is implemented in `Assembly.dataclass_to_dict`. When the equivalent dictionary is generated, additional keys are populated that correspond to the alias. That way, `Assembly.dataclass_to_dict(config)["topo"] == Assembly.dataclass_to_dict(config)[TOPO]`:

```

OO_style_target = TargetAssembly(
    config = TargetAssembly.Config(
        topo = "test",
        foo = [1, 2]
    )
)

Data_style_target = TargetAssembly(
    config = {
        TOPO: "test",
        "foo": [1,2]
    }
)

```

```

>>> print(OO_style_target.config)
{'topo': 'test', 'foo': [1, 2], 'Topo': 'test'}
>>> print(Data_style_target.config)
{'Topo': 'test', 'foo': [1, 2]}

```

As we can see, even though we instantiated `TargetAssembly.Config` with the argument `topo`, the key `"TOPO"` is also available in the resulting dictionary.

My main concern with this feature is inconsistency. While users might be glad to use a consistent, pythonic naming conventions for the inputs, they might be confused when they see the same "thing" being referred to in two different ways.

#### 2.5.3.1.6. Comparison with legacy `BaseParam` classes

While similar, there are several differences between this proposal and the `Param-DataField` system:

- The `Param` system does not provide type-hints. This is because type-hints must be known statically, while it is not possible to ascertain the types corresponding to a `DataField` statically.
- The `Param` system enforces the types, while this system does not. This makes the `Param` system safer and more powerful, but it also makes it less flexible. `DataField`s must be of a specific type or have some specific length. The `Param` system struggles with nested structures and arrays of unknown length (this, however, makes it naturally fit with command-line interfaces). In contrast, the `config`, `context`, and `param` dictionaries never enforced their type and tolerated many different formats, including complex, nested structures.
- The `Param` system can perform sanity checks and provide defaults. This system does not do this. This system was built trying to be a discrete layer on top of the current architecture. It does not provide extra functionality, only a more user-friendly layer of paint. The only exception to this is that runtime errors will be raised when required fields are not provided, although this was considered harmless given that one could assume that similar errors would be raised either way down the line.

#### 2.5.3.1.7. More on `Param` class functionality

`Param` objects provide most of the functionality that we want with this refactor, as they substitute dictionaries for objects. The two following inputs are equivalent:

```
params = {
    MY_PARAM: MyParam(foo=2, bar="3")
}
```

```
params = {
    MY_PARAM: {"foo": 2, "bar": "3"}
}
```

However, there are certain features that would make Param objects more user-friendly, mainly related to type-hinting and docstrings.

One of the most important methods to update is the constructor. Current constructors take the following format:

```
def __init__(self, params= None, source= None, source_params= None, **kw):
    params = BaseParam.assignKeywordArgs(params, kw)
    BaseParam.__init__(self, params, source, source_params)
```

This is a very flexible approach, as any input arguments are captured with `**kw` and then inserted into the `params` dictionary. This, however, means that the IDE is not aware of what inputs to expect and their type.

#### 2.5.3.1.7.1. Option A

The most straightforward possible change involves writing each input explicitly:

```
def __init__(self,
    startingFuel: Optional[float] = None,
    startingCM: Optional[Vector3] = None,
    inertiaDataOrigin: Optional[Literal["BODY_ORIGIN", "CENTER_OF_MASS"]] = None,
    params: Optional[dict[str, Any]] = None,
    source: Optional[str] = None,
    source_params: Optional[dict[str, str]] = None,
    **kw,
):
    """ Docstrings """
    if startingFuel is not None:
        kw["startingFuel"] = startingFuel
    if startingCM is not None:
        kw["startingCM"] = startingCM
    ...

    params = BaseParam.assignKeywordArgs(params, kw)
    BaseParam.__init__(self, params, source, source_params)
```

This would replicate exactly the existing behavior.

An alternative syntax for the same concept is possible using the `locals()` dictionary:

```
...
""" Docstrings """
for param in self._params:
    if locals().get(param, None) is not None:
        kw[param] = locals()[param]

params = BaseParam.assignKeywordArgs(params, kw)
...
```

This syntax is shorter but slightly less readable.

A bad aspect of this implementation is that the defaults must be `None`, which is not representative of the actual behavior. Moreover, it is not clear which arguments are required and which are optional. Finally, the hints are bloated due to the inclusion of `Optional`.

It might be possible to set the defaults in `__init__` to be the same as the defaults in `_params`:

```
def __init__(self,
    startingFuel: Optional[float] = None,
    startingCM: Vector3 = [0,0,0],
    inertiaDataOrigin: Literal["BODY_ORIGIN", "CENTER_OF_MASS"] = "BODY_ORIGIN",
    params: Optional[dict[str, Any]] = None,
    source: Optional[str] = None,
    source_params: Optional[dict[str, str]] = None,
    **kw,
):
    """ Docstrings """
```

However, this would create opportunities for bugs due to mismatch between `__init__` and `_params`. If the default in `__init__` is 1 and the default in `_param` is 2, then `__init__`'s input will be taken as if it were a user's input, thus overriding the default behavior. Moreover, for required fields, a default value is still needed in `__init__` so that the method can be called without specifying it (as it can be specified in the `params` dictionary).

#### 2.5.3.1.7.2. Option B

This option involves tricking the IDE into thinking the `__init__` has a certain shape, while in reality, it does not:

```

def __init__(self, params= None, source= None, source_params= None, **kw):
    params = BaseParam.assignKeywordArgs(params, kw)

    BaseParam.__init__(self, params, source, source_params)

if TYPE_CHECKING:
    def __init__(self,
        startingFuel: float,
        startingCM: Vector3 = [0.0, 0.0, 0.0],
        inertiaDataOrigin: Literal["BODY_ORIGIN", "CENTER_OF_MASS"] = "BODY_ORIGIN",
        source: Optional[str] = None,
        source_params: Optional[dict[str, str]] = None,
    ):
        """Docstring"""
        pass # Implemented in __init__ above

```

In the above code, `TYPE_CHECKING` is a boolean variable defined in the `typing` module. This variable is designed to be read as `True` when the IDE analyzes it, but `False` at runtime.

This means that when the IDE tries to find how the `init` is defined, it will see that `TYPE_CHECKING` is `True`, and thus believe that the **second** definition of `init` is overriding the first, thus that the second definition is the final definition.

On the other hand, at runtime, since `TYPE_CHECKING` is `False`, the second definition of `init` is never reached, thus the **first** definition of `init` is used.

One of the main advantages of this option is that it is impossible for it to have any effect on current behavior; as the code is functionally equivalent at runtime. Mismatches between `init` and `_param` will only be visual.

Because the second `init` is never really used, we may declare it as we please. As we can see, `startingFuel` does not have a default value, which serves to indicate that this is a required field. Moreover, all types are reflected faithfully without the `Optional[]`. All defaults are also exactly as defined in the `DataField` object. Also, The `**kw` dictionary is not included in this definition. This will make the IDE highlight unnecessary inputs, which could be sources of error.

A bad aspect about this option is that what the IDE sees and the reality are not exact matches, which may be confusing for some users and future developers.

#### 2.5.3.1.8. Autogenerating init

Since all required information is declared in `_params`, it is possible to automate the creation of these `init` (both for Option A and B).

Option B would be slightly easier to automate since it only involves adding code and not modifying existing code. A script could run on `svn/git` commit or on user demand, which would append to the definition of the class something like:

```

...
# Autogenerated code: DO NOT TOUCH
if TYPE_CHECKING:
    def __init__(self,
        ...
    )
# End of autogenerated code

```

#### 2.5.3.1.9. Setting fields

In the spirit of making `Param` classes more object-like, instead of dictionary-like, we might consider supporting accessing the fields of the `Param` class as attributes. These two statements would be equivalent:

```
print(my_params["foo"])
```

```
print(my_params.foo)
```

Setting the value could also be supported.

##### 2.5.3.1.9.1. Option A

Uses properties to wrap the inner dictionary:

```

@property
def foo(self) -> float:
    """Lorem ipsum"""
    return self["foo"]

@foo.setter
def foo(self, value: float) -> None:
    self["foo"] = value

```

The best thing about this approach is that the attributes can be given docstrings describing them. With this approach, when you hover over `my_params.foo`, the type (`float`) and docstring (`Lorem ipsum`) will show.

The bad aspects of this approach are that the syntax is quite long and that it mismatches between `_param` and the properties may cause errors. For example, if `_param` includes a new field `bar`, but the property for this field has not been defined (for whatever reason), then users that attempt to call `my_params.bar` will cause an error.

##### 2.5.3.1.9.2. Option B

Involves implementing the `getattr` and `setattr` methods in `BaseParam`:

```

def __setattr__(self, k, v):
    if k in self._params:
        self[k] = v
    else:
        super(BaseParam, self).__setattr__(k, v)

def __getattr__(self, k):
    if k in self._params:
        return self[k]
    else:
        raise AttributeError

```

This will make it so calling `my_params["foo"]` is equivalent to `my_params.foo` as long as "foo" is a field in the params object. Note that `setattr` is already implemented in `BaseParam`, so the existing method would need to be modified to support this behavior.

In the subclass definition, these "fake" attributes should be declared:

```

class MyParam(BaseParam):
    foo: str
    bar: int

```

The best thing about this approach is that even if developers forget to add the declaration of the "fake" attribute, `my_params.foo` will always work as long as "foo" is a field. The IDE will complain about an unknown attribute, but at runtime everything will be fine. Moreover, this syntax is much shorter.

The worst thing about this approach is that it is not possible to attach a docstring to each variable (so hovering over the variable will not show `LoRe Ipsum`).

#### 2.5.4. DshellCommon: Replacing anonymous dictionaries with explicit classes



**TBD:** Needs scrubbing. Notes brought over from [issue](#)

(<https://dartsgitlab-internal.jpl.nasa.gov/darts-lab/jpl-internal/all-access/development/simulation-framework/dshellcommon/-/issues/21>).

See <https://dartsgitlab-internal.jpl.nasa.gov/darts-lab/jpl-internal/all-access/development/simulation-framework/dshellcommon/-/issues/25> for context on why it might be desirable to move away from using anonymous dictionaries (a dictionary that can take any input) as inputs to classes/functions.

The proposed approach is creating a class that can act as a regular dictionary but that adds an extra layer of information for the user. Developers would declare what entries they expect the dictionary to have and their type, and optionally provide a docstring explaining how each entry might be used.

This proposed class has been (tentatively) named `InputDict` to reflect that it should be used to encapsulate user inputs and the fact that it can work like a dictionary.

##### 2.5.4.1. Simple use case

Simplest example usage:

```

class Knight(InputDict):
    '''Represents a Knight with age and a title'''
    age: int = NO_DEFAULT
    title: str = NO_DEFAULT

object_knight = Knight()
dict_knight = dict()
assert object_knight == dict_knight

object_knight = Knight(age=23)
dict_knight = dict(age=23)
assert object_knight == dict_knight

object_knight = Knight(age=23, title="Sir")
dict_knight = dict(age=23, title="Sir")
assert object_knight == dict_knight

```

PYTHON

In the above code snippet, `object_knight` and `dict_knight` will behave exactly the same. One can perform any operation on `dict_knight` and `object_knight` with the same effect.

The advantage of `InputDict` is that it is explicit in how it should be populated, thus writing code that uses `Knight` instead of the anonymous `dict` will be clearer and less error-prone.

Not only `InputDict` can be manipulated as dictionaries, but also the fields can be accessed as attributes. This has the advantage that type-hints are retained and the syntax is shorter:

- `print(knight["age"])` is equivalent to `print(knight.age)`
- `knight["age"] = 20` is equivalent to `knight.age = 20`

New code should aim to use attribute access over dictionary access; type-hinting is preserved this way. Dictionary-like access is supported to permit using `InputDict` as a regular dictionary (see below on how `InputDict` can be integrated into existing code).

There are other advantages to using `InputDict`: default values and required fields can be declared:

```

class Knight(InputDict):
    '''Represents a Knight with age and a title'''
    age: int
    title: str = "Sir"

invalid_knight = Knight()
# ValueError: Missing required field: 'age'

knight = Knight(age=23)
assert knight.title == "Sir"

```

PYTHON

This syntax is shorter than checking for presence of required fields and providing defaults inside functions. It is also clearer to users and allows reusability.

#### 2.5.4.2. Using `InputDict` with existing code

The following is an example on how to transform old code into code that uses `InputDict` as inputs. Imagine we have a function that takes an anonymous dictionary:

```

def train(knight):

    if "age" not in knight:
        raise ValueError("knight must have 'age'")

    if "title" not in knight:
        knight["title"] = "Sir"

    ... # Do stuff

```

PYTHON

The new syntax would resemble:

```

class Knight:
    age: int
    title: str = "Sir"

def train(knight: Knight):

    ... # Do stuff

```

PYTHON

Note that the `... # Do stuff` is exactly the same in both old and new code: other than checking for the required fields and providing the default, `Knight` will behave exactly like a dictionary.

Now, it is much clearer what the `train` function needs as input. If we have more functions that take `Knight` as input, then we don't need to perform the sanity checks for every function.

If you want to support using both `Knight` objects and dictionaries as inputs (because other parts of the code already use `train` and they pass dictionaries) you can do:

```

def train(knight: Knight):

    knight = Knight(**knight)

    ... # Do stuff

```

PYTHON

This will cast the dictionary (or a `Knight`) to a `Knight` object. Aside from allowing the `train` function to manipulate `knight` as an object with attributes, this will also apply the defaults and required checks on the input. This is important because a dictionary and an `InputDict` may differ even when instantiated **if and only if** the `InputDict` model uses defaults/required fields (i.e. does not use `NO_DEFAULT` in all its fields):

```

class Knight(InputDict):
    '''Represents a Knight with age and a title'''
    age: int
    title: str = "Sir"

dict_knight = dict(age=23)
object_knight = Knight(age=23)

# `object_knight["title"]` is `Sir`, while `object_knight["title"]` throws a KeyError
assert object_knight != dict_knight

dict_knight = dict() # No error thrown
object_knight = Knight() # Throws an error due to the missing required input title

```

PYTHON

#### 2.5.4.3. Deleting fields and testing for presence

It is not uncommon for existing code to query the presence of an entry in an input dictionary:

```

def train(knight):

    if "title" in knight:
        old_title = knight["title"]
        del knight["title"]

    assert "title" not in knight

```

PYTHON

In the code above, we tested that the field "title" was present in the dictionary, and run some code depending on that. We also deleted the field, which would later make our assert statement pass. This deletion (*del*) and presence behavior (*contains*) are not commonly translated into object attributes. Instead, a valid approach would be:

```
def train(knight):

    if knight.title is not None:
        old_title = knight.title
        knight.title = None

    assert knight.title is None
```

PYTHON

Note that both behaviors are currently permitted with `InputDict`. If one wants to use an `InputDict` with existing code that uses constructs as in the first example (`del`, `in`), they would declare the `InputDict` as:

```
class Knight(InputDict):
    '''Represents a Knight with age and a title'''
    age: int
    title: str = NO_DEFAULT

knight = Knight(age=23)
assert "title" not in knight
```

PYTHON

where `NO_DEFAULT` is a sentinel value for `InputDict` that lets it know that that field should not be included in the dictionary if the user does not specify its value. Trying to access the field as an attribute (`knight.title`) will raise a `KeyError`, similar to dictionary-like access (`knight["title"]`).

New functions that use `InputDict` should avoid using `del` and `in`, and instead do things like comparing to `None` (as in the second example). They would declare the `InputDict` as:

```
class Knight(InputDict):
    '''Represents a Knight with age and a title'''
    age: int
    title: Optional[str] = None

knight = Knight(age=23)
assert knight.title is None
```

PYTHON

The main advantage of this approach is clarity. By explicitly declaring the type of `title` as `Optional[str]` (which is equivalent to `Union[str, None]`), we are signaling to users, other developers, and type-checking tools that `title` might be `None` at some point of its life, and proper care should be taken.

#### 2.5.4.4. Using `Dfield` to add information to fields

`Dfield` represents a field of an `InputDict` class. On class creation, each annotation is interpreted as a field, and a `Dfield` is generated to store its information. Users may provide additional information to a field by using the `Dfield` function.

The following two snippets are equivalent:

```
class Knight(InputDict):
    age: int
    title: str = "Sir"
    weapon: str = Dfield(default="sword")
```

PYTHON

```
class Knight(InputDict):
    age: int = Dfield()
    title: str = Dfield(default="Sir")
    weapon: str = Dfield(default="sword")
```

PYTHON

Each `Dfield` object stores certain information about the associated field. This includes:

- Information about the default value. This is specified either through `default` or `default_factory`, but not both. If neither is specified, the field is assumed to be required. If `default` is provided, then the default value will be a unique deep copy of `default` (this allows defaulting to a mutable type safe). Alternatively, if `default_factory` is specified, `default_factory` will be called (without arguments) and the return value will be used.
- Whether to skip type checking. If type checking is activated but `type_check` of a field is `False`, then that field will not be type-checked. If type checking is not activated in the class, `type_check` will not have any effect. See details on type checking in <https://dartsgitlab-internal.jpl.nasa.gov/darts-lab/jpl-internal/all-access/development/simulation-framework/dshellcommon/-/issues/22>.
- Alias of the field. TBD
- A human-readable `description` of the field. If it is not specified, `InputDict` will try to scrape the definition from the class docstring
- Metadata about the field. `metadata` is a dictionary that other users may populate as they please. `InputDict` does not use it.
- Validators for the value of the field. `validators` is a list of either a validator function or a `FieldCheck`, which will be called always after type checking. If any returns `False`, an error is raised. See details on validation in <https://dartsgitlab-internal.jpl.nasa.gov/darts-lab/jpl-internal/all-access/development/simulation-framework/dshellcommon/-/issues/22>.
- The type annotation of the field.

#### 2.5.4.5. Validation

All code shown in previous examples performs no validation of the inputs, other than raising an error if a required field is not given. While this behavior is desirable for existing code, which used anonymous dictionaries that also did not validate the data, newer code might move to perform validation within `InputDict` instead of within the functions that use them.

This behavior might be supported. A dedicated discussion is active on issue: <https://dartsgitlab-internal.jpl.nasa.gov/darts-lab/jpl-internal/all-access/development/simulation-framework/dshellcommon/-/issues/22>

#### 2.5.4.6. Implementation

Implementation details have been left out of this Issue for brevity. However, the source code for `InputDict` is attached to this issue. The code has been extensively documented. Still, if something is not clear about it, questions are encouraged. Recommendations on alternative ways to implement a feature are similarly welcome.

#### 2.5.4.7. Changelog

- Created issue. Uploaded `input_dict.py` v1.
- Added information about `DField` and `Dfield`. Uploaded `input_dict.py` v2 (expanded the documentation on `DField`).

[input\\_dict.py](#)

#### 2.5.5. DshellCommon: Use of InputDict with Assembly classes notes



**TBD:** Needs scrubbing. Notes brought over from [issue](#)

(<https://dartsgitlab-internal.jpl.nasa.gov/darts-lab/jpl-internal/all-access/development/simulation-framework/dshellcommon/-/issues/23>).

Issue <https://dartsgitlab-internal.jpl.nasa.gov/darts-lab/jpl-internal/all-access/development/simulation-framework/dshellcommon/-/issues/19> identified to need to make Assembly constructors clearer. The main issue with current constructors is that they have the same signature: they take three dictionaries `config`, `context`, `params`. What goes inside each dictionary is not shown, and users are left to review supporting documentation and test cases to find out how to use this class. This often leads to users copy-pasting inputs blindly without understanding how each input impacts the inner workings of the assembly, which can lead to insidious undesirable behavior.

Issue <https://dartsgitlab-internal.jpl.nasa.gov/darts-lab/jpl-internal/all-access/development/simulation-framework/dshellcommon/-/issues/21> proposes a new class that can act as a regular dictionary but that adds an extra layer of information for the user: Developers would declare what entries they expect the dictionary to have and their type, and optionally provide a docstring explaining how each entry might be used. More details are available in the issue.

This issue reflects a proposed use of `InputDict` to substitute the three anonymous dictionaries `config`, `context`, `params`. Through this change, we aim to allow developers to extensively document their Assemblies for future users. We would also like to maintain backward compatibility, so that:

1. New-style assemblies can be constructed with `InputDict`
2. New-style assemblies can be constructed with anonymous dictionaries
3. Old-style assemblies can be constructed with `InputDict`
4. Old-style assemblies can be constructed with anonymous dictionaries

For the purpose of this issue, "new-style" assemblies are those that use `InputDict` to substitute the three anonymous dictionaries `config`, `context`, `params`, while old-style assemblies are those that don't. Compatibility requirement 1 is met by design. Compatibility requirement 3 is met because `InputDict` is designed to be able to 100% mimic the functionality of anonymous dictionaries. See more details on "Adapting old assemblies to use `InputDict`" below.

##### 2.5.5.1. Changes to how Assembly classes are declared

Each assembly that wants to use `InputDict` as their input to `config`, `context`, `params` would need to declare up to three inner classes, `Config`, `Context`, and `Params`, that inherit from `InputDict`. Those inner classes that are not declared are autogenerated empty (if an Assembly does not need any `context` input, the `Context` class need not be declared). If none of these classes is implemented, the `Assembly` is assumed to be "old-style"; no changes are done to its functionality (which ensures that compatibility requirement 4 is met).

Developers that use `InputDict` in their `Assembly` class are free to use as many of the features of `InputDict` as they desire. The minimum implementation, which acts exactly like an anonymous dictionary, would have all their fields assigned the `NO_DEFAULT` sentinel value. The next step would be declaring required fields and giving default values to optional fields. Finally, type checking and complex validation could be used. See <https://dartsgitlab-internal.jpl.nasa.gov/darts-lab/jpl-internal/all-access/development/simulation-framework/dshellcommon/-/issues/21> and <https://dartsgitlab-internal.jpl.nasa.gov/darts-lab/jpl-internal/all-access/development/simulation-framework/dshellcommon/-/issues/22>.

In the following snippet, we are declaring an `Assembly` that takes inputs for `config` and `params`, while `context` should be empty.



```

class Fortress(Assembly):
    """A fortress is a defensive building that...

    This Assembly should be created when...

    See its usage in the following tests: ...
    """

    class Config(InputDict, type_check=True):
        """
        Parameters
        -----
        age : PositiveInt
            The age of this fortress. Is used in...
        has_moat : bool, optional
            Whether the fortress has protective..., by default False
        """
        age : PositiveInt = Dfield(validators=[lambda age: age < 500])
        has_moat: bool = False

    class Params(InputDict):
        """
        Parameters
        -----
        walls : WallsParam
            Characteristics of the walls of this fortress.
        """
        walls: WallsParam

    def __init__(self,
                 parent_asm: Assembly,
                 name: str,
                 config: Config,
                 context: "Fortress.Context",
                 signal_ties: Optional[dict] = None,
                 params: Optional[Params] = None,
                 description: str = "",
                 tag: str = "",
                 ) -> None:

        if config.has_moat:
            print(params.walls)

        assert 0 < config.age < 500

        Assembly.__init__(...)

```

In the `Config` inner class, we declare two fields: `age` (a required positive int that should be < 500), and `has_moat` (an optional boolean that defaults to False). Since `type_check=True`, the types will be enforced: `age` MUST be an integer > 0 and `has_moat` has to be a boolean, else a `TypeError` is raised when these classes are instantiated.

In contrast, `Params` simply declares a required field `walls` that SHOULD be `WallParam` object. Since no `type_check=True` is provided, no errors will be raised if the input `walls` is not a `WallParam`. However, an error will be raised if the `walls` field is not defined at all. Additionally, both `Config` and `Params` provide docstrings that describe each of their inputs.

Finally, the `__init__` has the same signature as always, with the exception of additional typing hints to signal users that they should use the specific `InputDict` as inputs. Note that these type-hints, like most type-hints, are not enforced. Also, note that since we are not defining `Context` in this class, but rather we inherit it from `Assembly` (empty), we need to use the forward type hint `"Fortress.Context"`.

The `Fortress` assembly will be instantiated as:

```

fortress = Fortress( ...
    config = Fortress.Config(age=2),
    context= Fortress.Context(),
    params = Fortress.Params(walls=WallsParam(...))
)

```

When a new user encounters the `Fortress` class for the first they will:

1. Read the docstring of the class, which should given them information about the purpose and general working. If they believe this is the right assembly for the job, then they will want to create it.
2. They will check out the `__init__` method, and see the type hints for `config`, `context`, and `params`. They will see that they have specific classes associated with them.
3. For each of the inner classes, they will inspect them. They will see what fields are required, which are optional, and their defaults. They will also read the docstring, hopefully describing how each field influences the final `Assembly` class. The types of each field are clear; and they can also see the special requirement for `Config.age` (that it must be < 500).

All the above can be accomplished by looking at the source of a class; it is almost as self-documenting as possible. Moreover, if their IDE/editor supports it, all this information will be displayed by hovering over the classes. Syntax highlight will also point out places where they are about to make a mistake (missing inputs, incorrect types...).

#### 2.5.5.1.1. Auto-population of class attributes

Note that `Fortress` has not declared any of the required class variables: `__requiredConfigFields`, `__optionalConfigFields`, `__requiredContextFields`, `__optionalContextFields`, `__requiredParamsFields`, `__optionalParamsFields`, or `__params_type`. This is because, on class creation, the `Config`, `Context`, and `Params` objects are scanned to automatically populate these attributes:

```

print(Fortress._requiredConfigFields)
# ['age']

print(Fortress._optionalConfigFields)
# ['has_moat']

print(Fortress._config_type)
# {'age': typing.Annotated[int, RangeCheck(0, inf)], 'has_moat': <class 'bool'>}

```

The `_config_type` and `_context_type` are additionally populated.

### 2.5.5.1.2. Inheritance

Assembly that inherit from other Assembly should have their `InputDict` classes also inherit from the parent Assembly class' `InputDict`. This way, fields are added to the parent `InputDict`. It is also possible to modify previous fields, although this should be done with care not to break any parent class behaviour

```

class Castle(Fortress):
    """A castle is a type of fortress with a noble ruling it"""

    class Config(Fortress.Config):
        """..."""
        age : PositiveInt = Dfield(validators=[lambda age: age < 200]) # Change existing field

    class Params(Fortress.Params):
        ruler: NobleParam # Add a new field

    def __init__(self, ...
        config: Config,
        context: "Castle.Context",
        params: Params,
        ...
    ) -> None:
        ...

```

### 2.5.5.2. Adapting old assemblies to use `InputDict`

#### 2.5.5.2.1. `InputDict` as a documentation tool

In the simplest use case for `InputDict`, we would like them to act exactly like anonymous dictionaries, with the exception that they can provide additional information to users (what fields to use, their types, and docstrings). To do so, the `NO_DEFAULT` value is used. The two following snippets are equivalent:

```

class Fortress(Assembly):

    class Config(InputDict):
        """
        Parameters
        -----
        age : PositiveInt
            The age of this fortress. Is used in...
        has_moat : bool, optional
            Whether the fortress has protective...
        """
        age : PositiveInt = NO_DEFAULT
        has_moat: bool = NO_DEFAULT

    def __init__(self,
        parent_asm: Assembly,
        name: str,
        config: Config,
        context: "Fortress.Context",
        signal_ties: Optional[dict] = None,
        params: Optional["Fortress.Params"] = None,
        description: str = "",
        tag: str = "",
    ) -> None:

        if "age" not in config or not isinstance(config["age"], int):
            raise ValueError

        if config["age"] <= 0 or config["age"] >= 500:
            raise ValueError

        if "has_moat" not in config:
            config["has_moat"] = False
        elif not isinstance(config["has_moat"], bool)
            raise ValueError

        if config["has_moat"]:
            print("Must learn to swim!")

        Assembly.__init__(...)

fortress = Fortress( ...
    config = Fortress.Config(age=2),
    ...
)

```

```

class Fortress(Assembly):
    _optionalRequiredFields = ["age"]
    _optionalConfigFields = ["has_moat"]

    def __init__(self,
                 parent_asm,
                 name,
                 config,
                 context,
                 signal_ties= None,
                 params= None,
                 description= "",
                 tag= "",
                 ) -> None:

        if "age" not in config or not isinstance(config["age"], int):
            raise ValueError

        if config["age"] <= 0 or config["age"] >= 500:
            raise ValueError

        if "has_moat" not in config:
            config["has_moat"] = False
        elif not isinstance(config["has_moat"], bool):
            raise ValueError

        if config["has_moat"]:
            print("Must learn to swim!")

        Assembly.__init__(...)

fortress = Fortress( ...
                    config = dict(age=2),
                    ...
                    )

```

The only advantage to the new syntax over the old syntax is the use of type hinting (which can be picked up by the IDE). The content of `init` or other operations to be done on `config` are exactly the same in both cases with the same end result. No changes to old code are necessary, besides including the declaration of `Config` and adding the type-hints to the `init`.

#### 2.5.5.2.2. Using InputDict features

Ideally, `InputDict` is not only used as a source of documentation + type-hinting, but also to enforce required fields, provide defaults, and validate the input. The code below is equivalent to the two snippets above:

```

class Fortress(Assembly):
    class Config(InputDict, type_check=True):
        """
        Parameters
        -----
        age : PositiveInt
            The age of this fortress. Is used in...
        has_moat : bool, optional
            Whether the fortress has protective...
        """
        age : PositiveInt
        has_moat: bool = False

    def __init__(self,
                 parent_asm: Assembly,
                 name: str,
                 config: Config,
                 context: "Fortress.Context",
                 signal_ties: Optional[dict] = None,
                 params: Optional["Fortress.Params"] = None,
                 description: str = "",
                 tag: str = "",
                 ) -> None:

        if config["has_moat"]:
            print("Must learn to swim!")

        Assembly.__init__(...)

fortress = Fortress( ...
                    config = Fortress.Config(age=2),
                    ...
                    )

```

As we can see, all input validation is left to the `InputDict`, while the code that uses the data remains the same (remember `InputDict` may be used as regular dictionaries). The snippet above, however, will only work with `InputDict` inputs for `Config`. This is because if we assign to `config` an anonymous dictionary `dict(age=2)`, the required fields, defaults, and validations will not be performed. In the case above, a key error will be raised since `"has_moat"` is not in `dict(age=2)`.

To solve this, a line can be introduced at the beginning of `init`, which will cast any anonymous dictionary into the `InputDict` object, thus triggering all required field checks, defaults, and validation:

```

def __init__(self,
             parent_asm: Assembly,
             name: str,
             config: Config,
             context: "Fortress.Context",
             signal_ties: Optional[dict] = None,
             params: Optional["Fortress.Params"] = None,
             description: str = "",
             tag: str = "",
             ) -> None:
    config, context, params = self.init_dicts(config, context, params)

    if config["has_moat"]:
        print("Must learn to swim!")

    Assembly.__init__(...)

```

After that line, `config`, `context`, and `params` will be of the respective `InputDict` types. This feature allows us to meet compatibility requirement 2: all new-style assemblies can accept both anonymous dictionaries and `InputDict` with the exact same effect.

### 2.5.5.3. Implementation

Implementation details have been left out of this Issue for brevity. However, the source code for the necessary changes to `Assembly` is attached to this issue. The code has been documented. Still, if something is not clear about it, questions are encouraged. Recommendations on alternative ways to implement a feature are similarly welcome.

### 2.5.5.4. Changelog

- Created issue. Uploaded `assembly.py` v1.

[assembly.py](#)

### 2.5.6. DshellCommon: Validation with InputDict



**TBD:** Needs scrubbing. Notes brought over from [issue](#)

(<https://dartsgitlab-internal.jpl.nasa.gov/darts-lab/jpl-internal/all-access/development/simulation-framework/dshellcommon/-/issues/22>).

Issue <https://dartsgitlab-internal.jpl.nasa.gov/darts-lab/jpl-internal/all-access/development/simulation-framework/dshellcommon/-/issues/21> covers the motivation for creating a class that helps encompass user inputs in a more concise manner than simple anonymous dictionaries.

The implementation discussed in that issue does not cover validation of the inputs, other than raising an error if a required field is not given. While this behavior is desirable for existing code, which used anonymous dictionaries that also did not validate the data, newer code might move to perform validation within `InputDict` instead of within the functions that use them.

Ideally, we would make this an opt-in feature. Users are familiar with using normal dictionaries for their inputs and therefore it makes sense for the simplest form of `InputDict` to also not perform input validation. We would like to encourage "plug-and-play", and then once they are comfortable using these classes they can begin to expand their functionality.

#### 2.5.6.1. Type checking

The simplest form of validation is ensuring that user inputs are of the correct type. The "correct" type is inferred from the class annotations. To turn on type checking in an `InputDict`, users may add `type_check=True` to the class declaration.

For example, the two following code snippets are equivalent:

```

def train(knight):
    assert isinstance(knight["age"], int)
    assert "title" not in knight or isinstance(knight["title"], str)
    ... # Do stuff

knight = dict(age=23, title=42)
train(knight)

```

```

class Knight(InputDict, type_check=True):
    age: int
    title: str = "Sir"

def train(knight):
    ... # Do stuff

knight = Knight(age=23, title=42)
train(knight)

```

As we can see, only by adding `type_check=True` we are ensuring that our inputs are all of the correct types. Moreover, `InputDict` can perform more advanced type checking than a simple `isinstance`. If we want to add a field to our input that should be a list of strings, then `InputDict` will be able to correctly type check values, while `isinstance(value, List[str])` would fail.

This works for the most commonly used type-hints:

```

class Knight(InputDict, type_check=True):
    age: typing.Union[int, float]
    title: typing.Literal["Sir", "Lord"] = "Sir"
    weapons: typing.List[str] = []

```

In the previous example, `age` can be either a float or an int. `title` must be either "Sir" or "Lord". `weapons` must be a list of strings. Nested collections are also valid: `typing.List[typing.List[str]]` will require a list of lists, each with only strings.

Type checking for specific fields might be skipped by adding `type_check=False` to the field definition:

```
class Knight(InputDict, type_check=True):
    age: int
    title: str = Dfield(default="Sir", type_check=False)
```

### 2.5.6.2. Validation through `FieldCheck`

`FieldCheck` is class representing a validation function to an `InputDict` field.

Subclasses of `FieldCheck` must always implement the abstract method `check`, which takes in a value and should return `True` if this value is "valid". The definition of a "valid" value is completely up to the definition of the implementer: a `FieldCheck` subclass might check that a number is greater than zero, while another might check that the value is a correctly-spelled Klingon word.

Subclasses may optionally implement `fail_message`, which should return a human-readable fail message for the check.

#### 2.5.6.2.1. Usage with `Annotated` to expand types

Python's `typing.Annotated` can be used to append metadata to type hints (see <https://peps.python.org/pep-0593/>). This can be leveraged to make type checks perform additional validation.

When `InputDict` performs a type check, it finds if the type is `Annotated`. If so, it will first perform a regular type check with the type that is being "annotated". It will then scan its arguments (the metadata). If any of these objects are of the type `FieldCheck` it will run the check. If it fails, it will raise a `TypeError`:

Let's consider validating that an input is a positive integer. We might create `RangeCheck` that ensures a value is between two bounds, and then `Annotate` the `int` type:

```
class RangeCheck(FieldCheck):
    def __init__(self, lower: float, upper: float):
        self.lower = lower
        self.upper = upper

    def check(self, value) -> bool:
        return self.lower < value < self.upper

class Knight(InputDict, type_check=True):
    age: Annotated[int, RangeCheck(0, np.inf)]
    title: str = "Sir"

Knight(age=-1)
# TypeError: Type of field age is incorrect; Value -1 failed check RangeCheck
```

In order to prevent clutter and increase reusability, a type alias can be used:

```
PositiveInt = Annotated[int, RangeCheck(0, np.inf)]
class Knight(InputDict, type_check=True):
    age: PositiveInt
    title: str = "Sir"
```

Now, `PositiveInt` can be used in any `InputDict` that requires an integer greater than zero.

`Annotated` types, much like regular types, can be arbitrarily combined. For example, a list of positive multiples of three might be declared as:

```
PositiveInt = Annotated[int, RangeCheck(0, np.inf)]
MultipleOf3 = Annotated[PositiveInt, MultipleOfCheck(3)]
ListMultipleOf3 = List[MultipleOf3]
```

Possible `FieldCheck` of interest in DARTS could be:

- `RangeCheck`: tests whether a float is between two values. (`PositiveFloat = Annotated[float, RangeCheck(gt=0)]`)
- `QuantityCheck`: tests whether a float has units of the specified quantity. (`Area = Annotated[float, QuantityCheck("Area")]`)
- `PathCheck`: to check whether strings are valid files/directories. (`FilePath = Annotated[str, PathCheck(exists=True, dir_okay=False)]`)
- `SizeCheck`: to check the size of arrays/matrices. (`Vector3 = Union[SOAVector3, Annotated[Iterable[float], QuantityCheck(3)]]`)

It would be possible to populate a module with commonly used type alias so that users can import them and simply use them on their `InputDict`: `PositiveInt`, `Length`, `Mass`, `InertiaTensor`, `SpiceFile`...

#### 2.5.6.2.2. Usage in `InputDict` fields

Sometimes, we would like to validate a field without having to change its type hint. This is especially true for very specific validations that may only happen once. For example, the type `DimensionlessIntHigherThan16` might be a bit overkill.

In this case, we can use the `validators` input in the field declaration function `Dfield`:

```
class Knight(InputDict, type_check=True):
    age: int = Dfield(validators=[RangeCheck(gt=16), QuantityCheck("Dimensionless")])
    title: str = "Sir"
```

Alternatively, validation functions may be input directly, which should have the signature `(Any) -> bool` (take one input and return True or False):

```
higher_than_16_fun = lambda value: return value > 16

class Knight(InputDict, type_check=True):
    age: int = Dfield(validators=[higher_than_16_fun, QuantityCheck("Dimensionless")])
    title: str = "Sir"
```

Field validator functions are always run, independently of whether type checking is activated or not. If type checking is performed, then validator functions will run after the type check, and in the order that they were given in `validators`. This means that one can trust that `age` will be an integer by the time `higher_than_16_fun` is run.

### 2.5.6.3. Class-wide validation

`InputDict` subclasses may also implement the `validate` method, which will be called after the type checks, and can be used to perform additional checks:

```
class Knight(InputDict, type_check=True):
    age: int
    title: str = "Sir"

    def validate(self):
        if age < 40 and title == "Lord":
            raise ValueError
```

This is useful to perform validation that depends on the content of multiple fields. This method is not expected to return anything, but it should raise an Error if values are not in compliance with the desired constraints.

This method is called once all value assignments have been performed, when all type checks have passed, and after field-specific validators have passed.

Possible use of this method is to alter the values stored in the object. For example, one may use one default or another depending on how other fields were set. If behavior like this is implemented, the user is advised to clearly document it, as it can be unintuitive.

The method `pre_type_check` can also be implemented and follows a similar purpose to `validate`, except that it is called before type checking occurs and field-specific validators are run. Much like the `validate` method, this could be used to generate defaults that depend on other parameters. In this case, the generated default will go through type checking and field-specific validators, which might be desirable behavior.

### 2.5.6.4. Implementation

Implementation details have been left out of this Issue for brevity. However, the source code for `FieldCheck` is attached to this issue. The source for `InputDict` is available in issue <https://dartsgitlab-internal.jpl.nasa.gov/darts-lab/jpl-internal/all-access/development/simulation-framework/dshellcommon/-/issues/21>. The code has been extensively documented. Still, if something is not clear about it, questions are encouraged. Recommendations on alternative ways to implement a feature are similarly welcome.

### 2.5.6.5. Changelog

- Created issue. Uploaded `field_check.py` v1.

[field\\_check.py](#)

## 2.6. Sphinx documentation

### 2.6.1. Getting Started

#### 2.6.1.1. Introduction

##### 2.6.1.1.1. Introduction

This document is intended to give the user an overview of DshellCommon and provide examples of using DshellCommon modules to quickly build a simulation and assemble various models. Prior to continuing with this guide, it is suggested that the user familiarize themselves with Darts and Dshell (see User's Guides for Darts and Dshell).

##### 2.6.1.1.2. What is DshellCommon?

DshellCommon can be thought of as a "user interface" to Darts and Dshell whose function is to facilitate the assembly of desired models to build a simulation. As the name implies, DshellCommon is designed to serve as the foundation from which all future simulations are built upon, whether modeling spacecraft or rovers. DshellCommon consists of a suite of python modules and we'll summarize each of them in the following sections; for a more detailed description of each module, see the DshellCommon reference documentation.

##### 2.6.1.2. Module Description

The python modules that make up DshellCommon can be found in the `src` directory under `DshellCommon/python`. At the top level of this directory there are three modules, namely, `SimulationExecutive.py`, `SimulationAssembly.py`, and `BaseAssembly.py`. As you might have guessed, these modules also perform "top level" functions that all lower level DshellCommon modules depend upon.

Also in this top level directory are four subdirectories, namely, `assemblies`, `fsm`, `params`, and `utils`.

### assemblies

The `assemblies` directory in general (but not always) contains a single python module for each model to be simulated. These can be simulation executive models such as "Mission Elapsed Time", hardware models such as IMUs, or environment models such as gravity. In most cases the assemblies function to "connect" desired models that make up the simulation. By "connect" we are referring to tasks required access to the underlying compiled C++ files whose code actually contains the programming for the specific model. In addition to bringing in the desired model, tasks include managing the models inputs and outputs (flowIn's and flowOut's), and performing the required parameter initialization. As eluded to earlier; not always does a single python assembly module necessarily map directly to a single C++ based module. An engine assembly, for example, may perform the tasks related to configure an engine consisting of several models such as nozzles, actuators, etc. The idea though is to avoid having the "kitchen sink" approach where complex logic is required in the assembly to perform the required tasks of many different models. Instead, assemblies will be named according the function they provide and may inherit from other assemblies. Indeed, all assemblies inherit from at least the top level assembly `BaseAssembly.py`.

## fsm

The fsm directory contains the "Finite State Machine" related modules. These modules contain the mechanisms to "control" the simulation at runtime. These controls include the capability to execute user specified functions at virtually any user specified event.

## params

The params directory contains the modules responsible for initializing the required parameters in the assemblies.

## utils

Finally, the utils directory currently contains the doctestutils.py module.

The following sections describe the individual DshellCommon modules. We will be referring to `DshellCommon-RunScripts` as well as the individual module test scripts that are found under `DshellCommon/test`.

### 2.6.1.3. Basic DshellCommon Assembly Classes

#### 2.6.1.3.1. SimulationExecutive

The `SimulationExecutive.py` module is the "top most" module that contains class `SimulationExecutive` that provides all `DshellCommon` executive functions as well as the those available through its base class `DshellX`. As shown in Section 2 in the example user run script, the `SimulationExecutive` "sim" instance is where all `DshellCommon` objects "live". In this simple example, the user can see that only three "sim" methods are required prior to executing the simulation. The `SimulationExecutive` class also manages the FSM (Finite State Machine(s)). A top level state machine is created by default, but the `SimulationExecutive` can actually manage a FSM for each assembly if desired by the user. The `SimulationExecutive` also initializes the root frames, steps the sim, etc. It is suggested the user browse the test directories to get a feel for the rich set of capabilities available through the `SimulationExecutive`.

#### 2.6.1.3.2. SimulationAssembly

The `SimulationAssembly.py` module contains class `SimulationAssembly`. A single instance of this class is used in the `SimulationExecutive` as the default top level assembly to house all assemblies that exist in the simulation.

#### 2.6.1.3.3. BaseAssembly

The `BaseAssembly.py` module contains the `DshellCommon` `BaseAssembly` class that all assemblies use as their base. The `BaseAssembly` class in turn has the `Dshell` `Assembly` as its base. A somewhat new concept in `DshellCommon` is that all assemblies can have an optional Finite State Machine that can be configured by the user to control the simulation.

### 2.6.1.4. Useful DshellCommon Assembly Classes

#### 2.6.1.4.1. TargetAssembly

#### 2.6.1.4.2. VehicleAssembly

#### 2.6.1.4.3. ExternalDisturbanceActuatorAssembly

#### 2.6.1.4.4. Gravity Assemblies

#### 2.6.1.4.5. Time Assemblies

#### 2.6.1.4.6. IMU Assemblies

### 2.6.1.5. Targets

In the DARTS/Dshell framework, planetary bodies are called 'targets'. This terminology is largely historical due to the dominant focus in the past of landing spacecraft at specific locations (targets) on planetary bodies. We may eventually migrate to a more generic terminology, but for now please understand that a 'Target' in a simulation refers to any planetary body, whether they are the planned location of a spacecraft landing or not.

Targets have two primary roles in DARTS/Dshell simulations:

- Serve as a center of gravitational attraction for nearby spacecraft and
- Provide a way to create planetary bodies that have physical/visual representations and provide position and velocity information for other purposes, such as point-to-point communication analysis, day-night cycle analysis,

Note that we do not hard-code gravity models into Targets since each spacecraft might need different types of gravity model for its own operational requirements. So gravity models are separate but typically use Targets in order to get the relative position from the spacecraft to the Target's location so the gravity model can compute its gravitational contributions.

There are two primary types of Targets that are most relevant to most simulations: plain Target (created with `TargetAssembly`) and Spice Targets (created with `TargetSpiceAssembly`). Spice is software tool kit provided by JPL to help track the positions and velocities of planetary bodies (see <http://naif.jpl.nasa.gov/naif/toolkit.html> for more details). These two types of Targets are explained in more detail below.

In general, Targets create two bodies: (1) A PCI body (Planet Centered Inertial) that may (or may not) translate to track the actual motion of the corresponding planetary body, and (2) a PCR body (Planet Centered Rotating) that rotates as the Target generally rotations but translates with the corresponding PCI body. These two bodies are actually DARTS bodies for a variety of reasons including the ability to attach other bodies (such as spacecraft or rovers) to them.

#### 2.6.1.5.1. TargetAssembly

A plain target (created with `TargetAssembly`) represents a very simple concept: the PCR body rotations about the PCI body about one axis at a prescribed rotational rate. In general, the PCI body does not move and is usually coincident with the "inertial frame" of the simulation. Spice is NOT used to control its motion.

Here is what the description of a plain Target in a run script might look like<sup>[1]</sup>:

```
'Mars' : {  
  'class' : 'TargetAssembly',  
  'params' : { 'Target' : targets['Mars'],  
  }  
}
```

PYTHON

The "Target" parameter given will define the Target's name, radius, mass, and rotational rate. The "Target" parameter will usually be instances of the parameter class TargetBaseParam.

The joint between the PCI and PCR bodies in this case will always be a pin joint with the rotational axes defaulting to the Z axis.

Note that no 'Bodies' parameter item is given in the 'params' block. The TargetAssembly defaults to creating very simplistic PCI and PCR bodies. In most cases, the PCI body does not move and the PCR body rotates about the PCI body with a fixed rotational rate. The motions of both are prescribed and therefore the mass properties of the PCI and PCR bodies are irrelevant. It is permitted to add 'Bodies' parameter item, but it should extend the expected body parameters as specified in the function TargetBaseAssembly.defaultTargetBodyParams()---in particular, the joint between the PCI and PCR bodies must be a pin joint unless the run script takes responsibility for setting up the initial positions and rotational rates.

In this case, no initial position or attitude information parameters are provided so the PCI and PCR will initially both be coincident with the "inertial frame" of the simulation. If an 'Attitude' parameter specified in the 'params' block, it will be used to set the initial rotational angle of the PCR body with respect to its PCI body. Attitude parameter classes are usually derived from the StateAttitudeBaseParam class and support a variety of ways to initialize the angle of PCR with respect to PCI.

Targets created using TargetAssembly add a PinEncoder model which provides an angle signal for the 1-DOF rotation of the PCR body with respect to the PCI body.

#### 2.6.1.5.2. TargetSpiceAssembly

In most simulations, it is very desirable for the Targets to accurately track the motion of their corresponding physical planetary bodies. The Spice toolkit provides the software tools and libraries to do that. In these cases, users will want to use instances of the 'TargetSpiceAssembly' in their simulations.

For instance, to create a Spice Target for Mars, the description in the user's run script might look like this<sup>[2]</sup>:

```
'SpiceMars' : {  
  'class' : 'TargetSpiceAssembly',  
  'params' : { 'Target' : targets['SpiceMars'],  
              'Epoch' : states['MSL 05-19']['Epoch'],  
              }  
},
```

In this case, the 'Target' parameter item is an instance of the 'TargetSpiceParam' parameter class and primarily includes the name of the planetary body, its Spice ID (e.g. 499 for Mars), and a list of Spice kernels that will need to be loaded in order for Spice to have the appropriate data for its computations.

There is also an 'Epoch' state parameter that essentially only provides the exact epoch (date and time) for the Spice library to use for the initial date and time at the start of the simulation.

Spice Targets should NOT specify a 'Bodies' parameter; that is handled internally by the TargetSpiceAssembly class.

Spice Targets use a ball joint between the PCI and PCR bodies in order to accommodate the arbitrary rotations that Spice may require.

When Spice targets are initialized in the bindParams() phase, two things are done: (1) the SpiceTargetAssembly loads the kernels specified in the 'Target' parameter and then (2) sets the Spice epoch using the date and time specified in the 'Epoch' parameter. Then the parent class, TargetBaseAssembly bindParams() is called which uses body parameters specified in TargetSpiceAssembly.defaultTargetBodyParams which sets up the ball joint between the PCI and PCR bodies and token mass and inertial properties.

For Spice Targets, the full motion of the PCI body is controlled by Spice using two models SpiceFramePCIBodySync and SpiceFramePCRBBodySync that move the bodies to match the motions prescribed by Spice during the simulation (in preDeriv).

Spice Targets created using TargetSpiceAssembly add a BallEncoder model which provides three signals for the three rotational angles of the ball joint encoder as well as a full quaternion for the rotation between the PCI and PCR bodies.

#### 2.6.1.5.3. Topographic Objects

It is possible to attach topographic objects such as Digital Elevation Maps (DEM) or topographic mesh objects on to Targets by adding an additional config item. For instance to attach an analytic DEM object onto a target:

```
from DshellCommon.configs.TopoAnalyticDemConfig  
  
asm_info = {  
  ...  
  
  'SpiceMars' : {  
    'class' : 'TargetSpiceAssembly',  
    'topos' : {'zero' : TopoAnalyticDemConfig(topo='zero')},  
    'params' : { 'Target' : targets['SpiceMars'],  
                'Epoch' : states['MSL 05-19']['Epoch'] }  
  },  
  
  ...  
}
```

more complex types of analytic dems can be specified by filling out more of the TopoAnalyticDemConfig object (see DshellCommon/python/configs/TopoAnalyticDemConfig.py or TopoAnalyticDemConfig).

Multiple topographic entries can be attached to a target by adding more items to the 'topos' config item dictionary. The keys will be the names of the topographic objects.

Targets can host topographic objects created by specifying TopoStoreConfig ( TopoStoreConfig ) entries (for topographic objects that are saved in SimScape 'Store' objects).

There are also a set of "known" topographic objects such as 'SphericalSun' or 'EarthWGS84' which can be created by using the class 'TopoKnownConfig' ( TopoKnownConfig ). Note these are not attached to the parent Target since they are stand-alone bodies. They can be created this way for convenience purposes.

For usage examples, please see the ???.

#### 2.6.1.5.4. KnownTargetAssembly



Most use cases require well-known targets of the Solar System. For convenience, DARTS provides a set of pre-built TargetSpiceAssembly corresponding to all planets, the Sun, and several satellites. For each of these known targets, a dedicated Assembly class and TargetParam is defined. Both the Assembly and TargetParam classes are loaded with defaults corresponding to known constants of each body. All targets have their mass, gravitational constant, radius (or equatorial and polar radius), rotation rate, and NAIF ID defined. Some targets have Spherical Harmonic Coefficient data (for gravity modelling). Some targets have topology associated with them, but if a corresponding KnownTopo exists.

To create a Target Assembly corresponding to Mars, one might do: .. code-block:: python

```
'Mars' : {\n    'class' : 'MarsTargetAssembly', 'params' : {\n        'Epoch' : states['MSL 05-19']['Epoch'],\n    },\n}
```

or by directly calling the python initializer: .. code-block

```
python from DshellCommon.assemblies.known_targets.MarsTargetAssembly import MarsTargetAssembly\n\nMarsTargetAssembly(parent_asm, params={\n    'Epoch': states['MSL 05-19']['Epoch']\n})
```

If one wants to override one or more parameters while preserving the other defaults, the body-specific TargetParam may be used. Let's consider that we wish to change Mars' mass to 6e23 kilograms. One would do: .. code-block:: python from DshellCommon.assemblies.known\_targets.MarsTargetAssembly import MarsTargetAssembly from DshellCommon.assemblies.known\_targets.MarsTargetAssembly import MarsTargetParam

```
MarsTargetAssembly(\n    parent_asm, params={\n        'Epoch': states['MSL 05-19']['Epoch']\n        'Target': MarsTargetParam(mass=6e23)\n    })
```

Overriding the topology is also simple, as one simply has to pass the config dictionary: .. code-block:: python from DshellCommon.assemblies.known\_targets.MarsTargetAssembly import MarsTargetAssembly

```
MarsTargetAssembly(\n    parent_asm, config={\n        'topos' : {\n            'zero' : TopoAnalyticDemConfig(topo='zero')\n        }\n    }, params={\n        'Epoch': states['MSL 05-19']['Epoch']\n    })
```

Currently, the following known targets are available: \* CallistoTargetAssembly \* DeimosTargetAssembly \* EarthTargetAssembly (KnownTopos ID: EarthWGS84, Spherical Harmonics: GGM03C) \* EnceladusTargetAssembly (KnownTopos ID: SphericalEnceladus) \* EuropaTargetAssembly (KnownTopos ID: SphericalEuropa) \* GanymedeTargetAssembly \* IoTargetAssembly \* JupiterTargetAssembly (KnownTopos ID: Jupiter, Spherical Harmonics: jupiter) \* MarsTargetAssembly (KnownTopos ID: SphericalMars, Spherical Harmonics: MRO120F) \* MercuryTargetAssembly (Spherical Harmonics: MESS160A) \* MoonTargetAssembly (KnownTopos ID: SphericalMoon, Spherical Harmonics: GL0900D) \* NeptuneTargetAssembly (KnownTopos ID: Neptune, Spherical Harmonics: neptune) \* PhobosTargetAssembly (KnownTopos ID: SphericalPhobos) \* PlutoTargetAssembly \* SaturnTargetAssembly (Spherical Harmonics: saturn) \* SunTargetAssembly (KnownTopos ID: SphericalSun) \* TitanTargetAssembly (KnownTopos ID: Titan) \* UranusTargetAssembly (Spherical Harmonics: uranus) \* VenusTargetAssembly (KnownTopos ID: Venus, Spherical Harmonics: MGNP180U)

## Footnotes

### 2.6.1.6. Parameters

Add tutorial material or merge with params in reference. In the mean time:

See [DshellCommonReference](#)

### 2.6.1.7. Data Logging through Dstore

Dstore is a mechanism for logging in DshellCommon and uses the HDF5 pytables module. Dstore includes most of the logging features available in Dwatch, but adds HDF5, CSV, and Report output format as well as unit conversions capability and other controls.

To configure Dstore for specific logging requirements, Dstore reads a Python ConfigObj configuration file which contains the user log controls. The controls are also available at run time through the Dstore object.

Below are the contents of the Dstore ConfigObj file that was automatically rendered through Dstore function writeDefaultCfg(). The file contains commented section that also should help describe available specifications. The .cfg extension stands for a 3rd party Python extension ConfigObj. ConfigObj as used here is a convenient way to specify user input that Python can easily interpret.

We'll begin by dissecting the file into sections. Each section begins with the keywords in brackets. Python stores all specification following the bracketed keyword in a Python dictionary with that keyword as the key, until another bracketed keyword is encountered, which is interpreted as a nested dictionary. This means you have to be somewhat careful when moving things around or adding content.

Finally, there may be mention of Phasing in the following descriptions. Phasing is a DshellCommon mechanism of controlling the simulation in a similar fashion as the FSM but also has a built in printmanager which handles when to log. For more information see Phasing getting started section.

#### 2.6.1.7.1. Default CFG File

```

# that allows section and subsection specification. Sections are
# defined in brackets. Subsection in double brackets. The sections
# name "Parameters", "Includes", or starts with "Format_" are
# predefined. Others sections are user defined groups to log.

# The first section is "Parameters" which takes the top level logging
# specifications described below.

[Parameters]

# states - This is either a state or phase name or a list of states
# to be logged. If states = None (or not specified), then all states
# are recorded.

states = None

# startTime - This is the time in which to start logging. Group logging
# specification can modify startTime for its group but can not be
# earlier than the top level startTime specified in this section.
# If startTime is not specified logging begins immediately

startTime = 0.0

# timeInterval - This specifies the time interval in which to log.
# If timeInterval is NOT specified, then all IO times steps are
# recorded beginning after startTime. If None, only record beginning
# or end of states if specified in groups.

timeInterval = 0.005

# strictInterval - If strictInterval is True, log
# only on exact intervals of startTime + timeInterval.

strictInterval = False

# timeResolution - Floating point delta compare for strictInterval
# log test

timeResolution = 0.0001

# select - This specifies which output groups to log. If not
# specified or an empty list will log all groups, even included groups

select = []

# formats - This specifies the output formats. Supported formats
# are listed below.
# 1. 'hdf5' - This is default output
# 2. 'report' - Text output in customizable format
# 3. 'csv' - Comma Separated Value format and flavors thereof
# Note: An hdf5 will always be created. It will be in addition to the
# 'report' or 'csv' files that are created

formats = ['hdf5', 'report', 'csv']

# The Includes section is used to merge other '.cfg' files.
# The keywords in this section can be any unique string and are not used.
# Dstore ignores the "Parameters" and "Includes" sections in included files.
# Example:
# 1 = "./traj_aero.cfg"

[Includes]

# The following sections that begin with "Format_" contain special code

# Format_report - This section specifies report attributes

[Format_report]

# If filename is not specified, use the cfg filename with '.rpt'. To print
# to standard output, specify filename 'stdout'

filename = traj.rpt

# source - This indicates what states to log. onState is based on time.

source = ['onState', 'onEnterState']

# startTime - This controls the time when the report should start. Defaults to
# group or main startTimes if None

startTime = None

# stopTime - This controls the time when the report should stop. Ignored if None.

stopTime = None

# freq - Frequency of the onState to log. Cannot be more frequent than what is
# specified by group or top level timeInterval. Every record if None.

freq = None

# reportChar - The character used to format the report.

```

```

reportChar = -

# nameLength - Space allocated for variable names. Set to longest name for
# optimal alignment

nameLength = 15

# decimal - Set to True to force the floating point format g to always print a decimal.

decimal = False

# precision - Floating point precision

precision = 8

# ncolumns - The number of columns in the report

ncolumns = 3

# header - Contains variable source, units, description

header = True

# UTC - This correspond to a list or tuple containing the Universal Time:
#     Base yr, mon, day, hr, min, sec, microsec as integers, seconds from epoch as double
#     Can also be a variable in the namespace the has the vaule of the tuple.

UTC = None    # 'utc_ref'

# MET - This corresponds to a list or tuple containing Mission Elapsed Time:
#     days, hours, minute, second, microseconds as integers
#     Can also be the variable in the Dstore namespace that contains the tuple.

MET = None    # 'met_ref'

# Format_hdf5 - This section specifies default hdf5 attributes
[Format_hdf5]

# If no filename is specified, the cfg filename is used with the extension replaced with '.h5'

filename = traj.h5

# Description is stored in the HDF5 file.

description = File contains ISS Model and Trajectory Output

# Format_csv - This section specifies default csv attributes
[Format_csv]

# source - specifies where the history comes from: onEnterState, onState, onExitState, or any
# If includeStates is True, the FSM state names are included

source = ['onState', 'onEnterState']

# If no filename is specified, the cfg filename is used with the extension replaced with csv

filename = traj.csv

# startTime - This controls the time when the CSV should start. Defaults to
# group or main startTimes if None

startTime = None

# stopTime - This controls the time when the CSV should stop. Ignored if None.

stopTime = None

# freq - Frequency of the onState to log. Cannot be more frequent than what is
# specified by group or top level timeInterval. Every record if None.

freq = None

# comments - optional file comments

comments = Trajectory Data

# header - If True print the vaiable name header

header = True

# units - If True print the units

units = False

# colwidth - Column width spacing

colwidth = 12

# decimal - Set to True to force the floating point format g to always print a decimal.

decimal = True

# precision - Floating point precision

precision = 8

```

```

# Separates columns. Should be a single character.

delimiter = ,

# includeStates - If true print the state or phase names

includeStates = False

#
# The final sections contain groups of desired output parameters. Each group name should appear in
# single brackets and be followed by desired output parameter specification as shown below. A double
# bracketed GroupSpec may appear after all variable declarations for that group. GroupSpec is used to
# further control logging at the group level.
#
#[Group_name]
#
#   varname   = ["source", "OutputUnits", "PostProcClass", "Description", "InternalUnits", "VectorSuffix"]
#
# # GroupSpec - This is optional and has to be last in the configobj groups to conform with that
# # API. GroupSpec further manages logging at the group level.
#   [[GroupSpec]]
#     description = 'LAS' # will appear report header and H5 group description
#     timeInterval = 0.025 # log frequency for this group
#     startTime = 0.0 # start time for this group
#     vectorSuffix = '012M' # Name sequences of len 4 this Default XYZM
#     onEnterState = True # Log enter events for group
#     onExitState = False # Log exit events for group
#     stateCharLen = 50 # Size allotted for the state (phase) name in HDF5 file (default 40)
#

```

### 2.6.1.7.2. CFG File Section Descriptions

The Dstore cfg file may contain one or more user defined output groups to be logged. The groups may be defined in a given cfg file or may be included from other cfg files. This is important to keep in mind as Dstore has log controls at the top level as well as the group level. The Parameters section contains the top level log controls for the HDF5 output file. At the group level, each group may have further log specification for that particular group. Each group corresponds to a separate HDF5 table in the HDF5 file. Finally, both the CSV and Report options have their own specific log controls.

#### 2.6.1.7.2.1. [Parameters]

This section specifies the logging frequency, group selection, and file format type specification for desired output. The logging frequency is global for all output and can be further restricted at the group level. Note that the opposite is NOT true in that groups groups cannot be directed to log more frequent than the specification in this section. Group selection can pertain to groups defined within this cfg file or other cfg files included in as explained in the following section. In a similar fashion, CSV or Reports can not be generated more frequent than the specific group is logged.

#### 2.6.1.7.2.2. [Includes]

This section is used to include output specification of groups from other .cfg files. The [Parameters], [Includes], and [Format\_] sections of included files are ignored. The key in this section can be any unique value as the only used information is the filename value.

#### 2.6.1.7.2.3. [Format\_FILETYPE]

This section is used to specify attributes of the desired FILETYPE.

- **FILETYPE = 'hdf5'**

This filetype is written by default as a Python HDF5 pytables. If multiple groups are specified, each group is written as a separate table, so there is a correlation between Dstore Groups and HDF5 groups.

- **filename** - Output filename. Will take on the cfg filename with a .h5 basename if not specified.
- **description** - File description.

- Common to \***FILETYPE = 'csv'** and '**report'**

- **source** - Can equal onState, onEnterState, onExitState, or any combination thereof. This corresponds to the Dstore log method used to store the data.
- **startTime** - Time to start producing output
- **stopTime** - Time to end output.
- **freq** - Output frequency for onState if specified. This has no effect on onEnterState, or onExitState.
- **header** - If True, print variables name header
- **precision** - Desired number of significant digits. Default is 6.

- **FILETYPE = 'csv'**

- **filename** - This filetype will append the group name to the specified filename and a .csv'. If not specified, will take on the .cfg basename.
- **comments** - This will be the 1st line in the file.
- **units** - If True, print units name header:
- **colwidth** - Column width.
- **decimal** - If True force the floating point g format to always print a decimal.
- **delimiter** - Field separator character.
- **includeStates** - If True include the FSM states in the output.

- **FILETYPE = 'report'**

- **filename** - If not specified, will take on the .cfg basename and ".rpt".
- **reportChar** - Character used to format the report. '.' is default.

- **nameLength** - Space allocated for the group item name. 12 is default.
  - **ncolumns** - Number of columns. Default is 3.
  - **MET** - Optional MET time banner is output when specified (Mission Elapsed Time). Must be set to the desired reference MET as a sequence of days, hours, minute, second, microseconds as integers. MET = 0, 0, 0, 0, 0
- MET can also be set to a variable in the namespace that contains the above tuple. For example, if the variable in the namespace is equal to met, the following can be specified. MET = met
- **UTC** - Optional UTC time banner is output when specified (Coordinated Universal Time). Must be set to the desired reference UTC as a sequence of year, month, day, hour, minute, second, microseconds as integers, and the reference UTC time as a float. The UTC reference is user defined and may be set according to the mission such as the Greenwich Hour Angle. UTC = 2010, 7, 26, 0, 0, 0, 0.0
- UTC can also be set to a variable in the namespace that contains the above tuple. For example, if the variable in the namespace is equal to utc, the following can be specified. MET = met

#### 2.6.1.7.2.4. [GROUPNAME]

This section is used to specify groups that contain desired output parameters. In this example, there are 2 groups [group1] and [group2], but [group2] is not output. Also, [group3] is selected for output and exists in the include.cfg file.

- **select** - This is a list of single quoted variables to be output for the group. If not specified, all variables in the group are output.
- **VARIABLE\_DEFS** - This is one or more variable definitions taking on the form below described below.
  - name = ['source', 'units', 'postproc', 'description', 'internal\_units', 'vector\_suffix', 'aliases']
    - **name** - Variable name as to appear in output.
    - **source** - Internal object containing the value. This can be a string corresponding to a specNode, an expression to be evaluated, or a callable object. If this object is a string that has to eval'd each log time, expect an increase in run time. A callable object (like a specNode) is stored at initialization and does not have to be eval'd each log time which reduces run time.
    - **units** - Desired output units
    - **postproc** - Class to "postprocess" the internal value at each write time point. This is computationally slow and it is suggested to perform this type of processing after simulation termination if at all possible.
    - **description** - Variable description. This is typically a short description as in plot axis labels.
    - **internal\_units** - This is used to allow specification for internal units if they can not be retrieved internally. Specifying this field **will** take precedence and ignore any available internal units (a debug warning would be posted for this case).
    - **vector\_suffix** - This is a string containing letters used as a suffix to **name**. This specification will override the default group specification.
    - **aliases** - This is a list of string variable name aliases in which to name the variable in output. This field overrides the vector\_suffix field and the name. Aliases must be of same length as the source object value.
- - This section contains optional group output controls that allow the following specification. To abide by the ConfigObj rules, this must appear after all variable declarations for the particular group.
  - **description** - Group description. This appears in the hdf5 group table and in the report output.
  - **startTime** - Time to start logging the specified group.
  - **timeInterval** - Group log time interval. This specification can not log more frequent than that specified in the **[Parameters]** section. It is used to reduce the frequency.
  - **vectorSuffix** - String containing letters used as a suffix for sequences. This can be overwritten for each output item.
  - **onEnterState** - If set to False, the onEnterState is not logged.
  - **onExitState** - If set to False, the onEnterState is not logged.
  - **stateCharLen** - Size allotted for the state name in HDF5 file (40 is default)

#### 2.6.1.7.3. The Namespace Object

To create a Dstore instance, a namespace is generally provided to contain references to all the objects specified in the .cfg file. Here a module containing objects referenced in the cfg file is used, but any object that has an *dict* attribute can be used. The following is from test\_Dstore.

```

import math, sys
from Math.SOA_Py import SOAVector3, SOAQuaternion, SOAEuler
from DshellCommon.utils.units import unitConvert
from DshellCommon.SimulationExecutive import SimulationExecutive
sim = SimulationExecutive.simulationLookup()

class Container:pass
chaser = Container()
target = Container()
chaser.assembly = sim.assembly('ChaserVehicle', 0, True)
target.assembly = sim.assembly('TargetVehicle', 0, True)

class Position:
    def __init__(self, body_obj):
        self.body_obj = body_obj
    def __call__(self):
        return self.body_obj().positionInertialFrame()()

class Velocity:
    def __init__(self, body_obj):
        self.body_obj = body_obj
    def __call__(self):
        return self.body_obj().velocityInertialFrame()[3:]

class Magnitude:
    def __init__(self, vec_obj):
        self.vec_obj = vec_obj
    def __call__(self):
        return SOAVector3(self.vec_obj()).magnitude()

chaser.pos = Position(chaser.assembly.rootbody)
chaser.vel = Velocity(chaser.assembly.rootbody)
chaser.posmag = Magnitude(chaser.pos)
chaser.velmag = Magnitude(chaser.vel)
grav = chaser.assembly.findAssembly('gravity')
grav_model = grav.model('gravity')
chaser.grav_spec = grav_model.specNode()

target.pos = Position(target.assembly.rootbody)
target.vel = Velocity(target.assembly.rootbody)
target.posmag = Magnitude(target.pos)
target.velmag = Magnitude(target.vel)
grav = target.assembly.findAssembly('gravity')
grav_model = grav.model('gravity')
target.grav_spec = grav_model.specNode()

from numpy import array
class TargetPos(object):
    """Example of how to customize name output and specify unit conversion. When
    units_conversion is specified the units in the cfg are for display only. """
    def __init__(self):
        self.obj = Position(target.assembly.rootbody)
        self.table_names = ['TPCIX', 'TPCIy', 'TPCIZ']
        self.unit_conversion = array([0.3048, 0.3048, 0.3048])

    def __call__(self):
        return self.obj()

class TargetVel(object):
    """Example of how to customize name output and specify unit conversion. When
    units_conversion is specified the units in the cfg are for display only. This
    example shows output_units and from_units are for display only"""
    def __init__(self):
        self.obj = Velocity(target.assembly.rootbody)
        self.table_names = ['TVPCIX', 'TVPCIy', 'TVCIZ']
        self.unit_conversion = array([1.0, 1.0, 1.0])
        self.output_units = 'ft/s'
        self.from_units = 'm/s'

    def __call__(self):
        return self.obj()

```

#### 2.6.1.7.4. Dstore Specifications

The following sections detail the API of Dstore. Most options available in the `cfg` input file are also available at run time.

##### 2.6.1.7.4.1. Dstore Module Attributes

- **csv** - csv object (see csv attributes)
- **report** - report object (see report attributes)

##### 2.6.1.7.4.2. Dstore Module Functions

- **writeDefaultCfgO** - This writes a default CFG file.

##### 2.6.1.7.4.3. Constructor Call

```

dstore_obj = Dstore.Dstore(cfg_file=None,
                           namespace=None,
                           output_dir='./')

* **cfg_file** - None

  If no `cfg` file is provided, the default settings are used. These are the same settings that exist
  in the cfg file produced by writeDefaultCfg(). Updating this value after construction has no effect.

* **namespace** - None

  This is a module or class container containing the names needed by the user defined output items.

* **output_dir** - './'

  This is the location of the output. Updating this value after construction, currently has no effect.

```

#### 2.6.1.7.4.4. Dstore Object Attributes

See CFG description for attribute descriptions.

- **states** - See cfg description
- **startTime** - See cfg description
- **timeInterval** - See cfg description
- **strictInterval** - See cfg description
- **timeResolution** - See cfg description
- **select** - See cfg description
- **format** - See cfg description
- **outputDir** - See cfg description
- **h5f** - This is the Dstore HDF5 file object. The same object when reading an Dstore HDF5 file.

#### 2.6.1.7.4.5. Dstore Object Methods

- **addGroup(group\_name, group\_dict, namespace)** -
- **keys()** - Return a list of group names. To retrieve a group object use::: group\_obj = dstore\_obj[group\_name]
- **renderCfg(filename=None)** - Write out a cfg file with the current Dstore state. If no filename is given, use "[input\_cfg]rendered.cfg".
- **logOnState(time, state)** - Log the state according to the startTime, timeInterval, timeResolution
- **logOnEnterState(time, state)** - Log the record when entering a state.
- **logOnExitState(time, state)** - Log the record when exiting a state.
- **onStateHistory** - Return the states recorded from logOnState
- **onEnterStateHistory** - Return the states recording for logOnEnterState
- **onExitStateHistory** - Return the states recording for logOnExitState
- **values()** - Return the list of Dstore group objects.

#### 2.6.1.7.4.6. Dstore Group Object Attributes

See CFG description for attribute descriptions.

- **timeInterval**
- **startTime**
- **description**
- **onEnterState**
- **onExitState**
- **vectorSuffix**
- **stateCharLen**
- **data** - The group HDF5 data table.
- **header** - The group HDF5 header table.
- **name** - The group name.

#### 2.6.1.7.4.7. Dstore Group Object Methods

- **addItem(item\_name, item\_attributes, namespace)** - Add items to the group. Update must be called after all:: items have been added to the group.  
**item\_name** - The variable name as in the CFG file. **item\_attributes** - List of variable attributes as in the CFG file. **\*\* namespace** - The namespace containing the item source definition.
- **keys()** - List of item names in the group.
- **update()** - Method to update HDF5 table and when new items have been added.
- **values()** - Return the list of Dstore group item objects.

#### 2.6.1.7.4.8. Dstore Item Object Attributes

- **description** - Variable name description.

- **name** - Variable name
- **units** - Output units

#### 2.6.1.7.4.9. Dstore Item Object Methods

- **rawvalue** - Returns the item value without unit conversions.
- **value** - Returns the item value with unit conversion applied.

#### 2.6.1.7.4.10. Dstore csv Object Attributes

See the CFG descriptions.

- **colwidth**
- **comments**
- **decimal**
- **delimiter**
- **freq**
- **header**
- **includeStates**
- **precision**
- **startTime**
- **stopTime**
- **source**
- **units**

#### 2.6.1.7.4.11. Dstore report Object Attributes

See the CFG descriptions.

- **decimal**
- **freq**
- **header**
- **nameLength**
- **ncolumns**
- **precision**
- **reportChar**
- **startTime**
- **stopTime**
- **source**

#### 2.6.1.7.5. Usage Examples

##### 2.6.1.7.5.1. Print the Dstore Object

This will produce the information of the Dstore object that is also printed as the header of the traj.rpt.orig output and is shown below. Important to note that if there are no internal units, no unit conversion is performed. It can be seen below for DstoreGroup "chaser" that "ri" and "grav\_accel" will have unit conversions performed since both "Internal" and "Output" units are available. The DstoreGroup "target" shows one other way to specify unit conversions and output variable names. Note that the TargetPos() and TargetVel() are class instances defined in module namespace.py. This approach shows how to create callable class objects to customize output. This approach requires the unit\_conversion attribute value to be initialized as the cfg string units are NOT used to derive the unit conversion value. Here the unit string values are tags that take precedence over the cfg values.

Also, note that the tables specifies a \* for items that are callable. This is an indicator that run time performance is optimal. Note that variable "noncallable" source is not a callable object. This means Dstore performs an "eval" every time to log the value. A callable object is much faster than the "eval" approach.



```
DstoreGroup: chaser
Number of variables: 6
Description: Chaser Vehicle
```

| Variable<br>(Callable=*) | Units    |        | Source                                | Description        |
|--------------------------|----------|--------|---------------------------------------|--------------------|
|                          | Internal | Output |                                       |                    |
| mass *                   | kg       | lbm    | chaser.assembly.mass                  | Mass               |
| rimag *                  | None     | m/s    | chaser.posmag                         | Velocity Magnitude |
| vimag *                  | None     | m/s    | chaser.velmag                         | Velocity Magnitude |
| grav_accel *             | ft/s**2  | m/s**2 | chaser.grav_spec['flowOut.lin_accel'] | Grav Accel         |
| ri *                     | m        | ft     | chaser.pos                            | Position           |
| vi *                     | None     | m/s    | chaser.vel                            | Velocity           |

```
-----
```

```
DstoreGroup: target
Number of variables: 8
Description: Target Vehicle
```

| Variable<br>(Callable=*) | Units    |        | Source                                | Description        |
|--------------------------|----------|--------|---------------------------------------|--------------------|
|                          | Internal | Output |                                       |                    |
| mass *                   | None     |        | target.assembly.mass                  | Mass               |
| rimag *                  | None     | m/s    | target.posmag                         | Velocity Magnitude |
| vimag *                  | None     | m/s    | target.velmag                         | Velocity Magnitude |
| grav_accel *             | None     |        | target.grav_spec['flowOut.lin_accel'] | Grav Accel         |
| ri *                     | None     | m      | TargetPos()                           | Position           |
| vi *                     | m/s      | ft/s   | TargetVel()                           | Velocity           |
| vi2 *                    | None     | m/s    | target.vel                            | Velocity           |
| noncallable              | None     | m/s    | 9.81                                  | Grav constant      |

```
-----
```

#### 2.6.1.7.6. Using the Dstore methods to perform the logging

Below is a class that can be used by the FSM to log the desired dstore object.

```
class log_onState:
    def __init__(self, dstore_obj, state):
        self.dstore_obj = dstore_obj
        self.state_name = state
    def __call__(self):
        self.dstore_obj.onState(sim.time(), self.state_name)

states = [
    ['Initialize', [], [log_onState('Initialize', dstore_traj)], []],
    ['State1', [], [log_onState('State1', dstore_traj)], []]
]
```

PYTHON

#### 2.6.1.7.7. Using the Dstore methods to modify logging at run time

Below is a function that is executed as a transition function to change the group timeInterval to 30. sec. Note that this is the group timeInterval for "chaser" and NOT the file timeInterval.

```
def decrease_chaser_log():
    dstore_traj['chaser'].timeInterval = 30.0
    print("Reducing chaser log rate to %4.1f" % dstore_traj['chaser'].timeInterval)
```

PYTHON

#### 2.6.1.7.8. Adding groups at run time

The following is an example of adding a new DstoreGroup "aero" at runtime. The method addGroup takes as arguments the group name, a dictionary of group items, and the namespace.

```
dstore_traj.addGroup('aero',
    {'cm_aero_CA': ["get_aero_data()['cat']", "", None, 'CM Axial Aero Coefficient'],
     'cm_aero_CD': ["get_aero_data()['cdt']", "", None, 'CM Drag Aero Coefficient'],
     'cm_aero_CL': ["get_aero_data()['clt']", "", None, 'CM Lift Aero Coefficient']},
    namespace)
```

PYTHON

#### 2.6.1.7.9. Adding items at run time

The following is an example of adding DstoreItems to a DstoreGroup at runtime. After adding items the updatePrintGroup with the specific group must be executed.

```
for name, attr in (
    #
    # Name      Value Source      Output Units      Description
    ('cm_aero_FA', ["get_aero_data()['fa']", "", None, 'CM Axial Aero Force']),
    ('cm_aero_FN', ["get_aero_data()['fn']", "", None, 'CM Normal Aero Force']),
    ('cm_aero_FX', ["get_aero_data()['fx']", "", None, 'CM X Aero Force']),
    ('cm_aero_FY', ["get_aero_data()['fy']", "", None, 'CM Y Aero Force']),
    ('cm_aero_FZ', ["get_aero_data()['fz']", "", None, 'CM Z Aero Force'])
):
    cm_group.addItem(name, attr, namespace) # Each item has it's own namespace
# Need to refresh the DstoreGroup
dstore_traj.updatePrintGroup(cm_group)
```

PYTHON

#### 2.6.1.7.10. Query internal HDF5 table at run time

```

data = dstore_traj['chaser'].data
print("\nQuery example: Note: recType 3 contains onState records")
recs = data.where('(recType == 3) & (time > 40.) & (vimag > 7804.) & (rimag < 6589069.)')
for rec in recs:
    print('Chaser, time: %8.4f, State: %10s, VI: %10.4f, RI: %10.4f' % (rec['time'], rec['state'], rec['vimag'], rec['rimag']))

```

Will produce the following (see test\_Dstore).

```

Query example: Note: recType 3 contains onState records
Chaser, time: 170.0000, State: 3rd State, VI: 7811.6048, RI: 6589068.6863
Chaser, time: 200.0000, State: 3rd State, VI: 7813.9678, RI: 6586967.9710

```

### 2.6.1.7.11. Closing Output Files

At the end of the simulation, the output files can be closed if desired although Python will close all output before it ends the process.

```

dstore_traj.closeOutput()
dstore_event.closeOutput()

```

### 2.6.1.8. Finite State Machine

The description that follows will reference the code snippet below. In this example an FSM is created from the ISS assembly. In fact, the DshellCommon FSM allows each assembly to have its own FSM.

```

from DshellCommon.FsmUtils import uponStateEntry, generalTrigger
iss = sim.findassembly('ISS')
iss.createFsm(sim)
fsm = iss.fsm()
states = [
    # STATE_NAME, ENTRY_ACTION, DURING_ACTION, EXIT_ACTION
    ['Setup', None, None, None],
    ['Coast', [uponStateEntry, enter_store], None, None],
    ['LVLH Hold', [uponStateEntry, enter_store, cmd_hold], None, None],
    ['End', [uponStateEntry, enter_store, sim.terminate], None, None],
]
transitions = [
    # FROM_STATE, TO_STATE, CHECK_FUNCTION, ACTION_FUNCTIONS
    ['Setup', 'Coast', True, []],
    ['Coast', 'LVLH Hold', generalTrigger(sim.time, 'value >= 587.0'), []],
    ['LVLH Hold', 'End', generalTrigger(sim.time, 'value >= 2500.0'), []],
]
fsm.parseStates(states)
fsm.parseTransitions(transitions)
fsm.debugOff()
fsm.transition_info = sim.time
fsm.setCurrentState('Setup')

```

In general, a simulation might use the default FSM (the default is the SimulationAssembly fsm) which can be referenced from the SimulationExecutive instance, which is sim in this example. This could be done as follows.

```
fsm = sim.fsm()
```

#### 2.6.1.8.1. State Specification

States are the mechanism to specify controls to place on the simulation or to just distinguish an event that is desired to have logged. Here states is a list of lists. Each sublist contains the information of a given state described in order below.

- 1. State Name** - The 1st item in the list is the state name or title.
- 2. On Enter State Function (ENTER\_ACTION)** - The second item in the list is a function reference (or a list of function references) to be executive when transitioning into that state. Here we use a list containing uponStateEntry (is used to echo the simulation time and the state) and our user defined enter\_store function. None can be used when no function is specified.
- 3. On State Function (DURING\_ACTION)** - The 3rd item is a function reference (or list of function references) to be executed each I/O step while in that state (or None).
- 4. On Exit State Function (EXIT\_ACTION)** - The last item is a function reference to be executed when transitioning out of this state, and yes you guessed it, it can be a list of function references too or None if no action is desired.

#### 2.6.1.8.2. Transition Specification

Next we specify the information directing the sim to "transition" from one state to another. We do this through the transitions list, which like states, is a list of lists. Here each sublist contains information for a single transition described in order below.

- 1. From State** - This is a state that we specified in the states list and is the state we plan to transition out of.
- 2. To State** - This is the state that we specified in the states list that we want to transition into.
- 3. Monitor Function (CHECK\_FUNCTION)** - The monitor/check function is a reference to a function which when returns True will direct the simulation to transition from the From state to the To state. Here we are using the utility generalTrigger available in FsmUtils which is a callable class. Here we use the class approach to "store internally" the trigger information, namely, the variable to monitor; and the condition in which to trigger the transition.
- 4. Transition Function (ACTION FUNCTIONS)** - This is a reference to a function (or a list of function references) to execute if the monitor function returns True. The transition function will execute after executing On State functions then On Exit functions specified with the given From State and before the To State On Enter functions.

Note in the example that the fsm.transition\_info is set to the simulation time (the default is the number of steps that have been taken). Setting to time might be useful when printing the fsm condition so that the time is tagged to the the time the state occurred.

### 2.6.1.9. Controlling the Simulation with Phasing

Phasing is a module similar to the Finite State Machine used to halt the simulation through phases and issue user commands (phases are analogous to FSM states). Phasing sequences can be created at the assembly level for a particular vehicle, and there is a main phasing sequence at the simulation level. Phasing has Fixed, Free, Floating, and Optional type phases. Free phases can trigger any time after the previous Fixed phase. Optional phases are like Free phases, except that they are no longer monitored after the next Fixed phase occurs. Floating phases are similar to Optional phases, in that they are bounded by two Fixed phases, and that they may occur any time relative to Free and Optional phases that are being monitored. However; the Floating phase must occur before the next Fixed phase may occur. Phasing also has a RegulaFalsi type mechanism which alters the integration step size to hit the event within a specified tolerance. This is useful if you need to execute some type of control at exactly some time point or an exact point in the trajectory such as altitude as shown in the example below. The Step Manager module, which is related to Phasing, contains execution and step size management functions.

#### 2.6.1.9.1. Creating a Phasing / Step Manager Instance

The first step in getting phasing set-up is importing the PhasingManager class and the optional PhaseTime class. PhaseTime is a mechanism for monitoring time spent in a current event and triggering the phase when reaching that value. When a phase 'triggers', it is invoked and its command actions are performed. The command actions can be a single command or a list of commands.

The phasing instance is used to manage the user defined phases, as well as providing print management.

```
from DshellCommon.fsm.phasing import PhasingManager, PhaseTime
from DshellCommon.fsm.StepManager import StepManager
phasing = PhasingManager(sim)
manager = StepManager(sim)
```

PYTHON

#### 2.6.1.9.2. Initialize Desired Phases

The example below shows 3 phases being initialized. Phases require a number which is used by phasetype This can be provided in the phase specification as shown below or if not provided the PhaseManager will increment phase numbers by 10 when added through addPhase. Phases may not trigger in numerical order. For example, in the example below, phase 20 is a Free phase. Free phases can occur at any time point after the previous (number) Fixed phase. In the case below, phase 20 may trigger after phase 30 if the altitude is not reached prior to 1000 seconds.

```
phasing.addPhase(number=1,
                 name='Coast',
                 monitor=PhaseTime,
                 value=100.0,
                 phasetype='Fixed',
                 slope='GTE',
                 cmd=None)

phasing.addPhase(number=20,
                 name='Max Time',
                 monitor=sim.time,
                 value=1000.0,
                 phasetype='Free',
                 slope='GTE',
                 priority=50,
                 cmd=sim.terminate)

phasing.addPhase(number=30,
                 name='Max Time',
                 monitor=altitude,
                 value=400000.0,
                 phasetype='Free',
                 slope='GTE',
                 priority=50,
                 tolerance=10.0,
                 tolerance_type='Absolute',
                 monitor_type='RegulaFalsi',
                 cmd=sim.terminate)
```

PYTHON

The following are allowable phase specification. Those with default values are optional.

- 1. number** - All phases require a number. If not specified the addLog will set the number 10 higher than the current highest number. The number is how the sim manages what phases have to be monitored, or in other words, what candidate future phases can trigger. This is not to be confused with the actual phase trigger order.
- 2. name** - The name or title of the phase.
- 3. monitor** - This is the attribute that we want the sim to monitor to determine if it is time to phase. Monitor can be a simple function reference that returns the value as in the case with "sim.time" in phase 20 or a python callable class object "PhaseTime" as is the case with phase 1.
- 4. value** - This is simply the value or the target we're trying to hit with the monitor attribute. This can be a simple scalar or a callable function. When the monitor function reaches the target value, the phase triggers and then executes the specified cmd[s].
- 5. phasetype** - (Default = "Fixed") Currently we have four phase types, namely, "Fixed", "Free", "Optional", and "Floating". A "Fixed" phase has to occur before the next "Fixed" phase triggers". A "Free" phase can occur any time after the prior "Fixed" phase. Optional phases are like Free phases, except that they are no longer monitored after the next Fixed phase occurs. Floating phases are like Optional phases, in that they are bounded by two Fixed phases, and that they may occur any time relative to Free and Optional phases that are being monitored. However; the Floating phase must occur before the next Fixed phase may occur.
- 6. slope** - (Default = "ASC\_GTE")The slope determines how the value is approached. The value "GTE" means Greater Than or Equal". This means if the variable altitude is monitored with "GTE" it will trip as soon as the monitor value is greater or equal to the value. The value "CID" means Change in Direction, which monitors a variable for a sign flip in the first derivative.

**Valid "Trip" Options:** 'LT', 'GT', 'LTE', 'GTE', 'DES\_LT', 'DES\_LTE', 'ASC\_GT', 'ASC\_GTE', 'CID', and 'EQ'.

**Valid "Regula-Falsi" Options:** 'ASC' or 'DES'.

The "DES" stands for "descending" such that the monitor is decreasing to the target. Conversely, "ASC" defines that the parameter must be increasing to the target.

- 7. cmd** - (Default = None) This is the function reference (or list of function references) you want to execute (or callable class instance) when the phase triggers.

8. **priority** - (Default = 100) This tells the sim which phase to trigger if more than one phase is scheduled to trigger at a given time. Higher priority phases first.
9. **monitortype** - (Default = "Trip") This directs the sim how to hit the target value. Two options are available, "Trip" which means trigger whenever the threshold is passed. The second option "RegulaFalsi" instructs the sim to change the integration step size in order to hit the target within a specified tolerance. After hitting the tolerance, the integration stepsize is returned to the value prior to monitoring.
10. **tolerance** - (Default = 0.1) Tolerance used to determine if the monitor value is within the target value. This parameter is only used for Regula-Falsi and is not used during Trip monitortype. It means how close to the target value must the monitor value before the Regula-Falsi event is considered to be met.
11. **tolerance\_type** - (Default = "Percent") "Absolute" and "Percent" are options to determine how option tolerance is interpreted. This parameter is only used for Regula-Falsi and is not used during Trip monitortype.
12. **relaxation\_factor** - (Default = 0.85) This parameter is only used for option Regula-Falsi to slow down convergence in meeting monitor parameters target. Smaller value will have the affect of smaller integration step sizes in trying to achieve the target. Helpful for very dynamic parameters.

### 2.6.1.9.3. Phasing Print Manager

Phasing has a print manager which facilitates data logging. The print manager is a wrapper around the DshellCommon Dstore function. See Dstore getting started section. Note that the obj returned from the addLog method is a Dstore instance. Printing the object displays some useful log specification information. Also, the namespace is described in the Dstore documentation.

```
phasing.printmanager.setOutputDir('./results')
phasing.printmanager.log_start_of_phase = True

namespace = phasing.printmanager.namespace
namespace.vehicle1 = sim.assembly('Vehicle1', 0, True)
namespace.vehicle2 = sim.assembly('Vehicle2', 0, True)
for cfg_file in ('dwatch/traj.cfg', 'dwatch/event.cfg'):
    obj = phasing.printmanager.addLog(cfg_file, namespace)
```

PYTHON

The following are printmanager features.

1. **setOutputDir** - Method to set directory for output. The files themselves are defined in the Dstore .cfg files.
2. **log\_start\_of\_phase** - Log data at each phase trigger time.
3. **addLog** - Add the specified Dstore .cfg file and the corresponding namespace.

### 2.6.1.9.4. Execution

Phasing & Step Manager provide the following execution related items.

```
phasing.echo_freq = -1
print 'Initial Phases:\n', phasing
## # Begin Execution
print 'Begin Execution'
manager.execute()
phasing.printTriggeredPhases()
print 'Remaining phases:\n', phasing
```

PYTHON

1. **echo\_freq** - Echo to stdout sim time and the current phase.
2. **execute** - Execute the simulation until a sim.terminate is encountered.
3. **printTriggeredPhases** - Print a listing of phases that triggered as well as how well they achieved their target, time of trigger etc.
4. **printmanager.generateOutput** - Actually a print manager function to generate the output files.

### 2.6.1.9.5. Interactive Execution

Interactively invoking the simulation in the python shell, or better yet the ipython shell is useful for trajectory analysis, debugging, or general introspection. Step Manager currently has three mechanisms for stepping the sim described below.

Take a single step or multiple.

```
manager.step(nsteps=1)
```

PYTHON

Step to the specified phase. In the following example the sim will step and trigger all candidate phases until it reaches phase 30 (or until the sim terminates).

```
manager.steptoPhase(phasing_manager, 30)
```

PYTHON

or similarly step to a desired time. Here phases may trigger before halting at the desired time.

```
manager.steptoTime(10.0)
```

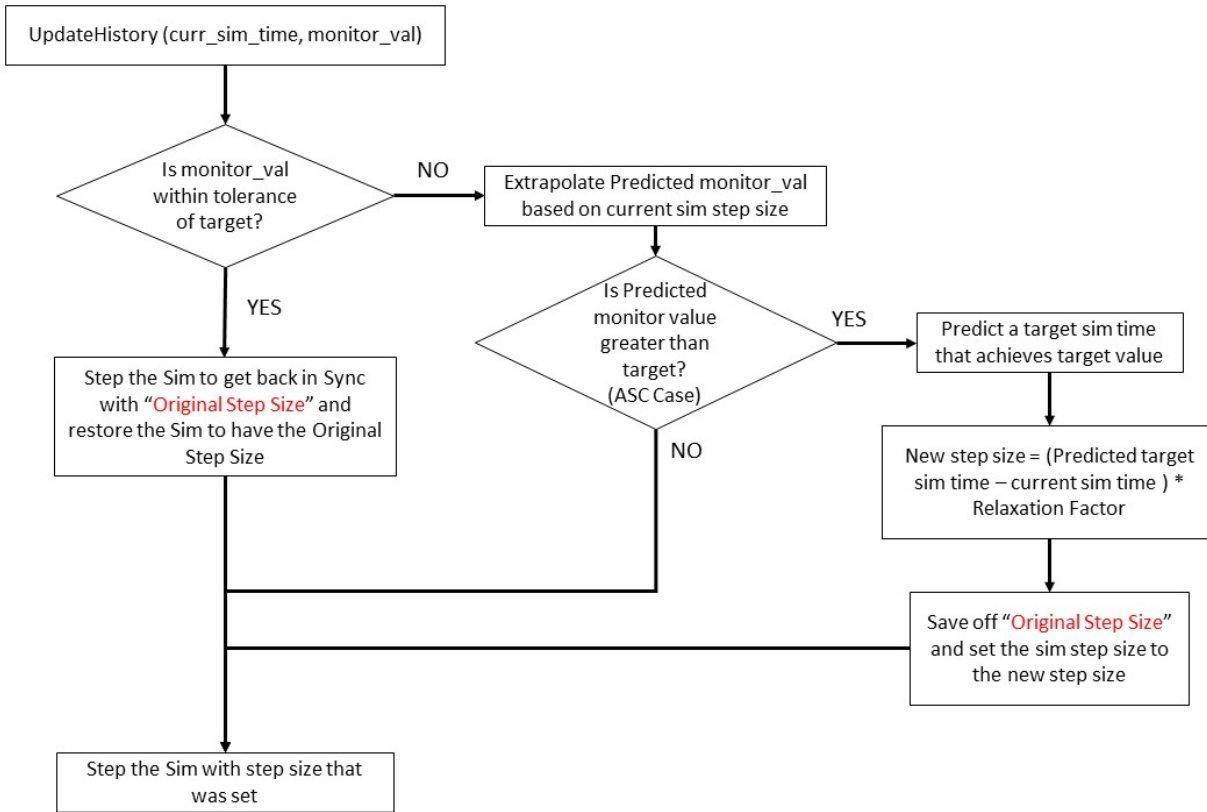
PYTHON

### 2.6.1.9.6. Regula-Falsi Implementation

Regula-Falsi is a technique used to determine the moment when a monitored value matches a target value within a specified tolerance. It does this by changing the integration and IO stepsizes to achieve the targeted value within the specified tolerance. Once the target value has been achieved, the associated phasing event will occur (i.e. Perform a Burn Command at a certain Mission Elapsed Time).

The way COMPASS implements Regula-Falsi is not true "Regula-Falsi" because the simulation does not step backwards if the target value is overshot. Instead, the COMPASS Regula-Falsi Implementation is more accurately described as a "Forward Cubic Spline" algorithm where the last four monitor values and simulation times are used to determine the appropriate step size to use in order to inch toward the future target value.

The following Flow Chart shows pseudo-code logic to illustrate how a Regula-Falsi Monitor Phase is executed by the PhaseManager for an Ascending case.



The UpdateHistory routine on the top of the flow chart shows that the current sim time and monitor value is collected into a local history array for both sim time and the monitor value. (The monitor value can be any measured value from the sim).

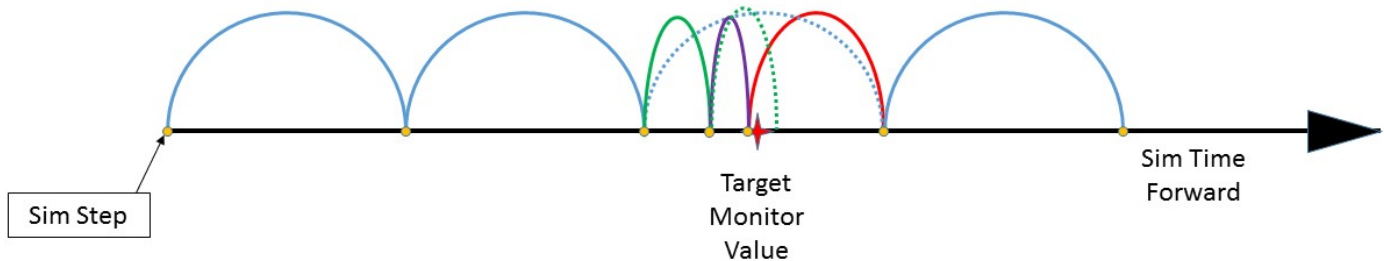
The local history array of sim time and monitor values are used to extrapolate the predicted monitor value based on the current step size. If it turns out that the predicted value surpasses the target value, then a new step size is determined. A predicted target sim time for the target value is determined using a CubicSpline on the local history arrays. The new step size is determined by taking the difference of the predicted target sim time and the current sim time, but wait...there's more! The new step size is also a function of a parameter passed in when the phase was created called... Relaxation Factor. The relaxation factor is multiplied by the difference of the two sim times and is used to slow down convergence in meeting the monitor value target. Smaller values will result in smaller integration step sizes used in trying to achieve the target monitor value.

```
new_stepsize = (pred_target_sim_time - current_sim_time) * relaxation_factor
```

PYTHON

The original step size is retained so that the simulation step size may be restored to that value after a Regula-Falsi phase event has occurred

This kind of logic leads toward the kind of simulation steps shown in the Figure Below



The yellow small dots on the time-line progression represent every time an official sim step was taken and data logged. An official sim step is defined to be the state after every IO Step the simulation has taken. When Regula-Falsi changes step sizes to inch towards the target value, it alters both the IO Step size and the integration step size. The dotted lines represent when the PhasingManager extrapolated the future monitor values but found that it would overshoot. The change in color of an arc represents the changing of the step size by the PhaseManager:

The flow chart logic repeats until the simulation takes a step in which the monitor value matches the target value within the Regula-Falsi tolerance and the data is logged. Afterwards, a calculated step size is used to reach a time that is consistent with the nominal step interval, then the nominal step size is restored. (see figure above Red Arc Step back in Sync to original Blue Arc Step)

**What happens if the monitored value already exceeds the target value (and tolerance) during Regula-Falsi?** Great Question! The way the Regula-Falsi is coded at this moment, the Regula-Falsi event will overshoot the target value and the simulation will terminate from an Exception: Cannot step back in time x (seconds) in Phase Y. For example, consider a cannonball sim with step sizes of 1.0 seconds with Earth gravity always pulling down the negative Z-Axis. The initial conditions are a cannonball of 1.0 Kilogram mass dropped from an inertial frame at sim time = 0. If a Regula-Falsi monitor type were to be added to the Phasing Manager with a monitor of Z position and a target value of -10.0, tolerance of 0.1, slope of 'DES', and a phasetype of 'Free', by the second step the sim would have already stepped over the target value and the Regula-Falsi event will not be triggered. The sim would terminate from an uncaught exception. An example of this thrown exception is shown below.

```

Traceback (most recent call last):
  File "CannonBallSim.py", line 141, in <module>
    manager.execute()
  File "/ms/ua/compass2/work/galaniz/dev/Cycle15/lib/PYTHON/DshellCommon/fsm/phasing.py", line 137, in execute
    res = self.step()
  File "/ms/ua/compass2/work/galaniz/dev/Cycle15/lib/PYTHON/DshellCommon/fsm/phasing.py", line 152, in step
    self._monitor()
  File "/ms/ua/compass2/work/galaniz/dev/Cycle15/lib/PYTHON/DshellCommon/fsm/phasing.py", line 227, in _monitor
    self._checkForTrigger()
  File "/ms/ua/compass2/work/galaniz/dev/Cycle15/lib/PYTHON/DshellCommon/fsm/phasing.py", line 237, in _checkForTrigger
    res = phase.monitor_inst()
  File "/ms/ua/compass2/work/galaniz/dev/Cycle15/lib/PYTHON/DshellCommon/fsm/phasing.py", line 562, in __call__
    return self.checkConvergence()
  File "/ms/ua/compass2/work/galaniz/dev/Cycle15/lib/PYTHON/DshellCommon/fsm/phasing.py", line 622, in checkConvergence
    self.changeStepsize(new_stepsize)
  File "/ms/ua/compass2/work/galaniz/dev/Cycle15/lib/PYTHON/DshellCommon/fsm/phasing.py", line 629, in changeStepsize
    raise Exception('Cannot step back in time %.14g in %s' % (new_stepsize, self.phase))
Exception: Cannot step back in time -1.9122324159021 in Phase: 1 SimpleExample (Free)

```

Consider this scenario when creating Regula-Falsi phases so as to not overshoot your target value.

### 2.6.1.9.7. Regula-Falsi Example

Let's assume an example of a cannonball of 1.0 Kilogram mass is being exerted on by gravity downwards (- Z-Axis) in an inertial frame. The initial conditions of our scenario would be for the cannonball to be dropped from inertial position origin at time 0.0

First Step is to create an instance of the PhasingManager by passing in your cannonball simulation.

```

from DshellCommon.fsm.phasing import PhasingManager, PhaseTime
phasing = PhasingManager(sim)

```

Now all that is needed is to add a phases to monitor the Regula-Falsi event.

```
phasing.addPhase(number=1,
                  name='SimpleExample',
                  monitor= ballZVel,
                  value= -48.5,
                  phasetype='Free',
                  slope='DES',
                  monitortype='RegulaFalsi',
                  tolerance=0.1,
                  tolerance_type='Absolute',
                  relaxation_factor=0.85,
                  cmd=sim.terminate)
```

PYTHON

These input fields are used to add Trip Event (by Default) and Regula-Falsi Phases. Some of the fields are **Regula-Falsi Specific/Unique**, which include:

1. **slope** - (Default = 'ASC\_GTE')
2. **tolerance** - (Default = 0.1)
3. **tolerance\_type** - (Default = "Percent")
4. **relaxation\_factor** - (Default = 0.85)

Elaboration of these parameters are explained in the above section 'Initialize Desired Phases'.

Let's explain the values of all the fields that are passed in this example

The number field will be a unique phase number identifier from other phases. 1 is chosen for this SimpleExample.

The monitor value will be populated with the cannonball's Z Velocity function call.

The target value is chosen to be -48.5 meters/second (m/s). (Note: The monitor and target values do not have to depend on time, but any measured value from the simulation)

The phase type is Free because the event can happen at any time.

The **slope** is descending because the velocity is increasing negatively. If the phase monitortype is '**RegulaFalsi**', the only valid strings are 'ASC' or 'DES'.

The monitor type is chosen as 'RegulaFalsi' to allow the event to be triggered at the target value within a tolerance.

The **tolerance** is chosen to be within 0.1 m/s. The tolerance field is only used when the monitortype is 'RegulaFalsi'.

**tolerance\_type** is 'Absolute' because the tolerance is not a percentage of the target value. This tolerance\_type field is only used when the monitortype is 'RegulaFalsi'

The **relaxation\_factor** is 0.85 to make sure that the predicted step size will not overshoot the target value. The lower the relaxation factor; the more steps will be taken to inch toward the target value. This relaxation\_factor field is only used when the monitortype is 'RegulaFalsi'.

Finally, the cmd field is populated with a reference function to call upon when the monitor value matches the target value within the specified tolerance. In this case, the sim.terminate function will be called upon the Regula-Falsi event being triggered.

All of that just takes care of the phasing setup.

We will not set the IO step size and integration steps for the simulation to allow the default step size of 1 second to be used

The final step is to execute the simulation by executing the Step Manager:

```
manager.execute()
```

PYTHON

During the simulation, the triggered events will be printed to the screen

```
Time:      4.9345 Start Phase:   1 SimpleExample (Free)
Monitor Var: ballZVel
Trigger time:      4.93449541
Monitor Val:      -48.5
- Sim Val:        -48.4074
Delta:            -0.0925999887
Simulation terminated at t=4.93450
```

Each of the triggered Phases has the Simulation Time at which the event was triggered, the target value and the monitored simulation value showing that the event triggered within the tolerance.

As a final note, it is possible to pass in the debug flag as true when adding the regula falsi monitor phases to the phase manager. This will allow more info during the simulation run of how the step sizes are being altered during the run to achieve the target value.

## 2.6.1.10. Utilities

### 2.6.1.10.1. CUBIC SPLINE (cubicspline.py)

Provides a CubicSpline class for interpolation.

This class is used to interpolate using a cubic spline given x, y sequences. x must be monotonically increasing or decreasing sequence. Also, len(x) == len(y)

```
:param x: List of independent values
:param y: List of dependent values
```

The class instance is callable as the following example shows usage:

```
>>> x = [1., 2., 3., 4., 5.]
>>> y = [2., 8., 12., 13., 25.]
>>> f = CubicSpline(x, y)
>>> print([f(x) for x in (0., 2., 3., 4., 5., 15., 45.)])
[-4.125, 8.0, 12.0, 13.0, 25.0, 113.75, 380.0]
```

PYTHON

### 2.6.1.11. Writing DshellCommon Simulation Run Scripts

In order to construct a simulation use tools provided by the DshellCommon module, users will create a run script. The run-scripts have several parts:

#### 2.6.1.11.1. Run script preamble

**The beginning of the run script has several 'preamble' elements**

- Do some basic imports
- Create the simulation executive
- Import the parameter libraries to be uses in the simulation

At a minimum, a preamble might look like this:

```
# Create the simulation

from DshellCommon.SimulationExecutive import SimulationExecutive

sim = SimulationExecutive()

# Load the parameter libraries

import sys
sys.path.append('./library')

import targets
import bodies
import states

# Get access to necessary assembly classes
from DshellCommon import assemblies
```

PYTHON

#### 2.6.1.11.2. Define the simulation definition block

The next part of a run script is the simulation definition block.

The following excerpt from an example simulation script illustrates the structure of simulation definition block:



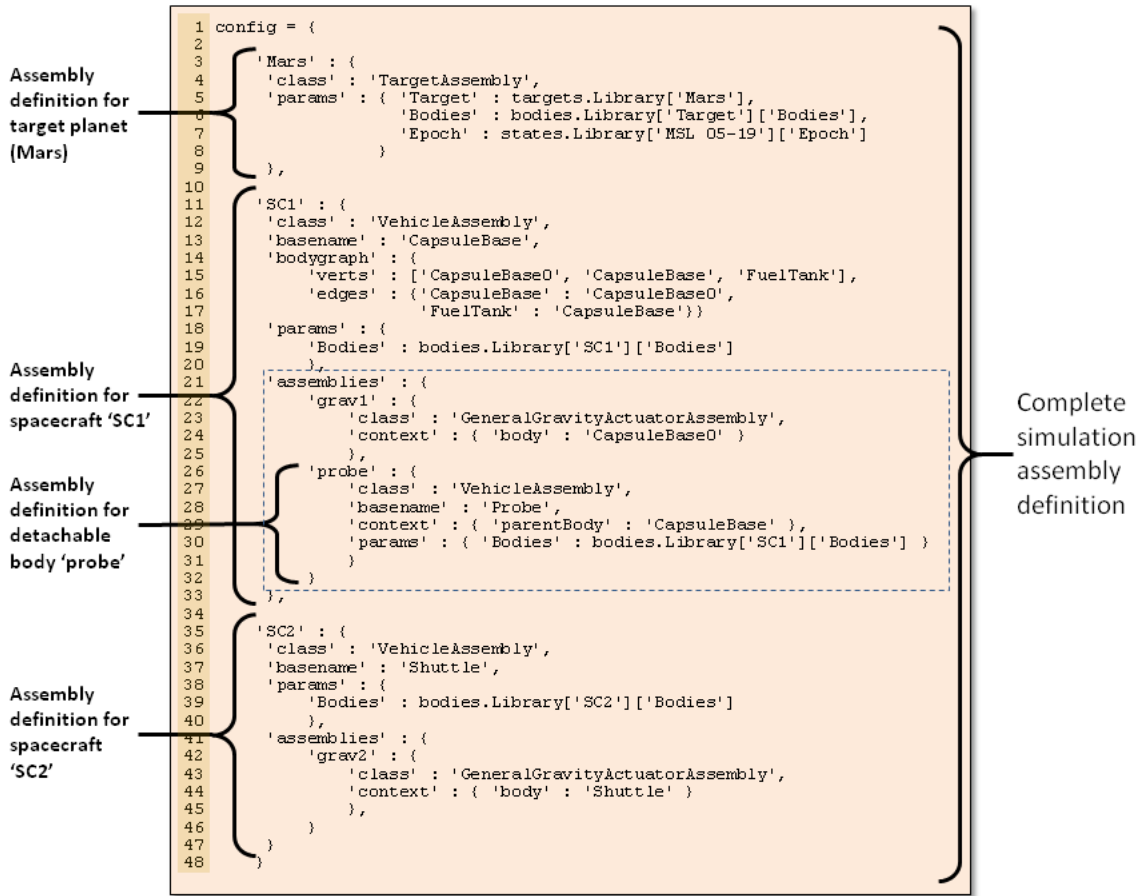


Figure 30. Layout of Example Simulation Definition Block

**In this simulation, we create**

- The target planet (Mars)
- Spacecraft 'SC1' (which contains an attached 'probe' body)
- Spacecraft 'SC2'

**NOTES**

- The structure of the assembly definition for the target planet, SC1, and SC2 are identical. An assembly class must be specified (the 'class' item), and the other items are optional.
- Note that in the SC1 assembly definitions there is an 'assemblies' block (enclosed in dotted lines). It has the same structure as the top-level block. In fact, the entire simulation definition block can be thought of as the 'assemblies' block for the top-level assembly.
- Spacecraft SC1, SC2, and probe are all instances of the VehicleAssembly class.
- The nested nature of the construction is deliberate. The 'probe' assembly is actually a separate VehicleAssembly because it could potentially be detached from SC1 and become free-flying spacecraft.

For more information about the syntax and grammar of the simulation definition block, see the reference section: [reference/scripts\\_bnf](#).

For an example of how to create and share signals in a run script, please this regression test:

2.6.1.11.2.1. Creating signals and signal ties in assembly definition

*Basic test*

Creating signals and signal ties basic regtest

*More detailed test*

Creating signals and signal ties more detailed regtest

2.6.1.11.3. Initialize the simulation

After the simulation structure is defined, all objects defined in it can be created by a few simple calls:

```

# Create all the assemblies
sim.createAssemblies(config)

sim.bindState()

sim.resetState(0.0)

```

PYTHON

#### 2.6.1.11.4. Run the simulation

At this point, the simulation is ready to run. The user can provide code to alter initial conditions, set up finite state machines, etc. If a finite state machine is not used, executing the simulation could be very simple:

```
# Run the simulation to a specific end time
tfinal = 5.0
while sim.time() < tfinal:
    sim.stepSim()
```

PYTHON

##### 2.6.1.11.4.1. Example run script

###### 2.6.1.11.4.1.1. test\_VehicleAssembly\_attach/script\_attachVehicle.py

The run script shown in this page is simplified from the test/test-Vehicle-attach/script\_attachVehicle.py doctest:

Create and attach 2 vehicles regtest

##### *Dwatch Regtest*

For examples on how to log data using Dwatch, please see this example:

Simple test for basic Dwatch functionality in a simulation

##### 2.6.1.11.4.2. Suggestions for writing run scripts

###### 2.6.1.11.4.2.1. Library Imports in User Scripts

In older user scripts, we often used the syntax:

```
sys.path.append('./library')
import bodies
```

PYTHON

and when we reference something in the bodies library file:

```
bodies.Library['SC']['CapsuleBase']
```

PYTHON

However, a cleaner syntax is do it differently. Import the library file like this:

```
sys.path.append('./library')
from bodies import Library as bodies
```

PYTHON

Then we can reference it like this:

```
bodies['SC']['CapsuleBase']
```

PYTHON

###### 2.6.1.11.4.2.2. Predefined Constants in User Scripts

In config dictionaries in user run scripts have many string tokens that are used in the scripts such the strings 'class', 'basename', 'params', 'signals', etc:

```
config = {
    'SC1' : {
        'class' : 'VehicleAssembly',
        'basename' : 'baseBody',
        'params' : {
            'Bodies' : bodies.Library['SC1']['Bodies']
        },
        'signals' : {
            ...

```

PYTHON

We have defined a set of constants for these string tokens. In order to use them you will need to do an import:

```
from DshellCommon.ConfigConstants import CLASS, BASENAME, PARAMS, BODIES, SIGNALS

config = {
    'SC1' : {
        CLASS : 'VehicleAssembly',
        BASENAME : 'baseBody',
        PARAMS : {
            BODIES : bodies['SC1']['Bodies']
        },
        SIGNALS : {
            ...

```

PYTHON

Notice that the string tokens have been replaced by uppercase constants. Most of the commonly encountered tokens have constants defined for them. To see the full list of available constants, please see the file:

##### **DshellCommon/python/assemblies/ConfigConstants.py**

Note that these constants can also be used in python Assembly classes.

##### **The goals for these constants are**

- Reduce potential errors due to misspellings of the string tokens. An error in a constant name will be caught immediately when the code is executed.

- The constants are very obvious in the user scripts and stand out from user-defined strings (eg, for assembly or body names). This helps the user read and understand the user script more easily.
- The constants reduce the amount of indenting required (slightly).

#### 2.6.1.12. Useful Example Scripts (Regression Tests)

Doctests tests form a valuable resource since they are example scripts that exercise many aspects of the system. DshellCommon has many regression tests. In order to make it easier to locate useful tests, the following table lists all the DshellCommon regression tests and a little about each of them. The link in the left column provides a link to allow readers to examine the script for the regtest.

| Link to regression test                         | Description   |
|---|---|
| <a href="#">test_BaseAssembly</a>               | Simple stand-alone test for BaseAssembly class  |
| <a href="#">test_Dwatch</a>                     | Simple test for basic Dwatch functionality in a simulation  |
| <a href="#">test_SimulationAssembly</a>         | Simple test for basic SimulationAssembly functionality  |
| <a href="#">test_SimulationExecutive</a>        | Simple test for basic SimulationExecutive functionality without Spice   |
| <a href="#">test_SimulationExecutive</a>        | Simple test for basic SimulationExecutive functionality with Spice  |
| <a href="#">test_child_signals</a>              | Example of creating signals and signal ties in the user run scripts by specifications in the user run script (in the assembly definition part).   |
| <a href="#">test_FSM_basic</a>                  | Create a finite state machine based on DshellCommonFSM and exercise its functionality in a stand-alone test (without a simulator).  |
| <a href="#">test_FSM_sim</a>                    | Create a finite state machine based on DshellCommonFSM and exercise its functionality in a simulation. Check basic transitions and termination during simulation.   |
| <a href="#">test_FSM_sim2</a>                   | Create a finite state machine based on DshellCommonFSM and exercise its functionality in a simulation. Check basic transitions and termination during simulation using the 'generalTrigger' trigger function from FsmUtils.py |
| <a href="#">test_derivedParamAndAssembly</a>    | Tests to create parameter classes of derived classes and test them in assembly creation.  |
| <a href="#">test_VehicleAssembly_checkpoint</a> | Test checkpointing with a simulation with 1 vehicle with one body. Start the simulation, checkpoint, run the simulation, restore from checkpoint and verify the restored state.   |
| <a href="#">testVA_script_1v1b</a>              | Manually create a VehicleAssembly with 1 body, no gravity.  |
| <a href="#">testVA_script_1v3b</a>              | Manually create a VehicleAssembly with 3 bodies, no gravity.  |
| <a href="#">testVA_script_2v1b</a>              | Manually create two VehicleAssemblies' 1 body each, no gravity.   |
| <a href="#">testVA2_script_1v3b_nograv</a>      | Create a VehicleAssembly with 3 bodies, no gravity, using the normal run script approach.   |
| <a href="#">testVA2_script_1v3b_grav</a>        | Create a VehicleAssembly with 3 bodies and gravity, using the normal run script approach.   |
| <a href="#">testVA2_script_2v3b_grav</a>        | Create two VehicleAssembly's with 2 bodies on one, 1 body on the other; and gravity, using the normal run script approach.  |
| <a href="#">testVA2_script_2v4b_grav</a>        | Create two VehicleAssembly's with 3 bodies on one (including one on a subsidiary VehicleAssembly), one body on the other; and gravity, using the normal run script approach. Tests hierarchical VehicleAssembly construction. |
| <a href="#">test_ExtDistActuatorAssembly</a>    | Create a free-flying vehicle without gravity and apply a force and torque to test ExternalDisturbanceActuatorAssembly.  |
| <a href="#">test_ExtDistMotorAssembly</a>       | Create a free-flying vehicle without gravity and apply a force and torque to test ExternalDisturbanceMotorAssembly.   |
| <a href="#">testVA_attach_detach1</a>           | Create a vehicle with 2 bodies and a separable probe body. Detach the probe and lock it (freeze it) with respect to the inertial frame.   |
| <a href="#">testVA_attach_detach2</a>           | Create a vehicle with 2 bodies and a separable probe body. Detach the probe and let it fly free with respect to the inertial frame.   |
| <a href="#">testVA_attach_attachToBody1</a>     | Create a vehicle with 2 bodies and a separable probe body. Detach the probe and lock it to a rotating planetary body (PCR).   |
| <a href="#">testVA_attach_attachToBody2</a>     | Create a vehicle with 2 bodies and a separable probe body. Detach the probe and let it fly freely with respect to a rotating planetary body (PCR).  |
| <a href="#">testVA_attach_attachVehicle</a>     | Create 2 vehicles. In the simulation one vehicle attaches to the other vehicle to roughly simulate docking a vehicle to the space station.  |
| <a href="#">test_VehicleAssembly_target</a>     | Create a vehicle with 3 bodies and a planetary target body. Set up simple gravity.  |
| <a href="#">test_VehicleAssembly_gravity</a>    | Create a vehicle with 3 bodies and a planetary target body. Set up an aspherical gravity tied to the planetary body.  |
| <a href="#">test_VehicleAssembly_spice</a>      | Create a vehicle with 3 bodies and a planetary target body initialized with Spice. Set up simple gravity.   |

| Link to regression test     | Description   |
|-----------------------------|---|
| test_VehicleAssembly_fsmsep | Create a vehicle with 2 attached bodies: a service module and command module. Separate the the two bodies during the simulation using a finite state machine. |
| test_Topos                  | Test adding topographic objects to Targets.   |
| test_MultiRun               | Test cases for multirun execution of a simulaiton.  |

#### Simple test for basic Dwatch functionality in a simulation

- ▶ [Click to see the DshellCommon/test/test\\_Ndarts/test\\_Dwatch/script1.py script](#)

#### Simple stand-alone test for BaseAssembly class

- ▶ [Click to see the DshellCommon/test/test\\_Ndarts/test\\_BaseAssembly/script1.py script](#)

Create a free-flying vehicle without gravity and apply a force and torque to test ExternalDisturbanceActuatorAssembly.

- ▶ [Click to see the DshellCommon/test/test\\_Ndarts/test\\_ExtDistActuatorAssembly/script.py script](#)

Create a free-flying vehicle without gravity and apply a force and torque to test ExternalDisturbanceActuatorAssembly.

- ▶ [Click to see the DshellCommon/test/test\\_Ndarts/test\\_ExtDistMotorAssembly/script.py script](#)

#### Simple test for basic Dwatch functionality in a simulation

- ▶ [Click to see the DshellCommon/test/test\\_Ndarts/test\\_fsm/script\\_sim.py script](#)

Create a finite state machine based on DshellCommonFSM and exercise its functionality in a stand-alone test (without a simulaton)

- ▶ [Click to see the DshellCommon/test/test\\_Ndarts/test\\_fsm/script\\_basic.py script](#)

Create a finite state machine based on DshellCommonFSM and exercise its functionality in a simulation. Check basic transitions and termination during simulation.

- ▶ [Click to see the DshellCommon/test/test\\_Ndarts/test\\_fsm/script\\_sim.py script](#)

Create a finite state machine based on DshellCommonFSM and exercise its functionality in a simulation. Check basic transitions and termination during simulation using the 'generalTrigger' trigger function from FsmUtils.py

- ▶ [Click to see the DshellCommon/test/test\\_Ndarts/test\\_fsm/script\\_sim2.py script](#)

Test cases for multirun execution of a simulaiton.

- ▶ [Click to see the DshellCommon/test/test\\_Ndarts/test\\_MultiRun/script2.py script](#)

#### Simple test for basic SimulationAssembly functionality

- ▶ [Click to see the DshellCommon/test/test\\_Ndarts/test\\_SimulationAssembly/script.py script](#)

Test adding topographic objects to Targets.

- ▶ [Click to see the DshellCommon/test/test\\_Ndarts/test\\_Topos/script1.py script](#)

#### *SimulationExecutive related*

##### Simple test for basic SimulationExecutive functionality

- ▶ [Click to see the DshellCommon/test/test\\_Ndarts/test\\_SimulationExecutive/script1.py script](#)

##### Simple test for basic SimulationExecutive functionality

- ▶ [Click to see the DshellCommon/test/test\\_Ndarts/test\\_SimulationExecutive/script2.py script](#)

#### *TargetAssembly related*

VehicleAssembly with 3 bodies and a planetary target body, simple gravity.

- ▶ [Click to see the DshellCommon/test/test\\_Ndarts/test\\_VehicleAssembly\\_target/script.py script](#)

Create 2 vehicles. In the simulation one vehicle attaches to the other vehicle to roughly simulate docking a vehicle to the space station.

- ▶ [Click to see the DshellCommon/test/test\\_Ndarts/test\\_VehicleAssembly\\_attach/script\\_attachVehicle.py script](#)

#### *VehicleAssembly related*

Manually create a VehicleAssembly with 1 body, no gravity.

- ▶ [Click to see the DshellCommon/test/test\\_Ndarts/test\\_VehicleAssembly/script\\_1v1b.py script](#)

Manually create a VehicleAssembly with 3 bodies, no gravity.

- ▶ [Click to see the DshellCommon/test/test\\_Ndarts/test\\_VehicleAssembly/script\\_1v3b.py script](#)

Manually create two VehicleAssemblies' 1 body each, no gravity.

- ▶ [Click to see the DshellCommon/test/test\\_Ndarts/test\\_VehicleAssembly/script\\_2v1b.py script](#)

VehicleAssembly with 2 bodies and a separable probe body. Detach the probe and lock it (freeze it) with respect to the inertial frame

- ▶ [Click to see the DshellCommon/test/test\\_Ndarts/test\\_VehicleAssembly\\_attach/script\\_detach1.py script](#)

VehicleAssembly with 2 bodies and a separable probe body. Detach the probe and let it fly away with respect to the inertial frame

- ▶ [Click to see the DshellCommon/test/test\\_Ndarts/test\\_VehicleAssembly\\_attach/script\\_detach2.p script](#)

VehicleAssembly with 2 bodies and a separable probe body. Detach the probe and lock it to a rotating planetary body (PCR)

- ▶ [Click to see the DshellCommon/test/test\\_Ndarts/test\\_VehicleAssembly\\_attach/script\\_attachToBody1.py script](#)

VehicleAssembly with 2 bodies and a separable probe body. Detach the probe and let it fly freely with respect to a rotating planetary body (PCR)

- ▶ [Click to see the DshellCommon/test/test\\_Ndarts/test\\_VehicleAssembly\\_attach/script\\_attachToBody2.py script](#)

Test checkpointing with a simulation with 1 vehicle with one body. Start the simulation, checkpoint, run the simulation, restore from checkpoint and verify the restored state.

- ▶ [Click to see the DshellCommon/test/test\\_Ndarts/test\\_VehicleAssembly\\_checkpoint/script.py script](#)

Create a vehicle with 2 attached bodies: a service module and command module. Separate the the two bodies during the simulation using a finite state machine

- ▶ [Click to see the DshellCommon/test/test\\_Ndarts/test\\_VehicleAssembly\\_fsmsep/script.py script](#)

VehicleAssembly with 3 bodies and a planetary target body, aspherical gravity.

- ▶ [Click to see the DshellCommon/test/test\\_Ndarts/test\\_VehicleAssembly\\_gravity/script1.py \(with AsphericalGravityActuatorAssembly\) script](#)

VehicleAssembly with 3 bodies and a planetary target body with Spice, simple gravity.

- ▶ [Click to see the DshellCommon/test/test\\_Ndarts/test\\_VehicleAssembly\\_spice/SpiceFramesTest.py script](#)

*VehicleAssembly2 related*

Regular creation of a VehicleAssembly with 3 bodies, no gravity.

- ▶ [Click to see the DshellCommon/test/test\\_Ndarts/test\\_VehicleAssembly2/script\\_1v3b\\_nograv.py script](#)

Regular creation of a VehicleAssembly with 3 bodies, with gravity.

- ▶ [Click to see the DshellCommon/test/test\\_Ndarts/test\\_VehicleAssembly2/script\\_1v3b\\_grav.py script](#)

Regular creation of two VehicleAssembly with 2+1 bodies, with gravity.

- ▶ [Click to see the DshellCommon/test/test\\_Ndarts/test\\_VehicleAssembly2/script\\_2v3b\\_grav.p script](#)

Regular creation of two VehicleAssembly with 3+1 bodies, with gravity.

- ▶ [Click to see the DshellCommon/test/test\\_Ndarts/test\\_VehicleAssembly2/script\\_2v4b\\_grav.py script](#)

*Miscellaneous*

Test basic derived Assembly and param classes

- ▶ [Click to see the DshellCommon/test/test\\_Ndarts/test\\_derivedParamAndAssembly/script.py script](#)

Example of creating signals and signal ties in the user run scripts by specifications in the user run script (in the assembly definition part).

- ▶ [Click to see the DshellCommon/test/test\\_Ndarts/test\\_child\\_signals/script1.py \(basic test\) script](#)

Example of creating signals and signal ties in the user run scripts by specifications in the user run script (in the assembly definition part).

- ▶ [Click to see the DshellCommon/test/test\\_Ndarts/test\\_child\\_signals/script2.py \(more detailed test\) script](#)

## 2.6.2. Reference

### 2.6.2.1. DshellCommon Simulation Executive

#### 2.6.2.1.1. SimulationExecutive

##### 2.6.2.1.1.1. Introduction

The SimulationExecutive is the simulation engine.

SimulationExecutive is derived from the C++ DshellX simulation engine.

#### 2.6.2.1.1.1.1. How to use the SimulationExecutive

In order to use the SimulationExecutive in your user run scripts, only a few lines are needed:

```
# Create the simulation
from DshellCommon.SimulationExecutive import SimulationExecutive
sim = SimulationExecutive()
```

This constructs the simulation object and does some basic initialization.

#### Popular SimulationExecutive Functions

Some of the most common functions for the SimulationExecutive are:

#### Functions related to accessing and advancing the simulation time:

| Function  | Purpose   |
|---|---|
| <a href="https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/Dshell++/html/classDshell__Py_1_1Simulation.html#a0c33975fda4de1a5214990d13f62caaf#a4dcb139d466f705cdb1d19ce5720392d">sim.step()</a><br>(https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/Dshell++/html/classDshell__Py_1_1Simulation.html#a0c33975fda4de1a5214990d13f62caaf#a4dcb139d466f705cdb1d19ce5720392d) | Step the simulation forward in time one step and advance any finite state machines set up by the simulation script. |
| <a href="https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/Dshell++/html/classDshell__Py_1_1Simulation.html#a0c33975fda4de1a5214990d13f62caaf">sim.advanceTimeBy(delta_time)</a> (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/Dshell++/html/classDshell__Py_1_1Simulation.html#a0c33975fda4de1a5214990d13f62caaf)   | Step the simulation forward by <i>delta_time</i> seconds.   |
| <a href="https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/Dshell++/html/classDshell__Py_1_1Simulation.html#acd0a8b0df0621972f77870f02e2549d9">sim.advanceTimeTo(time)</a> (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/Dshell++/html/classDshell__Py_1_1Simulation.html#acd0a8b0df0621972f77870f02e2549d9)   | Step the simulation to <i>time</i> seconds.   |
| <a href="https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/Dshell++/html/classDshell__Py_1_1Simulation.html#a2ab4e9ed003fe1c9b0a8e4fe604a4bbc">sim.time()</a> (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/Dshell++/html/classDshell__Py_1_1Simulation.html#a2ab4e9ed003fe1c9b0a8e4fe604a4bbc)  | Return the current simulation time in seconds.  |

#### Functions for saving/restoring/resetting the state of the simulation

| Function   | Purpose  |
|--|--|
| <a href="https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/Dshell++/html/classDshell__Py_1_1Simulation.html#a55e2c6eeaea60b4107e683a4e98c42af">sim.checkpoint(filename)</a><br>(https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/Dshell++/html/classDshell__Py_1_1Simulation.html#a55e2c6eeaea60b4107e683a4e98c42af)            | Write the entire state of the simulation out to a checkpoint file. |
| <a href="https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/Dshell++/html/classDshell__Py_1_1Simulation.html#a758ddf674dc004582e5bac5b2aafa751">sim.restoreFromCheckpoint(filename)</a><br>(https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/Dshell++/html/classDshell__Py_1_1Simulation.html#a758ddf674dc004582e5bac5b2aafa751) | Restore the entire state of the simulation from a checkpoint file. |
| <a href="https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/Dshell++/html/classDshell__Py_1_1Simulation.html#a9f4d5b8d4581ad4be096320ecbbcf8c0">sim.clear()</a><br>(https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/Dshell++/html/classDshell__Py_1_1Simulation.html#a9f4d5b8d4581ad4be096320ecbbcf8c0)                         | Reset the integrator back to time=0.0                              |

#### Functions for creating and dealing with assemblies

| Function   | Purpose   |
|--|---|
| <a href="https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/Dshell++/html/classDshell__Py_1_1Simulation.html#ab16f089baec6ad340920753a8097b5fd">sim.assemblyList()</a><br>(https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/Dshell++/html/classDshell__Py_1_1Simulation.html#ab16f089baec6ad340920753a8097b5fd)              | Returns a list of all assembly objects in the simulation. |
| <a href="https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/Dshell++/html/classDshell__Py_1_1Simulation.html#a1f082c0f3f2adea6a8d56a50d2b3c6a7">sim.assembly(name,depth,strict)</a><br>(https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/Dshell++/html/classDshell__Py_1_1Simulation.html#a1f082c0f3f2adea6a8d56a50d2b3c6a7) | Get the specified assembly object                         |

#### Functions for dealing with the integrator

| Functions for dealing with the integrator | Purpose |
|---|---------|
|---|---------|

| Functions for dealing with the integrator  | Purpose   |
|--|---|
| <code>sim_currIntegrator()</code><br>( <a href="https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/Dshell++/html/classDshell_Py_1_1Simulation.html#adc6967d27796015d2c93029071b592e0">https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/Dshell++/html/classDshell_Py_1_1Simulation.html#adc6967d27796015d2c93029071b592e0</a> )       | Get the integrator object                               |
| <code>sim_integratorSelect(name)</code><br>( <a href="https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/Dshell++/html/classDshell_Py_1_1Simulation.html#ab83c00272af2c0cfad0094dd30f1857f">https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/Dshell++/html/classDshell_Py_1_1Simulation.html#ab83c00272af2c0cfad0094dd30f1857f</a> ) | Switch to the specified integrator                      |
| <code>sim_stepSize(sec_nanosec)</code><br>( <a href="https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/Dshell++/html/classDshell_Py_1_1Simulation.html#a91077e8b2f302d1d2ddd0432dc2ee8f5">https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/Dshell++/html/classDshell_Py_1_1Simulation.html#a91077e8b2f302d1d2ddd0432dc2ee8f5</a> )  | Get or set the step size (integer seconds, nanoseconds) |

There are many other commands that SimulationExecutive supports including commands to help set up the simulation structure (Assemblies), dealing with the integrator, setting up the simulation time step size, etc. For more details see the SimulationExecutive API Documentation section below.

#### 2.6.2.1.1.2. Related Regression Tests

For an example of how to use the most basic functions of SimulationExecutive, see these test:

##### 2.6.2.1.1.2.1. test\_SimulationExecutive/script\*.py

Note: Many of the regression tests use SimulationExecutive.

- Simple test for basic SimulationExecutive functionality without Spice
- Simple test for basic SimulationExecutive functionality with Spice

##### 2.6.2.1.1.2.2. test\_VehicleAssembly\_checkpoint/script.py

- Test checkpointing with a simulation with 1 vehicle with one body. Start the simulation, checkpoint, run the simulation, restore from checkpoint and verify the restored state.

#### 2.6.2.1.1.3. SimulationExecutive API Documentation



##### Note

For Doxygen documentation, please see: [SimulationExecutive](#)

([https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/DshellCommon/html/classpython\\_1\\_1SimulationExecutive\\_1\\_1SimulationExecutive.html](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/DshellCommon/html/classpython_1_1SimulationExecutive_1_1SimulationExecutive.html))

## 2.6.2.2. DshellCommon Parameter Basics

### 2.6.2.2.1. Parameter Class Overview

A parameter class is basically a dictionary that lists the allowed parameters, describes each of the parameters and provides the associated units.

Parameter classes are the preferred method for setting parameters for a given assembly and model. Although these classes can be treated by the user as a dictionary, they offer several benefits over dictionaries:

- *Name checking*: Parameter classes have a list of all allowed parameters and whether each parameter is required or optional.
- *Descriptions*: Each parameter is associated with a long description to allow introspection.
- *Units*: Parameters are associated with units. These are used to provide input parameter unit conversions (**future capability**)
- *Alternate inputs*: Each assembly and model depends on a specific set of parameters in a specific reference frame or coordinate system. Parameter classes provide the user the flexibility of inputting a given parameter type in a different way. These inputs are converted to the form required by the assembly. For example: A parameter class can convert keplerian elements into the cartesian vectors required by the VehicleAssembly. Without this the assembly would have to handle all permutations of desired inputs making it much more complex or multiple assemblies would be required.

#### 2.6.2.2.1.1. Parameter Class Functionality

Although parameter classes are primarily dictionary-like objects, they are functional python classes and can have methods.

- *Default values*: A common function is to set a default value in the parameter class before it is loaded by the assembly and converted to a DvarBranch.

```
if 'rotationAngle' not in params:
    params['rotationAngle'] = 0.0
```

PYTHON

- *Derived values*: Another common function is to calculate derived values from required inputs.

```
if 'radiusPole' in params:
    params['flattening'] = (params['radiusEquator'] - params['radiusPole']) / params['radiusEquator']
    params['eccentricity'] = math.sqrt(params['radiusEquator']**2 - params['radiusPole']**2) / params['radiusEquator']
```

PYTHON

#### 2.6.2.2.1.1.1. Functionality Parameter Classes Should Not Have

Parameter classes should not contain code that is intended to be called::: \* directly by the user to generated derived values \* cyclically during the simulation

#### 2.6.2.2.1.1.2. Naming Convention

Parameter classes have a naming convention. The prefix shall be descriptive of the high level nature of the parameters. Using the high level description will help group like param classes. The fundamental parameter class will have the name 'Base'. Derived classes will replace 'Base' with a more descriptive name. The suffix will always be Param. Example: TargetBaseParam and TargetAsphericalParam.

#### 2.6.2.2.1.1.3. Creating a Parameter Class

There are two basic types of parameter classes, Base and Derived. A Base class derives directly from BaseParam. It contains parameters that are essential to a broad category of similar models. Derived classes derived from a Base class and typically provide additional parameters that are required for a more specific or specialized model.

Code example defining a parameter class with one parameter field, 'Q', might look like this:

```

from DshellCommon.params.BaseParam import BaseParam, ParamInfo

Q = 'Q'

class TestParam(BaseParam):
    """
    Test parameter
    """
    _params = {
        Q : ParamInfo(required=True,
                       description='Q',
                       data_type=float),
    }

def __init__(self, params=None, source=None, source_params=None, sim=None, **kw):
    """Constructor."""
    params = BaseParam.assignKeywordArgs(params, kw)

    BaseParam.__init__(self, params, source, source_params, sim)

```

where the constant 'Q' is introduced to reduce the chances of misspelling.

An instance of this parameter can be constructed using one of two approaches:

- Using the 'params' dictionary:

```
qpar = TestParam(params = {'Q' : 1.0})
```

or

- Using the keyword approach:

```
qpar = TestParam(Q = 1.0)
```

Notice the difference in the parameter field name in the two approaches (quoted in 1 and without in 2). Using the keyword approach has some advantages since it looks simpler; and misspellings of the parameter field 'Q' will be caught automatically.

#### 2.6.2.2.1.2. DshellCommon API Reference - Primary Param Classes

BaseParam EpochParams StateParams TargetParams BodyParams NodeParams GravityParams

##### 2.6.2.2.1.2.1. BaseParam

###### BaseParam Class Introduction

Base class for DshellCommon parameter classes

The BaseParam class will be the parent class of all Param classes defined in DshellCommon.

- Derived from the C++ DshellParams class
- Provides a layer over the C++ DshellParams class for any needed additional functionality.

###### BaseParam Class API Documentation



###### Note

For Doxygen documentation, please see: `BaseParam<BaseParam: :BaseParam>`

DshellCommon.params.BaseParam

##### 2.6.2.2.1.2.2. Epoch Parameters

###### Class Documentation

###### Introduction

The Epoch Param classes have a single purpose: set the j2000BaseTime that resides in Dshell. This time is used to compute all other derived times.



###### Note

Instead of using the Epoch Param classes, you can pass the epoch through the `start_time` parameter in the SimulationExecutiveNdarts constructor.

###### Epoch Param Classes

- EpochBaseParam:** not intended to be used directly.
- EpochDateParam:** Use Spice to interpret a string based time and date..
- EpochETParam:** Directly set seconds from J2000 epoch in ephemeris time.

###### Class API Documentation





*Note*

For Doxygen documentation, please see: \* EpochBaseParam \* EpochDateParam \* EpochETParam

**EpochDateParam**

DshellCommon.params.EpochDateParam.EpochDateParam.init

DshellCommon.params.EpochDateParam.EpochDateParam.setInitTime

**EpochETParam**

DshellCommon.params.EpochETParam

2.6.2.2.1.2.3. State Parameter

**Class Documentation**

**Introduction**

The State Param classes are used to convert various states into inertial states for use by Darts. The state is defined as a translational and rotational state, and there are two base classes from which the remaining State Param classes are derived:

- StateAttitudeBaseParam
- StatePosVelBaseParam

**State Attitude Param Classes**

Document State Attitude Param classes.

**State Position/Velocity Param Classes**

State Position Velocity param classes provide various means of generating an inertial cartesian state required by Darts.

Document State Pos/Vel Param classes.

**Class API Documentation**



*Note*

For Doxygen documentation, please see

- StatePosVelBaseParam
- StatePosVelCartesianParam
- StatePosVelSphericalParam
- StateLatLonAltParam
- StatePosVelVehRelParam
- StateAttitudeBaseParam
- StateAttitudeEulerParam
- StateAttitudeFPAlignParam
- StateAttitudeVehRelParam

2.6.2.2.1.2.4. Target Parameters

**Introduction**

The Target Param classes establish parameters for planetary bodies.

**Target Param Classes**

TargetBaseParam TargetBallParam TargetSpiceParam TargetAsphericalParam

**Class API Documentation**



*Note*

For Doxygen documentation, please see

- TargetBaseParam
- TargetBallParam
- TargetSpiceParam
- TargetAsphericalParam

2.6.2.2.1.2.5. Body Parameters

**Introduction**

The BodyParam class provides all parameters required to specify a single Darts body.

### BodyParam API Documentation



*Note*

For Doxygen documentation, please see: [BodyParam](#)

Body Parameter definition

#### 2.6.2.2.1.2.6. Node Parameters

##### Introduction

The NodeParam class provides all parameters required to specify a single Darts node.

Construction parameters:

param params - Dictionary of param fields and values

param source - The source for all the parameter values

param source\_params - Dictionary of parameter field names and corresponding data source.

param kw - Keywords (all parameter field values can be passed in as keyword/value assignments)

Parameter fields (either in the 'params' dictionary or as a keyword argument):

- **referenceToNode**: The 3-vector to the node frame from the reference frame
- **referenceToNodeQuat**: attitude of node with respect to reference frame
- **referenceFramePath**: if set, the reference frame wrt to home the transform values are specified. If unspecified, the parent body of the node is used as the reference frame.
- **bodyToNode**: The 3-vector to the node frame from the reference frame

### NodeParam API Documentation



*Note*

For Doxygen documentation, please see: [NodeParam](#)

Node Parameter definition

#### 2.6.2.2.1.2.7. Gravity Parameters

##### Introduction

Most gravity parameter classes are derived from GravityBaseParam.

##### GravityBaseParam

The base class for most gravity parameter classes.

##### GravityAsphericalParam

The gravity force actuator attributes for an aspherical gravity model.

### Class API Documentation

- **GravityBaseParam API**



*Note*

For Doxygen documentation, please see: [GravityBaseParam](#)

Gravity Actuator Parameter definition

- **GravityBaseParam API**



*Note*

For Doxygen documentation, please see: [GravityAsphericalParam](#)

Ashperical Gravity Actuator Parameter definition

#### 2.6.2.2.2. Simplified Parameter Object Construction

In DshellCommon, we define the BaseParam class and use it as a parent class for most parameter classes. Originally, constructing a parameter object required specifying the parameter data using a dictionary. In early user scripts, we often used syntax like this for specifying body parameters:

```
'Par1' : ExampleParam(
    params = {
        'length' : 20.0,
        'const1' : 3.0,
    },
    source = None ),
```

The use of dictionary for the params object works fine but can be a little error prone to create or update because of all the string constants.

Note that the suggested corresponding parameter constructor class looks like this (omitting some of the imports and docstrings for brevity):

```
from DshellCommon.params.BaseParam import BaseParam
from DshellCommon.ConfigConstants import UNITLESS

LENGTH = 'length'
CONST1 = 'const1'

class ExampleParam(BaseParam)

    _params = { LENGTH : ParamInfo(required = True,
                                    units = 'm',
                                    description = 'length',
                                    data_type = float),

                CONST1 : ParamInfo(required = False,
                                    units = UNITLESS,
                                    description = 'const 1',
                                    data_type = float,
                                    default_value = 2.0)
            }

    def __init__(self, params=None, source=None,
                 source_params=None, sim=None, **kw):
        """Constructor."""

        params = BaseParam.assignKeywordArgs(params, kw)

        BaseParam.__init__(self, params, source, source_params, sim)
```

By adding the call to `assignKeywordArgs()` as shown on line 25, parameters passed in as keywords (kw) are appended to the param dictionary as if they had been specified there.

Notice that constants used for the parameter fields are defined at the module level (lines 4 and 5.) This makes the code in parameter class definition less problem prone (since we are dealing with constants instead of raw strings).

This approach means the `ExampleParam` invocation above can be re-coded as follows:

```
'Par1' : ExampleParam( length = 20.0,
                       const1 = 3.0,
                       source = None ),
```

As the keyword arguments are processed, each is checked against the known parameters and an error is raised if it is unknown.

Also notice that a default value was defined for 'const1' in line 18 above. If the user had not defined 'const1' here, the default value would have been used (2.0).

This has several advantages over the earlier approach:

- No python dictionary is necessary.
- The parameter constructor is cleaner, shorter, and easier to read.
- Misspellings of parameter names (eg 'length') are caught early in the process.
- This approach should be more robust to errors by new users.

### 2.6.2.2.3. Units

#### 2.6.2.2.3.1. Using units in parameter specifications

All model variables and flows defined in model definition file can specify their units. This includes scratch and state variables, parameters, flowIns and flowOuts. In order to take advantage of this capability, the desired units need to be specified in the .mdl file:

```
[params]

[[angle]]
Type = double
Length = -
Units = rad
Brief description = ""The angle""
```

Note that the **angle** parameter has been specified as radians here. The units information is saved in the model meta data (as a string) and is available for run-time inspection.

In the parameter specification, the units can be used to automatically convert the **angle** parameter to radians:

```

from DshellCommon.units import deg

Library = {
    'State': { 'Attitude':
        AngleParam({ 'angle': 23.0 * deg })
    }
}

```

By multiplying the numerical value with the **deg** object as shown here, the resulting value is converted to radians internally. This ensures that the value stored in the model variable is in the desired units.

Similar conversions can be done for length (meters, feet), force (newtons, lbf), speed (m/s, feet/sec, fps), etc.

To specify a variable without units, use the keyword: 'unitless'.

Notice that each type of unit needed in the file needs to be imported explicitly.

#### 2.6.2.2.3.2. Types of Units Available

The units conversion code is based on the **quantities** package. See <http://packages.python.org/quantities/index.html> for more information.

The **quantities** package supports most common units SI and US Customary Units.

The easiest way to determine if a particular unit definition is supported is to try to import it. For example:

```

>>> from DshellCommon.units import feet
>>> feet
UnitQuantity('feet', 1.0 * ft, 'feet')

```

#### 2.6.2.2.4. Parameter Fields

##### 2.6.2.2.4.1. Overview of Parameter Field Classes

The Dshell parameters are composed of several parameter fields. The most generic of these is ParamInfo. However we now have many new parameter fields that are more specialized for validating input data. The following table lists the available parameter fields and a bit about their behavior. Note that the term "Array" in these class names means that the item is a list (or tuple) of the underlying parameter field types. To determine which arguments each parameter field class supports, see their source code in `DshellCommon/python/params/fields/`

| Field Name   | File                | Notes  |
|--|---------------------|--|
| BooleanParamField (API Docs <BooleanParamField>)               | boolean.py          | Accepts only True or False values  |
| BooleanArrayParamField (API Docs <BooleanArrayParamField>)     | boolean_array.py    | List or tuple of BooleanParamField values  |
| FilenameParamField (API Docs <FilenameParamField>)             | filename.py         | Accepts the full path to a file. By default, the file <b>MUST</b> exist unless the optional constructor argument <b>must_exist</b> is set to False.  |
| FilenameArrayParamField (API Docs <FilenameArrayParamField>)   | filename_array.py   | List or tuple of FilenameParamField values   |
| FloatParamField (API Docs <FloatParamField>)                   | float.py            | Floating point value (floats, doubles, int, long ints). Two options are available to control each field's behavior:<br><br><b>positive_only</b> - Requires the float values to be positive (not zero). Default is False.<br><br><b>positive_or_zero_only</b> - Requires the float values to be zero or positive (non-negative). Default value is False.<br><br><b>minimum</b> - Requires that the float value be greater than or equal to this value.<br><br><b>maximum</b> - Requires that the float value be less than or equal to this value. |
| FloatArrayParamField (API Docs <FloatArrayParamField>)         | float_array.py      | List or tuple of FloatParamField values. The options <b>positive_only</b> , <b>positive_or_zero_only</b> , <b>minimum</b> , and <b>maximum</b> can be given and operate as described for FloatParamField. Another option is available:<br><br><b>max_length</b> - Sets the maximum allowed length of the array.  |
| FloatOrVector3ParamField (API Docs <FloatOrVector3ParamField>) | float_or_vector3.py | Allows a single floating point value or a 3-vector (a list/tuple of three float values or an SOAVector3 object)  |
| GeometryParamField (API Docs <GeometryParamField>)             | geometry.py         | A dictionary of body/part geometry data. Three types are supported: Primitive shapes ('sphere', 'cone', 'cylinder', 'cube/'/box'), known "Topo" objects such as SphericalSun, or a reference to a SimScape store object (note that the existence of the SimScape store directory is checked immediately). This field is used in BodyParam.py for the 'geometry' param field or for 'partGraphics' (which is deprecated; use 'geometry' instead).<br><br>To find out more please visit <a href="#">GeometryParamFieldRef</a> .                    |

| Field Name   | File               | Notes  |
|--|--------------------|--|
| InertiaMatrixParamField (API Docs <InertiaMatrixParamField>)   | inertia_matrix.py  | <p>A 3x3 inertia matrix composed of valid float values. Three options are available to control the behavior of the field:</p> <p><b>epsilon</b> - Specify the epsilon to be used when checking the matrix validity. Default is SOABase.epsilon.</p> <p><b>allow_zero</b> - Allow an inertia matrix of all zeros (for phantom bodies). Default is True.</p> <p><b>allow_semidefinite</b> - Allow semi-definite inertia matrices. Default is True.</p>   |
| IntegerParamField (API Docs <IntegerParamField>)               | integer.py         | <p>Accepts only integer values. Two options are available to control the behavior of the integer param field:</p> <p><b>positive_only</b> - Allow only positive integers (not zero). Default is False.</p> <p><b>positive_or_zero_only</b> - Allow only positive or zero integers (non-negative). Default is False.</p> <p><b>minimum</b> - Requires that the float value be greater than or equal to this value.</p> <p><b>maximum</b> - Requires that the float value be less than or equal to this value.</p> <p>This field also supports the normal <b>choices</b> option to provide a list of allowable integer values.</p> |
| IntegerArrayParamField (API Docs <IntegerArrayParamField>)     | integer_array.py   | <p>List or tuple of IntegerParamField values. The options <b>positive_only</b>, <b>positive_or_zero_only</b>, <b>minimum</b>, and <b>maximum</b> can be given and operate as described for FloatParamField. Another option is available:</p> <p><b>max_length</b> - Sets the maximum allowed length of the array.</p>  |
| JointLimitsParamField (API Docs <JointLimitsParamField>)       | joint_limits.py    | A list or tuple of float values.   |
| ModelEnumParamField (API Docs <ModelEnumParamField>)           | model_enum.py      | <p>A field that contains a value (or list/tuple of values) from a model enum. This field has two required options:</p> <p><b>model_name</b> - The name of the model the enum is defined in</p> <p><b>enum_name</b> - The name of the enum</p> <p>If this field is to be an array of enum values, set the option <b>list_permitted</b> to True. You may also specify the <b>length</b> option to require the list of enum values to be a particular length.</p>   |
| PointerParamField (API Docs <PointerParamField>)               | pointer.py         | Accepts pointer values (long int). The field may be set to 0 if the option <b>allow_zero</b> is set to True.   |
| PointerArrayParamField (API Docs <PointerArrayParamField>)     | pointer_array.py   | List or tuple of PointerParamField values  |
| QuaternionParamField (API Docs <QuaternionParamField>)         | quaternion.py      | Accepts a legal quaternion value either as an SOAQuaternion object or a list/tuple of 3 float values that form a legal unit quaternion.  |
| RotationMatrixParamField (API Docs <RotationMatrixParamField>) | rotation_matrix.py | <p>A valid 3x3 rotation matrix. One option is available to control the behavior of the field:</p> <p><b>epsilon</b> - Specify the epsilon to be used when checking the matrix validity. Default is SOABase.epsilon.</p>  |
| SpiceIDParamField (API Docs <SpiceIDParamField>)               | spice_id.py        | Accepts string Spice ID names or Spice integer ID values.  |
| SpiceIDArrayParamField (API Docs <SpiceIDArrayParamField>)     | spice_id_array.py  | List or tuple of SpiceIDParamField values.   |
| StringParamField (API Docs <StringParamField>)                 | string.py          | Accepts a string value.  |
| StringArrayParamField (API Docs <StringArrayParamField>)       | string_array.py    | List or tuple of StringParamField values   |
| Vector3ParamField (API Docs <Vector3ParamField>)               | vector3.py         | Accepts a list or tuple of 3 float values or an SOAVector3 object.   |
| Vector3ArrayParamField (API Docs <Vector3ArrayParamField>)     | vector3_array.py   | List or tuple of Vector3ParamField values  |

Table: **Parameter Field Classes**

#### 2.6.2.2.4.2. Parameter Field API Documentatin

##### **BooleanParamField API Documentation**

DshellCommon.data\_fields.boolean

##### **BooleanArrayParamField API Documentation**

DshellCommon.data\_fields.boolean\_array

##### **FilenameParamField API Documentation**

DshellCommon.data\_fields.filename

##### **FilenameArrayParamField API Documentation**

DshellCommon.data\_fields.filename\_array

##### **FloatParamField API Documentation**

DshellCommon.data\_fields.float

##### **FloatArrayParamField API Documentation**

DshellCommon.data\_fields.float\_array

##### **FloatOrVector3ParamField API Documentation**

DshellCommon.data\_fields.float\_or\_vector3

##### **GeometryParamField API Documentation**

DshellCommon.data\_fields.geometry

##### **InertiaMatrixParamField API Documentation**

DshellCommon.data\_fields.inertia\_matrix

##### **IntegerParamField API Documentation**

DshellCommon.data\_fields.integer

##### **IntegerArrayParamField API Documentation**

DshellCommon.data\_fields.integer\_array

##### **JointLimitsParamField API Documentation**

DshellCommon.data\_fields.joint\_limits

##### **ModelEnumParamField API Documentation**

DshellCommon.data\_fields.model\_enum

##### **PointerParamField API Documentation**

DshellCommon.data\_fields.pointer

##### **PointerArrayParamField API Documentation**

DshellCommon.data\_fields.pointer\_array

##### **QuaternionParamField API Documentation**

DshellCommon.data\_fields.quaternion

##### **RotationMatrixParamField API Documentation**

DshellCommon.data\_fields.rotation\_matrix

##### **SpiceIDParamField API Documentation**

DshellCommon.data\_fields.spice\_id

##### **SpiceIDArrayParamField API Documentation**

DshellCommon.data\_fields.spice\_id\_array

##### **StringParamField API Documentation**

DshellCommon.data\_fields.string

## StringArrayParamField API Documentation

DshellCommon.data\_fields.string\_array

## Vector3ParamField API Documentation

DshellCommon.data\_fields.vector3

## Vector3ArrayParamField API Documentation

DshellCommon.data\_fields.vector3\_array

### 2.6.2.3. DshellCommon API Reference - Primary Assembly Classes

#### 2.6.2.3.1. Assembly

##### 2.6.2.3.1.1. Class Documentation

### Introduction

The Assembly class is the parent class of all classes defined in DshellCommon.

- Assembly is (swig-wrapped) C++ Assembly class
- Assembly provides a layer over the C++ Assembly class for any needed additional functionality such as:
  - Data structures and functions to track the origins of parameters
  - Functions for checking types of parameter data structures
  - Functions to create new assemblies based on assembly definitions
  - Low-level utilities for providing factory-like conversion of string names to python classes.

### Construction Options

- config - None
- context - None
- signalTies - None
- params - None

### Suggestions for Developers Writing New Classes derived from Assembly

Here are few suggestions for developers when writing new classes derived from Assembly:

- Use ParentClassX.func() syntax rather than super.func() syntax for invoking members of parent classes.
- At the end of each class definition file, include a line like:

```
NewClass.register()
```

This registers the new class with the class factory managed by the Assembly class. Also, add a line to import the new file in the appropriate *init.py* file.

- **When creating a group of related classes**
  - Put the related classes into a separate directory (as is done with gravity, imu, and time).
  - Derive the new base class for the group of related classes from Assembly.
  - Factor any common code into the new base class.
  - Create a '*init.py*' file in the new directory and add lines in it to import each instantiable class. For example, see 'DshellCommon/python/assemblies/*init.py*'.
  - Ensure all functions in classes which are derived from the new base class call their parent's versions of the same function (since it may be desirable to add/modify code in the new base class without updating all derived classes).

### Related Regression Tests

- Simple stand-alone test for BaseAssembly class

### Assembly Class API Documentation



#### Note

For Doxygen documentation, please see: `Assembly<Assembly::Assembly>`

Manages a group of Model, DartsGraph and Signal objects.

C++ includes: Assembly.h

### 2.6.2.3.2. SimulationAssembly

#### 2.6.2.3.2.1. Introduction

The SimulationAssembly is the top-level assembly for simulations.

The only functionality that SimulationAssembly adds is to add the 'adv\_dyn' flow model to advance the dynamics simulation.

#### 2.6.2.3.2.1.1. The Role of SimulationAssembly and Dvar

The SimulationAssembly class has two significant and related roles in simulations. Let the instance of the SimulationAssembly class that the SimulationExecutive creates be called **top\_asm** in the following discussion. **top\_asm** contains all the simulations created for the simulation. Since all assemblies are created hierarchically and are contained directly or indirectly in *top\_asm*, the implication is that full Dvar names of any assembly or model-related object in the system has 'Assemblies', the Dvar name of **top\_asm** in simulations using SimulationExecutive.

For instance, in the regression test `../regtests/test_VehicleAssembly_target`, the full Dvar 'specString' of the gravitational acceleration is:

```
' .Dshell.Assemblies.SC1.grav.models.grav.flowOut.lin_accel'
```

where:

- **Dshell** - is the name of the top-level Dvar branch containing all Dvar data related to models, signals, and assemblies. Note that there is a parallel branch 'mbody' that contains all multibody bodies, hinges, and nodes.
- **Assemblies** - is the Dvar name of the top-level instance of the SimulationAssembly class. The name 'Assemblies' is defined by the SimulationExecutive object when creating the top-level SimulationAssembly.
- **SC1** - is the name of the spacecraft (VehicleAssembly)
- **grav** - is the name of the gravity assembly
- **models** - is the name of the Dvar branch for all model-related Dvars for the gravity assembly.
- **grav** - is the name of the gravity model
- **flowOut** - is the Dvar branch for all 'grav' model flowOut's
- **lin\_accel** - is the name of the Dvar leaf for the linear acceleration flowIn

#### 2.6.2.3.2.2. Construction Options

SimulationAssembly has no config, context, signalTies, or params options.

The SimulationAssembly class will never need to be created by the end user or developer since it is created automatically by the SimulationExecutive.

#### 2.6.2.3.2.3. Related Regression Tests

#### 2.6.2.3.2.4. SimulationAssembly API Documentation



#### Note

For Doxygen documentation, please see: `SimulationAssembly<SimulationAssembly::SimulationAssembly>`

The following API documentation includes many functions from parent classes that end users should not use.

SimulationAssembly - The top-level assembly class for the simulation

#### 2.6.2.3.3. VehicleAssembly

##### INTRODUCTION

**VehicleAssembly** is the base vehicle assembly class.

The VehicleAssembly models a vehicle such as spacecraft, ground vehicle, or aerial vehicle.

The VehicleAssembly class is derived from the C++ Assembly class will be the parent class of all vehicular assemblies.

The VehicleAssembly is very flexible. It can be used to model

- Simple one-body vehicles
- Vehicles with multiple connected bodies
- Vehicles that can separate
- Vehicles with parts that can be detached or ejected

The key idea here is that any vehicle or part of a vehicle that can potentially be disconnected from its parent vehicle should be modeled as a VehicleAssembly in its own right.

In the initial configuration, VehicleAssembly vehicles are attached by specifying them hierarchically. In other words, if a vehicle has part that could be detached, the detachable part should be modeled as a VehicleAssembly in its own right and defined in the 'assemblies' list of its parent vehicle.

Each vehicle has a 'root' body. This is the first body in the multibody tree/chain in the vehicle.

Each vehicle also has a 'base' body. This is the body the main body for the vehicle. In many cases this is the same as the root body. But if the main body of the vehicle is modeled as a translational body coincident with a ball joint, then the 'base' body could be the second (rotational) body since it has both translational and rotational degrees of freedom.

A few notes about VehicleAssemblies

- If a vehicle has a gravity model, it must be associated with the 'root' body of the vehicle.
- Any attachment to a vehicle must be an attachment to its 'root' body. The 'root' body of a vehicle may be attached to any body of its parent vehicle.
- If gravity is included in the simulation, each VehicleAssembly must include appropriate gravity models. The gravity models for child vehicles will be inactive initially since the top-most parent vehicle's gravity model will provide gravity for all child vehicles. When a vehicle separates from its parent, its separate gravity model will be activated.

#### CONSTRUCTING VEHICLEASSEMBLYS



- **config** - None

VehicleAssembly has one required config setting:

- **'basename'**

This is the name of the first body in the vehicle.

A vehicle also has a 'root' body, which may or may not be the 'base' body. The 'root' body is actually the first body of the spacecraft in terms of its multibody tree.

In some cases, it is convenient to have a first 'pseudo' body (that is a 3DOF translational body with zero mass and inertia) followed by the *real* first body of the vehicle (that has non-zero mass and inertia and is attached to the 'root' body via a ball joint). In this case, the 'root' body is the 3DOF body and the 'base' body is the second body which is, in a sense, the first *real* body of the vehicle.

VehicleAssembly has one optional config setting:

- **'bodygraph'**

The 'bodygraph' describes the bodies in the vehicle and their connectivity. Note that if there is only one body in the vehicle, it is not necessary to specify the 'bodygraph' since it is redundant.

The 'bodygraph' is a dictionary with two parts:

- 'edges' - a dictionary describing multibody connectivity information in the following form of key/value pairs:
- 'verts' - THIS IS HERE FOR LEGACY REASONS. PLEASE DO NOT SPECIFY THIS. A list of the bodies in the spacecraft IN ORDER: parents then children.  
<childBodyName> : <parentBodyName>

- **context**

VehicleAssembly has one optional context setting:

- **'parentBody'**

If 'parentBody' is specified, the 'root' body of a child vehicle will be attached to the the body with the specified name in the parent vehicle.

- **signalTies** - None

- **param**

VehicleAssembly has two required parameters:

- **'Bodies'**

This should point to a dictionary of `BodyParam` parameter class objects. It must include entries for all bodies in the assembly. The 'Bodies' parameter data is used to initialize the multibody/mass properties of the bodies composing the vehicle.

VehicleAssembly has two optional parameters:

- **'Position'**

This should point to a `StatePosVelCartesianParam` parameter class object (or any parameter class object derived from `StatePosVelBaseParam`). It is used to initialize the initial position of the vehicle.

- **'Attitude'**

This should point to a `StateAttitudeFPAlignParam` parameter class object (or any parameter class object derived from `StatePosVelBaseParam`). It is used to initialize the initial attitude of the vehicle.



*Note*

The 'Position' and 'Attitude' parameters are optional but must appear together.

#### 2.6.2.3.3.1. Example Construction Syntax

```
config = {
    'SC1' : {
        'class' : 'VehicleAssembly',
        'basename' : 'Probe',
        'params' : { 'Bodies' : bparams['SC1']['Bodies'] },
    }
}
```

PYTHON

or:

```

config = {
    'SC1' : {
        'class' : 'VehicleAssembly',
        'basename' : 'CapsuleBase',
        'bodygraph' : {
            'verts' : ['CapsuleBase0', 'CapsuleBase', 'FuelTank'],
            'edges' : {'CapsuleBase' : 'CapsuleBase0',
                      'FuelTank' : 'CapsuleBase'}},
        'params' : {
            'Bodies' : bparams['SC1']['Bodies'],
            'Position' : states.Library['MSL 05-19']['Position'],
            'Attitude' : states.Library['MSL 05-19']['Attitude'],
        },
        'assemblies' : { }
    }
}

```

#### 2.6.2.3.3.2. VehicleAssembly Usage

##### 2.6.2.3.3.2.1. Signal Information

There are no signals in this assembly.

##### 2.6.2.3.3.2.2. Param Class Information

###### 2.6.2.3.3.2.2.1. UserClockParam

Describes the base time characteristics (at sim time=0.0) of user defined clocks

##### Required Parameters

None

##### Optional Parameters

| Name            | Type   | Length | Quantity | Units | Description                           |
|-----------------|--------|--------|----------|-------|---------------------------------------|
| <i>baseTime</i> | string | 1      | N/A      |       | time value of clock at sim time = 0.0 |

##### 2.6.2.3.3.2.2.2. StateAttitudeBaseParam

State attitude knowledge/delivery specification for the start of a simulation and attitude rate. When using this parameter class in a VehicleAssembly, the inertial frame in the descriptions for *initQ* and *initOmega* refers to the Planet-Centered Inertial (PCI) frame. When this class is used with the *stateInit* utility, the inertial frame refers to the frame provided by the assembly passed in as an argument.

##### Required Parameters

| Name              | Type          | Length | Quantity       | Units | Description   |
|-------------------|---------------|--------|----------------|-------|---|
| <i>initQ name</i> | double string | 4 1    | Quaternion N/A |       | initial attitude quaternion from body frame to inertial frame either directly specified or derived from angle data name of instance |

##### Optional Parameters

| Name                  | Type    | Length | Quantity        | Units | Description  |
|-----------------------|---------|--------|-----------------|-------|--|
| <i>initAdjustquat</i> | double  | 4      | Quaternion      |       | attitude quaternion adjustment                           |
| <i>initOmega</i>      | vector3 | 1      | AngularVelocity | rad/s | body angular velocity with respect to the inertial frame |
| <i>initOmegaBody</i>  | vector3 | 1      | AngularVelocity | rad/s | body angular velocity with respect to the body frame     |

##### 2.6.2.3.3.2.2.3. BodyParam

The **BodyParam** class provides all parameters required to specify a single Darts body.

##### Required Parameters

| Name        | Type   | Length | Quantity | Units | Description   |
|-------------|--------|--------|----------|-------|---------------|
| <i>mass</i> | double | 1      | Mass     | kg    | The body mass |

##### Optional Parameters

| Name            | Type    | Length | Quantity | Units | Description                                      |
|-----------------|---------|--------|----------|-------|--|
| <i>bodyToCM</i> | vector3 | 1      | Length   | m     | The 3-vector to the body CM from the body frame. |

| Name  | Type   | Length                              | Quantity  | Units | Description  |
|---|--|-------------------------------------|---|-------|--|
| <i>bodyToJoint</i><br><i>bodyToJointQuat</i> <i>cmInertia</i><br><i>description</i> <i>gearRatio</i><br><i>geometry</i>   | vector3 double double<br>string double \{\}                      | 1 4 None 1 1 None                   | Length Quaternion<br>MomentsOfInertia N/A<br>Dimensionless N/A                        | m     | The 3-vector from the body frame to the body hinge The quaternion from the body frame to the prode for the body hinge The 3x3 body inertia matrix about the body CM (do not specify both inertia and cmInertia) DEPRECATED: Body description. IGNORED. Gear ratio for PIN joints Geometry to be loaded into DScene.  |
| <i>inbToJoint</i> <i>inbToJointQuat</i><br><i>inertia</i> <i>jointAxes</i> <i>jointLimits</i><br><i>jointType</i><br><i>negativeIntegralSense</i><br><i>partGraphics</i> <i>prescribed</i><br><i>prescribedType</i> <i>subhinge</i> | vector3 double double<br>vector3 double string bool \{\}<br>bool | 1 4 None ** 1 1 None * None<br>None | Length Quaternion<br>MomentsOfInertia<br>Dimensionless N/A N/A N/A<br>N/A N/A N/A N/A | m     | The 3-vector from the inboard body frame to the body hinge The quaternion from the inboard body frame to the body hinge The 3x3 body inertia matrix about the body frame (do not specify both inertia and cmInertia) The hinge rotation/translation axis Limits for range of joint generalized coordinate [min, max] The hinge type (string). Supported: BALL, COMPOSITE-TRANSLATIONAL, FULL6DOF, FULL6DOF_INERTIAL, GIMBAL, LOCKED, PIN, PLANAR, SLIDER, TRANSLATIONAL, UJOINT, CUSTOM Body products of inertia values expressed in NEGATIVE inertia integral sense (defaults to True) DEPRECATED: Geometry to be loaded into DScene. Whether the joint is prescribed or free (scalar boolean or array of booleans for each subhinge) DEPRECATED: Type of prescribed joint. WARNING: Not permitted in Ndarts Subhinge specification for custom hinge. |

#### 2.6.2.3.3.2.2.4. StatePosVelBaseParam

State position (and velocity) knowledge/delivery specification for the start of a simulation.

The refVertical (see ref Reference\_Datums\_Verticals\_section) is used in the computation of initial attitude from flight-path and aero angle quantities.

#### Required Parameters

| Name        | Type   | Length | Quantity | Units | Description      |
|-------------|--------|--------|----------|-------|------------------|
| <i>name</i> | string | 1      | N/A      |       | name of instance |

#### Optional Parameters

| Name               | Type   | Length | Quantity | Units | Description   |
|--------------------|--------|--------|----------|-------|---|
| <i>refVertical</i> | string | 1      | N/A      |       | direction defining vertical for bank angle 'RADIAL', ELLIPSOID_VERTICAL, ELLIPTIC_NORMAL', (defaults to RADIAL) |

2.6.2.3.3.2.2.5. NodeParam

The **NodeParam** class provides all parameters required to specify a single Darts node.

Required Parameters

| Name              | Type    | Length | Quantity | Units | Description   |
|-------------------|---------|--------|----------|-------|---|
| <i>bodyToNode</i> | vector3 | 1      | Length   | m     | The 3-vector to the node frame from the body frame. |

Optional Parameters

| Name                  | Type   | Length | Quantity   | Units | Description                                  |
|-----------------------|--------|--------|------------|-------|--|
| <i>bodyToNodeQuat</i> | double | 4      | Quaternion |       | attitude of node with respect to body frame. |

2.6.2.3.3.2.2.6. SpringDamperParam

The attributes for a tether modeled by SpringDamper actuator

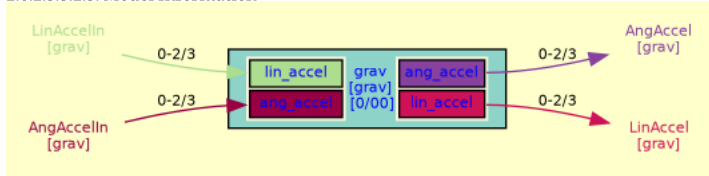
Required Parameters

| Name                 | Type                 | Length | Quantity    | Units | Description  |
|----------------------|----------------------|--------|-------------|-------|--|
| <i>K node1 node2</i> | double string string | 1 1 1  | N/A N/A N/A |       | spring constant for the spring<br>The name of the node on one end of the spring<br>The name of the node on the other end of the spring |

Optional Parameters

| Name                     | Type          | Length | Quantity | Units | Description  |
|--------------------------|---------------|--------|----------|-------|--|
| <i>C unsprung_length</i> | double double | 1 1    | N/A N/A  |       | damping constant for the spring<br>Unsprung length of the spring |

2.6.2.3.3.2.3. Model Information



2.6.2.3.3.2.3.1. GeneralGravity

Applies commanded gravity acceleration on the s/c

- *Keywords:* "Gravity!Commanded"
- *Class:* actuator

Author: David Henriquez / Garrett Sohl (converted to flowIn/Outs) The GeneralGravity actuator model continuously applies the linear and angular acceleration on the s/c. By default, the gravity acceleration vectors are initialized as zero vectors at model instantiation. The GeneralGravity model flowIns are *lin\_accel*[3] and *ang\_accel*[3], which are the commanded linear and angular accelerations, respectively. The GeneralGravity model states are *lin\_accel*[3] and *ang\_accel*[3], which are used to store the commanded linear and angular accelerations, respectively. The GeneralGravity model flowOut is the last commanded acceleration vectors. And the format of the flowOut is also *lin\_accel*[3] and *ang\_accel*[3].

FlowIns

| Name             | Type   | Length | Quantity            | Units   | Value           | Description      |
|------------------|--------|--------|---------------------|---------|-----------------|------------------|
| <i>ang_accel</i> | double | 3      | AngularAcceleration | rad/s^2 | [0.0, 0.0, 0.0] | <i>ang_accel</i> |
| <i>lin_accel</i> | double | 3      | Acceleration        | m/s^2   | [0.0, 0.0, 0.0] | linear accel     |

Parameters

None

Continuous States

None

Scratch

None

States

| Name             | Type   | Length | Quantity            | Units   | Value           | Description   |
|------------------|--------|--------|---------------------|---------|-----------------|---------------|
| <i>ang_accel</i> | double | 3      | AngularAcceleration | rad/s^2 | [0.0, 0.0, 0.0] | angular accel |
| <i>lin_accel</i> | double | 3      | Acceleration        | m/s^2   | [0.0, 0.0, 0.0] | linear accel  |

FlowOuts

| Name             | Type   | Length | Quantity            | Units   | Value           | Description                        |
|------------------|--------|--------|---------------------|---------|-----------------|------------------------------------|
| <i>ang_accel</i> | double | 3      | AngularAcceleration | rad/s^2 | [0.0, 0.0, 0.0] | angular gravitational acceleration |
| <i>lin_accel</i> | double | 3      | Acceleration        | m/s^2   | [0.0, 0.0, 0.0] | linear gravitational acceleration  |

#### 2.6.2.3.3.3. Related Regression Tests

#### 2.6.2.3.3.4. VehicleAssembly tests

- One vehicle and one body, no gravity, manual (1v1b)
- One vehicle with three bodies, no gravity, manual (1v3b)
- Two vehicles with one body each, no gravity, manual (2v1b)

#### 2.6.2.3.3.5. VehicleAssembly2 tests

- One vehicle and 3 bodies, no gravity, regular
- One vehicle with three bodies, with gravity, regular
- Two vehicles with 2+1 bodies, with gravity, regular
- Two vehicles with 2+2 bodies, with gravity, regular

#### 2.6.2.3.3.6. VehicleAssembly with target body tests

- VehicleAssembly with 3 bodies and a planetary target body, simple gravity

#### 2.6.2.3.3.7. VehicleAssembly with aspherical gravity tests

- VehicleAssembly with 3 bodies and a planetary target body, aspherical gravity

#### 2.6.2.3.3.8. VehicleAssembly with Spice tests

- VehicleAssembly with 3 bodies and a planetary target body with spice, simple gravity

#### 2.6.2.3.3.9. VehicleAssembly with attachment/detachment tests

- VehicleAssembly with 2 bodies and a separable probe body. Detach the probe and lock it (freeze it) with respect to the inertial frame
- VehicleAssembly with 2 bodies and a separable probe body. Detach the probe and let it fly with respect to the inertial frame
- VehicleAssembly with 2 bodies and a separable probe body. Detach the probe and lock it to a rotating planetary body (PCR)
- VehicleAssembly with 2 bodies and a separable probe body. Detach the probe and let it fly freely with respect to a rotating planetary body (PCR)
- Create 2 vehicles. In the simulation one vehicle attaches to the other vehicle to roughly simulate docking a vehicle to the space station

#### 2.6.2.3.3.10. VehicleAssembly with FSM tests

- Create a vehicle with 2 attached bodies: a service module and command module. Separate the the two bodies during the simulation using a finite state machine

#### 2.6.2.3.3.11. VehicleAssembly Class API Documentation



#### Note

For Doxygen documentation, please see: [VehicleAssembly](#)

([https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/DshellCommon/html/classpython\\_1\\_1assemblies\\_1\\_1VehicleAssembly\\_1\\_1VehicleAssembly.html](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/DshellCommon/html/classpython_1_1assemblies_1_1VehicleAssembly_1_1VehicleAssembly.html))

#### 2.6.2.4. DshellCommon API Reference - Other Assembly Classes

##### 2.6.2.4.1. ArmAssembly

###### 2.6.2.4.1.1. Class Documentation

An arm assembly class.

###### 2.6.2.4.1.2. Usage Scenario

This is a simple usage of the Arm Assembly. Start with the standard imports and import the Assembly. Create the Sim with SimulationExecutive.

#### ‘ArmAssembly’ test (with no external disturbances)

- ▶ [Click to see the DshellCommon/usage/usage\\_ArmAssembly.py script](#)

###### 2.6.2.4.1.3. Usage

###### 2.6.2.4.1.3.1. Introduction

An arm assembly class.

###### 2.6.2.4.1.3.2. Signal Information

There are no signals in this assembly.

###### 2.6.2.4.1.3.3. Param Class Information

###### 2.6.2.4.1.3.3.1. BaseMotorParam

“ The attributes for the link assembly.

#### Required Parameters

None

#### Optional Parameters

| Name        | Type   | Length | Quantity | Units | Description            |
|-------------|--------|--------|----------|-------|------------------------|
| <i>Type</i> | string | 1      | N/A      |       | String parameter field |

#### 2.6.2.4.1.3.3.2. UserInputIntParam

UserInputIntParam parameters.

Required Parameters

| Name        | Type | Length | Quantity | Units | Description                      |
|-------------|------|--------|----------|-------|----------------------------------|
| <i>name</i> | int  | *      | N/A      |       | The user input real number array |

Optional Parameters

None

2.6.2.4.1.3.3.3. UserInputRealParam

“UserInputRealParam parameters.

Required Parameters

| Name        | Type   | Length | Quantity | Units | Description                      |
|-------------|--------|--------|----------|-------|----------------------------------|
| <i>name</i> | double | *      | N/A      |       | The user input real number array |

Optional Parameters

None

2.6.2.4.1.3.3.4. PyExtDistModuleNameParam

“PyExtDistModuleNameParam parameters.

Required Parameters

| Name        | Type   | Length | Quantity | Units | Description                       |
|-------------|--------|--------|----------|-------|-----------------------------------|
| <i>name</i> | string | 1      | N/A      |       | The PyExtDistModuleNameParam name |

Optional Parameters

None

2.6.2.4.1.3.3.5. NodeParam

“The **NodeParam** class provides all parameters required to specify a single Darts node.

Required Parameters

| Name              | Type    | Length | Quantity | Units | Description   |
|-------------------|---------|--------|----------|-------|---|
| <i>bodyToNode</i> | vector3 | 1      | Length   | m     | The 3-vector to the node frame from the body frame. |

Optional Parameters

| Name                  | Type   | Length | Quantity   | Units | Description                                  |
|-----------------------|--------|--------|------------|-------|--|
| <i>bodyToNodeQuat</i> | double | 4      | Quaternion |       | attitude of node with respect to body frame. |

2.6.2.4.1.3.3.6. PyExtDistFunctionNameParam

PyExtDistFunctionNameParam parameters.

Required Parameters

| Name        | Type   | Length | Quantity | Units | Description                         |
|-------------|--------|--------|----------|-------|-------------------------------------|
| <i>name</i> | string | 1      | N/A      |       | The PyExtDistFunctionNameParam name |

Optional Parameters

None

2.6.2.4.1.3.3.7. BodyParam

The **BodyParam** class provides all parameters required to specify a single Darts body.

Required Parameters

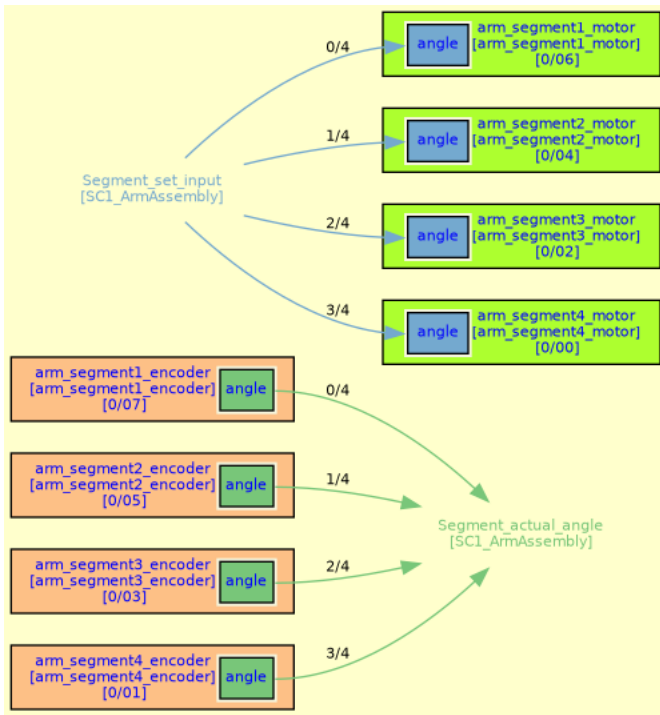
| Name        | Type   | Length | Quantity | Units | Description   |
|-------------|--------|--------|----------|-------|---------------|
| <i>mass</i> | double | 1      | Mass     | kg    | The body mass |

Optional Parameters

| Name            | Type    | Length | Quantity | Units | Description                                      |
|-----------------|---------|--------|----------|-------|--|
| <i>bodyToCM</i> | vector3 | 1      | Length   | m     | The 3-vector to the body CM from the body frame. |

| Name  | Type   | Length                              | Quantity  | Units | Description  |
|---|--|-------------------------------------|---|-------|--|
| <i>bodyToJoint</i><br><i>bodyToJointQuat</i> <i>cmInertia</i><br><i>description</i> <i>gearRatio</i><br><i>geometry</i>   | vector3 double double<br>string double \{\}                      | 1 4 None 1 1 None                   | Length Quaternion<br>MomentsOfInertia N/A<br>Dimensionless N/A                        | m     | The 3-vector from the body frame to the body hinge The quaternion from the body frame to the pnode for the body hinge The 3x3 body inertia matrix about the body CM (do not specify both inertia and cmInertia) DEPRECATED: Body description. IGNORED. Gear ratio for PIN joints Geometry to be loaded into DScene.  |
| <i>inbToJoint</i> <i>inbToJointQuat</i><br><i>inertia</i> <i>jointAxes</i> <i>jointLimits</i><br><i>jointType</i><br><i>negativeIntegralSense</i><br><i>partGraphics</i> <i>prescribed</i><br><i>prescribedType</i> <i>subhinge</i> | vector3 double double<br>vector3 double string bool \{\}<br>bool | 1 4 None ** 1 1 None * None<br>None | Length Quaternion<br>MomentsOfInertia<br>Dimensionless N/A N/A N/A<br>N/A N/A N/A N/A | m     | The 3-vector from the inboard body frame to the body hinge The quaternion from the inboard body frame to the body hinge The 3x3 body inertia matrix about the body frame (do not specify both inertia and cmInertia) The hinge rotation/translation axis Limits for range of joint generalized coordinate [min, max] The hinge type (string). Supported: BALL, COMPOSITE-TRANSLATIONAL, FULL6DOF, FULL6DOF_INERTIAL, GIMBAL, LOCKED, PIN, PLANAR, SLIDER, TRANSLATIONAL, UJOINT, CUSTOM Body products of inertia values expressed in NEGATIVE inertia integral sense (defaults to True) DEPRECATED: Geometry to be loaded into DScene. Whether the joint is prescribed or free (scalar boolean or array of booleans for each subhinge) DEPRECATED: Type of prescribed joint. WARNING: Not permitted in Ndarts Subhinge specification for custom hinge. |

2.6.2.4.1.3.4. Model Information



#### 2.6.2.4.1.4. Related Regression Tests

**`ArmAssembly` test (with no external disturbances)**

▶ [Click to see the DshellCommon/test/test\\_Ndarts/test\\_ArmAssembly/script\\_NoExtDist.py script](#)

**`ArmAssembly` test (with External Disturbance Python Model)**

▶ [Click to see the DshellCommon/test/test\\_Ndarts/test\\_ArmAssembly/script\\_ExtPyDist.py script](#)

**`LinkAssembly` test (with spring damper motors)**

▶ [Click to see the DshellCommon/test/test\\_Ndarts/test\\_ArmAssembly/script\\_springDampers.py script](#)

#### 2.6.2.4.1.5. Class API Documentation



##### Note

For Doxygen documentation, please see: [Arm Assembly<assemblies::ArmAssembly::ArmAssembly](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/DshellCommon/html/classpython_1_1assemblies_1_1ArmAssembly_1_1ArmAssembly.html)  
([https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/DshellCommon/html/classpython\\_1\\_1assemblies\\_1\\_1ArmAssembly\\_1\\_1ArmAssembly.html](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/DshellCommon/html/classpython_1_1assemblies_1_1ArmAssembly_1_1ArmAssembly.html))

#### 2.6.2.4.2. AsphericalGravityActuatorAssembly

##### 2.6.2.4.2.1. Class Documentation

An aspherical gravity force sensor model.

##### 2.6.2.4.2.1.1. Introduction

The AsphericalGravityActuatorAssembly

##### 2.6.2.4.2.1.2. Construction Options

- config - GeneralGravityActuatorAssembly has one optional config setting. Inherited from its base class, GravityActuatorAssembly:
  - **'node'**

The name of the node that the gravity is to be applied to. Defaults to the name of the assembly if not provided.
- context - GeneralGravityActuatorAssembly has one required config setting. Inherited from its base class, GravityActuatorAssembly:
  - **'body'**

The name of the body that the gravity is to be applied to.
- signalTies - None
- param - AsphericalGravityActuatorAssembly has two required parameters and one optional parameter; inherited from its base class, GravityActuatorAssembly:
  - **'Node'**

[Optional] This should point to a `NodeParam` parameter class object. This is usually stored in a file in the library of your run script such as: `./library/bodies.py`. This information is used to initialize the relative position of the node with respect to the body reference coordinates.
  - **'Model'**

This should point to a `GravityBaseParam` parameter class object. This is usually stored in a file in the library of your run script such as: `./library/actuators.py`. This information is used to initialize the gravity model constants.
  - **'Target'**



This should point to a `TargetAsphericalParam` parameter class object. This is usually stored in a file in the library of your run script such as `./library/targets.py`. This information is used to initialize the gravity model constants.

#### 2.6.2.4.2.1.3. Example Construction Syntax

```

config = {
  'SC1' : {
    'class' : 'VehicleAssembly',
    ...
    'assemblies' : {
      'grav' : {
        'class' : 'AsphericalGravityActuatorAssembly',
        'context' : { 'body' : 'CapsuleBase0' },
        'params' : { 'Model' : actuators.Library['Gravity']['Standard'],
                    'Target' : targets.Library['Mars']
                  }
        }
      }
    },
    ...
  }
}

```

This configuration block would normally appear in the 'assemblies' section for a VehicleAssembly.

#### 2.6.2.4.2.1.4. Signals

##### FlowIns

None.

##### FlowOuts

The `AsphericalGravityActuatorAssembly` has two flowout signals for user inspection of the applied gravity force and torque:

- **LinAccel** - 3-vector of linear gravitational acceleration
- **AngAccel** - 3-vector of rotational gravitational acceleration

#### 2.6.2.4.2.2. Usage

##### 2.6.2.4.2.2.1. Introduction

“An aspherical gravity force sensor model.

##### 2.6.2.4.2.2.2. Signal Information

| Signal Name     | Type   | Length | Quantity            | Units   | Description  |
|-----------------|--------|--------|---------------------|---------|--|
| <i>AngAccel</i> | double | 3      | AngularAcceleration | rad/s^2 | Gravitational angular acceleration from GeneralGravityModel grav |
| <i>LinAccel</i> | double | 3      | Acceleration        | m/s^2   | Gravitational linear acceleration from GeneralGravityModel grav  |

#### 2.6.2.4.2.2.3. Param Class Information

##### 2.6.2.4.2.2.3.1. NodeParam

“The `NodeParam` class provides all parameters required to specify a single Darts node.

##### Required Parameters

| Name              | Type    | Length | Quantity | Units | Description   |
|-------------------|---------|--------|----------|-------|---|
| <i>bodyToNode</i> | vector3 | 1      | Length   | m     | The 3-vector to the node frame from the body frame. |

##### Optional Parameters

| Name                  | Type   | Length | Quantity   | Units | Description                                  |
|-----------------------|--------|--------|------------|-------|--|
| <i>bodyToNodeQuat</i> | double | 4      | Quaternion |       | attitude of node with respect to body frame. |

##### 2.6.2.4.2.2.3.2. GravityAsphericalParam

“The gravity force actuator attributes for an aspherical gravity model.

##### Required Parameters

| Name                  | Type       | Length | Quantity          | Units | Description   |
|-----------------------|------------|--------|-------------------|-------|---|
| <i>J2 UseGradient</i> | double int | 1 1    | Dimensionless N/A |       | Unnormalized gravitational J2 term of target gravity flag to use gravity gradient: - 0/False no gradient, 1/True - use gradient |

##### Optional Parameters

| Name            | Type          | Length | Quantity                               | Units          | Description   |
|-----------------|---------------|--------|--|----------------|---|
| <i>BodyName</i> | string        | 1      | N/A                                    |                | Name of the attracting body   |
| <i>GJ3</i>      | double double | 1 1    | GravitationalConstant<br>Dimensionless | $m^3/(kg^s^2)$ | universal gravitational constant Unnormalized gravitational J3 term of target gravity |
| <i>Rgrav</i>    | double        | 1      | Length                                 | m              | Reference radius used in gravity model of target                                      |

#### 2.6.2.4.2.2.3.3. TargetBaseParam

• This class provides the parameters used to describe a central body or target.

This is the base class for all Target parameter classes.

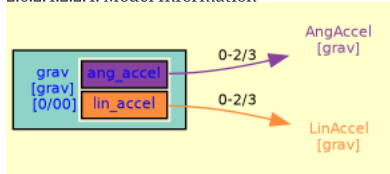
#### Required Parameters

| Name                | Type          | Length | Quantity        | Units | Description   |
|---------------------|---------------|--------|-----------------|-------|---|
| <i>mass name</i>    | double string | 1 1    | Mass N/A        | kg    | Mass of target Name of the target body  |
| <i>rotationRate</i> | double        | None   | AngularVelocity | rad/s | Rotation rate of the target. May be a single float or three element list of floats. |

#### Optional Parameters

| Name                           | Type          | Length | Quantity                       | Units | Description  |
|--------------------------------|---------------|--------|--------------------------------|-------|--|
| <i>eccentricity flattening</i> | double double | 1 1    | Dimensionless<br>Dimensionless |       | Eccentricity of the target body ellipsoid (optional)<br>Ellipsoidal flattening of target body (optional) |
| <i>radiusEquator</i>           | double        | 1      | Length                         | m     | Equatorial radius of target body   |
| <i>radiusPole</i>              | double        | 1      | Length                         | m     | Polar radius of target body (optional)   |
| <i>rotationAngle</i>           | double        | 1      | Angle                          | rad   | Rotation angle of the target   |

#### 2.6.2.4.2.2.4. Model Information



#### 2.6.2.4.2.2.4.1. AsphericalGravity

- *Keywords:* "Gravity!J2 term" "Gravity!J3 term"
- *Class:* actuator

Author: Bryan Martin The AsphericalGravity actuator is a gravity model which inherits all properties of the PointMassGravity model and adds aspherical compensation terms to it. Currently, only the J2 and J3 zonal harmonics are supported. (See Wertz, "Spacecraft Attitude Determination and Control")

FlowIns  
None

#### Parameters

| Name                     | Type        | Length | Quantity | Units | Value                | Description                                      |
|--------------------------|-------------|--------|----------|-------|----------------------|--|
| <i>BodyMass BodyUuid</i> | double uint | 1 1    | Mass N/A | kg    | 6.41430089366e+23 45 | Mass of target body uuid                         |
| <i>EquatRadius</i>       | double      | 1      | Length   | m     | 3396190.0            | Reference radius used in gravity model of target |

| Name                      | Type                     | Length  | Quantity  | Units          | Value  | Description  |
|---------------------------|--------------------------|---------|---|----------------|--|--|
| <i>GJ2 J3 UseGradient</i> | double double double int | 1 1 1 1 | GravitationalConstant<br>Dimensionless<br>Dimensionless N/A | $m^3/(kg*s^2)$ | 6.67259e-11<br>0.00195639057766<br>3.14647623011e-05 0 | universal gravitational constant Unnormalized<br>gravitational J2 term of target gravity Unnormalized<br>gravitational J3 term of target gravity flag to use gravity gradient: -0/False no gradient, 1/True - use gradient |

Continuous States  
None

Scratch

| Name         | Type   | Length | Quantity | Units | Value                          | Description                 |
|--------------|--------|--------|----------|-------|--------------------------------|-----------------------------|
| <i>Accel</i> | double | 6      | Mixed    |       | [0.0, 0.0, 0.0, 0.0, 0.0, 0.0] | acceleration due to gravity |

States

| Name                | Type   | Length | Quantity | Units | Value           | Description                           |
|---------------------|--------|--------|----------|-------|-----------------|---------------------------------------|
| <i>BodyPosition</i> | double | 3      | Length   | m     | [0.0, 0.0, 0.0] | body inertial position vector (J2000) |
| <i>BodyVelocity</i> | double | 3      | Velocity | m/s   | [0.0, 0.0, 0.0] | body inertial velocity vector (J2000) |

FlowOuts

| Name             | Type   | Length | Quantity            | Units     | Value           | Description                        |
|------------------|--------|--------|---------------------|-----------|-----------------|------------------------------------|
| <i>ang_accel</i> | double | 3      | AngularAcceleration | $rad/s^2$ | [0.0, 0.0, 0.0] | angular gravitational acceleration |
| <i>lin_accel</i> | double | 3      | Acceleration        | $m/s^2$   | [0.0, 0.0, 0.0] | linear gravitational acceleration  |

2.6.2.4.2.2.5. Related Regression Tests

**VehicleAssembly` class with one vehicle and three bodies (1v3b) and gravity**

► [Click to see the DshellCommon/includes/test/test\\_Ndarts/test\\_VehicleAssembly\\_gravity/script1.py script](#)

2.6.2.4.2.3. AsphericalGravityActuatorAssembly Class API Documentation



*Note*

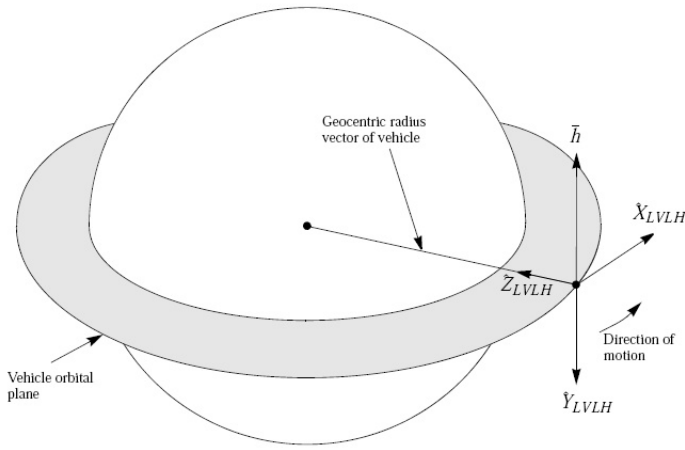
For Doxygen documentation, please see: [AsphericalGravityActuatorAssembly<AsphericalGravityActuatorAssembly::AsphericalGravityActuatorAssembly](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/DshellCommon/html/classpython_1_1assemblies_1_1gravity_1_1AsphericalGravityActuatorAssembly_1_1AsphericalGravityActuatorAssembly.html)  
([https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/DshellCommon/html/classpython\\_1\\_1assemblies\\_1\\_1gravity\\_1\\_1AsphericalGravityActuatorAssembly\\_1\\_1AsphericalGravityActuatorAssembly.html](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/DshellCommon/html/classpython_1_1assemblies_1_1gravity_1_1AsphericalGravityActuatorAssembly_1_1AsphericalGravityActuatorAssembly.html))

2.6.2.4.3. CmFrameStateSensorAssembly

2.6.2.4.3.1. Class Documentation

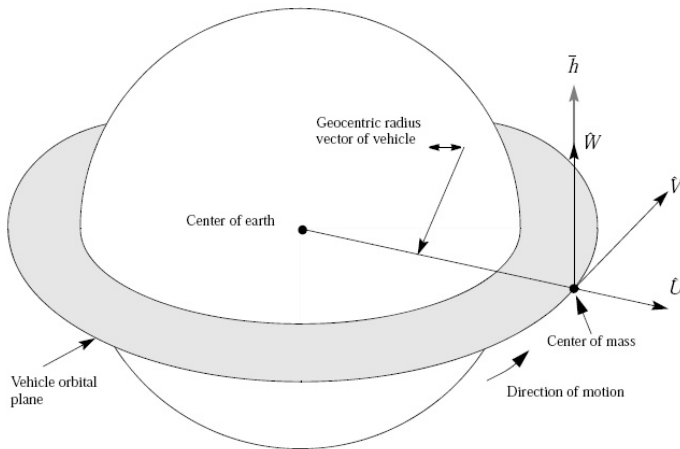
The CmFrameStateSensorAssembly is an assembly class for calculating the UVW or Local Vertical, Local Horizontal (LVLH) state of a vehicle. This assembly is responsible for creating and updating the UVW or LVLH frame for the input sensed body. The frame is dynamically moved to the CM of the component or composite vehicle bodies.

2.6.2.4.3.1.1. Visual Representation of LVLH Frame



- Name: Local-vertical local-horizontal (LVLH) coordinate system.
- Origin: Point of interest (e.g., vehicle center-of-mass).
- Orientation: The  $X_{LVLH} - Z_{LVLH}$  plane is the instantaneous orbit plane at epoch.  
 The  $Z_{LVLH}$  axis lies along the geocentric radius vector to the point of interest and is positive radially inward.  
 The  $Y_{LVLH}$  axis lies along the instantaneous orbital angular momentum vector ( $\vec{h}$ ) at epoch and is positive in the direction opposite the angular momentum vector.  
 The  $X_{LVLH}$  axis completes a right-handed system.
- Characteristics: Quasi-inertial, right-handed, Cartesian system. This system is quasi-inertial in the sense that it is treated as an inertial coordinate system, but is redefined at each new epoch

#### 2.6.2.4.3.1.2. Visual Representation of UVW Frame



- Name: U, V, W (UVW) coordinate system.
- Origin: Point of interest (e.g., vehicle center-of-mass).
- Orientation: The  $U - V$  plane is the instantaneous orbit plane at epoch.  
 The  $U$  axis lies along the geocentric radius vector to the point of interest and is positive radially outward.  
 The  $W$  axis lies along the instantaneous orbital angular momentum vector ( $\vec{h}$ ) at epoch and is positive in the direction of the angular momentum vector.  
 The  $V$  axis completes a right-handed system.
- Characteristics: Quasi-inertial, right-handed, Cartesian system. This system is quasi-inertial in the sense that it is treated as an inertial coordinate system, but is redefined at each new epoch.

#### 2.6.2.4.3.2. CmFrameStateSensorAssembly Usage

##### 2.6.2.4.3.2.1. Introduction

The CmFrameStateSensorAssembly is an assembly class for calculating the UVW or Local Vertical, Local Horizontal (LVLH) state of a vehicle. This assembly is responsible for creating and updating the UVW or LVLH frame for the input sensed body. The frame is dynamically moved to the CM of the component or composite vehicle bodies.

## LIMITATIONS

Modification of assembly config inputs after assembly creation is not supported.

Modification of sensedBodyName in assembly context is not supported, however; targetName may be modified.

Creation of a UVW or LVLH state for a vehicle initialized exactly over a planetary pole may create an incorrect orientation of the UVW/LVLH frame due to the FrameHelper algorithm not being able to use the typical Z-axis (Up direction) for its frame computations.

## CONSTRUCTION OPTIONS

- **config** - CmFrameStateSensorAssembly has one required and two optional config inputs:

“◦ **'frameType'**

*(Required) The type of the frame that this model will create to measure the frame attitude state.*

◦ **'nodeName'**

*The name of the node that this assembly will create to measure the frame attitude state. Defaults to the frameType parameter if not provided.*

◦ **'velocityReference'**

*Specifies whether to use inertial or relative velocity in creating the CM based frame. Defaults to INERTIAL if not provided.*

- **context** - CmFrameStateSensorAssembly requires two context inputs below:

“◦ **'targetName'**

*The name of the central body the attitude state is associated with.*

◦ **'sensedBodyName'**

*The vehicle body this frame is measuring attitude state.*

- **signalTies** - Optional.

- **params** - CmFrameStateSensorAssembly has two optional parameters:

“◦ **'frame'**

*This must be an instance of an CmFrameStateParam parameter class object. This information is used to initialize the center of mass type, euler angle sequence and euler angle module for output purposes. Applies param class defaults if not provided.*

◦ **'sensorLocation'**

This should point to a NodeParam parameter class object. This information is used to initialize the location of the sensor to the location of the given node. Note this param is only useful when using a FIXED node, as it will be over-ridden when using a COMPONENT or COMPOSITE center of mass type.

“

### 2.6.2.4.3.2.2. Signal Information

| Signal Name | Type          | Length | Quantity                | Units | Description  |
|-------------|---------------|--------|-------------------------|-------|--|
| angles quat | double double | 3 4    | Unspecified Unspecified |       | Euler angles extracted using desired sequence. Body orientation relative to input reference frame. |

### 2.6.2.4.3.2.3. Param Class Information

#### 2.6.2.4.3.2.3.1. NodeParam

“ *The **NodeParam** class provides all parameters required to specify a single Darts node.*

#### Required Parameters

| Name       | Type    | Length | Quantity | Units | Description   |
|------------|---------|--------|----------|-------|---|
| bodyToNode | vector3 | 1      | Length   | m     | The 3-vector to the node frame from the body frame. |

#### Optional Parameters

| Name           | Type   | Length | Quantity   | Units | Description                                  |
|----------------|--------|--------|------------|-------|--|
| bodyToNodeQuat | double | 4      | Quaternion |       | attitude of node with respect to body frame. |

### 2.6.2.4.3.2.3.2. CmFrameStateParam

The attributes for CM frame state.

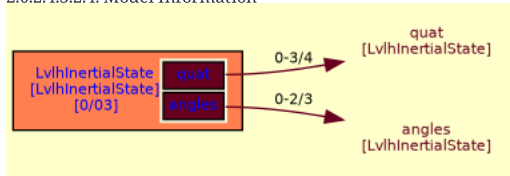
Required Parameters

None

Optional Parameters

| Name  | Type                 | Length         | Quantity    | Units | Description  |
|---|----------------------|----------------|-------------|-------|--|
| <i>angleModulo</i><br><i>centerOfMassType</i><br><i>eulerSequence</i> | string string string | None None None | N/A N/A N/A |       | Specify angles to be $-\pi \rightarrow \pi$ or $0 \rightarrow 2\pi$ (use string names 'MINUS_PI_TO_PI' or ZERO_TO_TWO_PI, respectively) Specify either 'FIXED', 'COMPONENT' or 'COMPOSITE' for CM location The euler sequence to use in reporting angles relative to the reference frame |

#### 2.6.2.4.3.2.4. Model Information



#### 2.6.2.4.3.2.4.1. CmFrameStateSensor

Maintains the CM based frame origin/attitude and outputs the orientation of a vehicle relative to the frame.

- *Keywords:* "Position" "Attitude" "Idealized"
- *Class:* sensor

Author: Scott Nemeth This sensor model uses node and frame objects to calculate the orientation of a body relative to the desired frame.

FlowIns

None

Parameters

| Name  | Type   | Length   | Quantity                                    | Units   | Value   | Description  |
|---|--|--|---|---|---|--|
| <i>CM_FRAME_UUID</i><br><i>PCI_FRAME_UUID</i><br><i>PCR_FRAME_UUID</i><br><i>angleModulo</i><br><i>centerOfMassType</i><br><i>eulerSequence</i><br><i>frameType</i><br><i>velocityReference</i> | int int int enum<br>enum<br>enum<br>enum<br>enum<br>enum | 1 1 1 1<br>1<br>1<br>1<br>1  | N/A N/A N/A N/A<br>N/A<br>N/A<br>N/A<br>N/A |   | 92 16 33<br>ZERO_TO_TWO_PI<br>COMPOSITE<br>YZX<br>LVLH<br>INERTIAL                        | <b>Not for user input</b> - UUID for the Center of Mass based frame. <b>Not for user input</b> - UUID for the planet centered inertial frame. <b>Not for user input</b> - UUID for the planet centered relative frame. |
| Specify angles to be $-\pi \rightarrow \pi$ or $0 \rightarrow 2\pi$ (use string names 'MINUS_PI_TO_PI' or ZERO_TO_TWO_PI, respectively)   | Allowed keys: MINUS_PI_TO_PI, ZERO_TO_TWO_PI             | Specify either 'FIXED', 'COMPONENT' or 'COMPOSITE' for CM location | Allowed keys: COMPONENT, COMPOSITE, FIXED   | The euler sequence to use in reporting angles relative to the reference frame | Allowed keys: XYX, XYZ, XZX, XZY, YXY, YXZ, YZX, YZY, ZXY, ZXZ, ZYX, ZYZ, <u>NOT_USED</u> | The type of the frame that this model will create to measure the frame attitude state.   |

Continuous States

None

Scratch

None

States

None

FlowOuts

| Name | Type | Length | Quantity | Units | Value | Description |
|------|------|--------|----------|-------|-------|-------------|
|------|------|--------|----------|-------|-------|-------------|

| Name          | Type   | Length | Quantity | Units | Value  | Description   |
|---------------|--------|--------|----------|-------|--|---|
| <i>angles</i> | double | 3      | Angle    | rad   | [-3.4028234663852886e+38, -3.4028234663852886e+38, -3.4028234663852886e+38]<br>Euler angles extracted using desired sequence.<br><i>quat</i> double 4 Quaternion<br>[-3.4028234663852886e+38, -3.4028234663852886e+38, -3.4028234663852886e+38, -3.4028234663852886e+38] | Body orientation relative to input reference frame. |

#### 2.6.2.4.3.3. Related Regression Tests

##### Unit test for the LVLH and UVW state assembly

► [Click to see the DshellCommon/test/test/test\\_Ndarts/test\\_CmFrameStateSensor/script.py script](#)

#### 2.6.2.4.3.4. Class API Documentation



##### Note

For Doxygen documentation, please see: [Center of Mass Based Frame State Sensor Assembly<CmFrameStateSensorAssembly::CmFrameStateSensorAssembly>](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/DshellCommon/html/classpython_1_1assemblies_1_1CmFrameStateSensorAssembly_1_1CmFrameStateSensorAssembly.html)  
([https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/DshellCommon/html/classpython\\_1\\_1assemblies\\_1\\_1CmFrameStateSensorAssembly\\_1\\_1CmFrameStateSensorAssembly.html](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/DshellCommon/html/classpython_1_1assemblies_1_1CmFrameStateSensorAssembly_1_1CmFrameStateSensorAssembly.html))

For C++ API documentation, please see: [Center of Mass Based Frame State Sensor Model<CmFrameStateSensor::CmFrameStateSensor>](#)



**TBD:** Fix Doxygen link for CmFrameStateSensor model

#### 2.6.2.4.4. ExternalDisturbanceActuatorAssembly

##### 2.6.2.4.4.1. Class Documentation

###### 2.6.2.4.4.1.1. Introduction

The ExternalDisturbanceActuatorAssembly purpose is to apply a force / torque on a body in the specified INERTIAL or BODY reference frames. The assembly utilizes the single C++ model ExternalDisturbance.

###### 2.6.2.4.4.1.2. Construction Options

- config
  - 'node' - Required
 

The name of the node that the force/torque is to be applied to. If no node is given the node takes the name of the assembly.
- context
  - 'body' - Required
 

The name of the body that the force/torque is to be applied to.
- signalTies - None
- param
  - 'Node' - Required
 

This is a NodeParam object.
  - 'Frame' - Optional
 

The Frame setting is a string with the following two values:

    - 'BODY' - apply the force / torque to the body frame. This is the default value.
    - 'INERTIAL' - apply the force / torque to the inertial frame.

###### 2.6.2.4.4.1.3. Example Construction Syntax

```

config = {
  'SC1' : {
    'class' : 'VehicleAssembly',
    ...
    'assemblies' : {
      ...
      'force' : {
        'class' : 'ExternalDisturbanceActuatorAssembly',
        'node' : 'VentForce',
        'context' : {'body': 'CapsuleBase'},
        'params' : {'Node': bodies.Library['SC1']['Nodes']['CapsuleBase']['Force'],
                   'Frame': FrameParam(params={'name': 'BODY'}, source=None)}
      }
    },
    ...
  }
}

```

This configuration block would normally appear in the 'assemblies' section for a VehicleAssembly.

##### 2.6.2.4.4.1.4. Signals

###### FlowIns

Two flowin signals exist to command the desired applied force and torque.

- **CmdForce** - Commanded force vector applied to the node
- **CmdTorque** - Commanded torque vector applied to the node

**FlowOuts** - None

#### 2.6.2.4.4.2. Usage Scenario

This is a simple usage of the External Disturance Actuator Assembly. Start with the standard import and import the Assembly. Create the Sim with SimulationExecutive.



```

"""
Test the VehicleAssembly class with two instances of ExternalDisturbanceActuatorAssembly
=====
(Adapted from test/test_Ndarts/test_ExtDistActuatorAssembly/script.py)

>>> from DshellCommon.utils.sphinxutils import generateRST
>>> import sys

>>> from pprint import pformat
>>> from math import fabs

>>> from Math.SOA_Py import SOAVector3

Create the simulation
-----

>>> from DshellCommon.SimulationExecutiveNdarts import SimulationExecutiveNdarts

>>> sim = SimulationExecutiveNdarts(banner=False)

Load the parameter libraries

>>> sys.path.append('./library')

>>> import targets
>>> import bodies
>>> import states

>>> from DshellCommon.params.DisturbanceModeParam import DisturbanceModeParam

Define the simulation configuration
-----

>>> from DshellCommon.assemblies import ExternalDisturbanceActuatorAssembly
>>> from DshellCommon.assemblies import TargetAssembly
>>> from DshellCommon.assemblies import VehicleAssembly

>>> config = {
...     'Mars': {
...         'class': 'TargetAssembly',
...         'params': { 'Target': targets.Library['Mars'],
...                     'Bodies': bodies.Library['Target']['Bodies'],
...                     }
...     },
...     'SC1': {
...         'class': 'VehicleAssembly',
...         'basename': 'CapsuleBase',
...         'bodygraph': {
...             'verts': ['CapsuleBase0', 'CapsuleBase', 'FuelTank'],
...             'edges': {'CapsuleBase': 'CapsuleBase0',
...                       'FuelTank': 'CapsuleBase'},
...         },
...         'params': {
...             'Bodies': bodies.Library['SC1']['Bodies'],
...             'Position': states.Library['MSL 05-19']['Position'],
...             'Attitude': states.Library['MSL 05-19']['Attitude'],
...         },
...         'assemblies': {
...             'vforce': {
...                 'class': 'ExternalDisturbanceActuatorAssembly',
...                 'node': 'VentForce2',
...                 'context': {'body': 'CapsuleBase'},
...                 'params': {'Node': bodies.Library['SC1']['Nodes']['CapsuleBase']['Force']},
...             },
...             'dforce': {
...                 'class': 'ExternalDisturbanceActuatorAssembly',
...                 'node': 'DragForce',
...                 'context': {'body': 'CapsuleBase'},
...                 'params': {'Node': bodies.Library['SC1']['Nodes']['CapsuleBase']['Force'],
...                            'DisturbanceMode': DisturbanceModeParam(mode='INERTIAL', source=None)},
...             },
...         },
...     },
... }

Create all the assemblies
-----

>>> scs = sim.createAssemblies(config)

>>> sim.bindState()

>>> sim.resetState(0.0)

>>> dforce = sim.assembly('dforce', 0, True)

```

### 2.6.2.4.4.3. Usage

#### 2.6.2.4.4.3.1. Introduction

A force/torque actuator assembly class.

#### 2.6.2.4.4.3.2. Signal Information

| Signal Name | Type   | Length | Quantity | Units | Description  |
|-------------|--------|--------|----------|-------|--|
| CmdForce    | double | 3      | Force    | N     | Force input for dforce generalized force actuator  |
| CmdTorque   | double | 3      | Torque   | N*m   | Torque input for dforce generalized force actuator |

#### 2.6.2.4.4.3.3. Param Class Information

##### 2.6.2.4.4.3.3.1. NodeParam

“ The **NodeParam** class provides all parameters required to specify a single Darts node.

#### Required Parameters

| Name       | Type    | Length | Quantity | Units | Description   |
|------------|---------|--------|----------|-------|---|
| bodyToNode | vector3 | 1      | Length   | m     | The 3-vector to the node frame from the body frame. |

#### Optional Parameters

| Name           | Type   | Length | Quantity   | Units | Description                                  |
|----------------|--------|--------|------------|-------|--|
| bodyToNodeQuat | double | 4      | Quaternion |       | attitude of node with respect to body frame. |

#### 2.6.2.4.4.3.3.2. FrameParam

“ Frame parameters.

#### Required Parameters

| Name | Type   | Length | Quantity | Units | Description    |
|------|--------|--------|----------|-------|----------------|
| name | string | 1      | N/A      |       | The frame name |

#### Optional Parameters

None

#### 2.6.2.4.4.3.3.3. DisturbanceModeParam

“ The **DisturbanceModeParam** class provides a parameter for ExternalDisturbanceForce models.

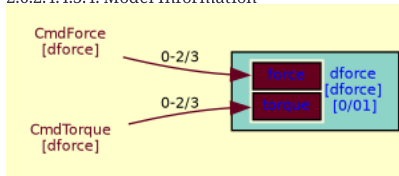
#### Required Parameters

None

#### Optional Parameters

| Name | Type   | Length | Quantity | Units | Description   |
|------|--------|--------|----------|-------|---|
| mode | string | None   | N/A      |       | Whether external forces are applied in body or inertial coordinates |

#### 2.6.2.4.4.3.4. Model Information



#### 2.6.2.4.4.3.4.1. ExternalDisturbance

Applies forces and torques to a node

- **Keywords:** "Disturbance model"
- **Class:** actuator

Author: David Henriquez / Garrett Sohl (modified to use flows) The ExternalDisturbance actuator model applies forces and torques to the actuator node on which the model is attached. The torque and force flowins are used to specify the applied force vector and the applied torque vector.

#### FlowIns

| Name   | Type   | Length | Quantity | Units | Value           | Description |
|--------|--------|--------|----------|-------|-----------------|-------------|
| force  | double | 3      | Force    | N     | [0.0, 0.0, 0.0] | force       |
| torque | double | 3      | Torque   | N*m   | [0.0, 0.0, 0.0] | torque      |

#### Parameters

| Name | Type | Length | Quantity | Units | Value | Description |
|------|------|--------|----------|-------|-------|-------------|
|------|------|--------|----------|-------|-------|-------------|

| Name                    | Type | Length | Quantity | Units | Value    | Description  |
|-------------------------|------|--------|----------|-------|----------|--|
| <i>disturbanceFrame</i> | enum | 1      | N/A      |       | INERTIAL | Whether external forces are applied in body or inertial coordinates<br>Allowed keys:<br>BODY, INERTIAL |

#### Continuous States

None

#### Scratch

None

#### States

None

#### FlowOuts

None

#### 2.6.2.4.4.3.5. Related Regression Tests

- Vehicle with external disturbance forces Create a free-flying vehicle without gravity and apply a force and torque to test ExternalDisturbanceActuatorAssembly.

#### 2.6.2.4.4.3.6. ExternalDisturbanceActuatorAssembly Class API Documentation



#### Note

For Doxygen documentation for the underlying model, please see: [ExternalDisturbance<Dshell::ExternalDisturbance>](#)

The ExternalDisturbanceActuatorAssembly is specific to the ExternalDisturbance model which applies forces and torques to the actuator node on which the model is attached in the specified Frame. The Frame can be BODY or INERTIAL with BODY as the default if no frame is specified.

#### 2.6.2.4.5. ExternalDisturbanceMotorAssembly

##### 2.6.2.4.5.1. Class Documentation

##### 2.6.2.4.5.1.1. Introduction

The ExternalDisturbanceMotorAssembly purpose allows the user to specify the appropriate C++ model available to apply a force / acceleration to a specific joint type. Currently, only a pin joint force is available.

##### 2.6.2.4.5.1.2. Construction Options

- config
  - 'model' - Required The name of the C++ model to apply the force/acceleration to the appropriate joint. Currently, only the following pin joint force can be specified.
    - 'GeneralForcePin'
- context
  - 'body' - Required::
    - The name of the body that the force/acceleration is to be applied to.
- signalTies - None
- param None

##### 2.6.2.4.5.1.3. Example Construction Syntax

```

config = {
  'SC1' : {
    'class' : 'VehicleAssembly',
    ...
    'assemblies' : {
      ...
      'pinforce' : {
        'class' : 'ExternalDisturbanceMotorAssembly',
        'model' : 'GeneralForcePin',
        'context' : {'body' : 'PinBody'},
      }
    },
    ...
  }
}

```

This configuration block would normally appear in the 'assemblies' section for a VehicleAssembly.

##### 2.6.2.4.5.1.4. Signals

#### FlowIns

One flowin signal exists to command the desired applied force and torque.

- Cmd** - Commanded force/acceleration applied to the joint

#### FlowOuts - None

##### 2.6.2.4.5.1.5. Related Regression Tests

- Vehicle with motor disturbance Create a free-flying vehicle without gravity and apply a force and torque to test ExternalDisturbanceMotorAssembly.

2.6.2.4.5.1.6. ExternalDisturbanceMotorAssembly Class API Documentation



**Note**

For Doxygen documentation for the underlying model, please see: [GeneralForcePin<Dshell>::GeneralForcePin](#)

The ExternalDisturbanceMotorAssembly allows for the 1-DOF pin force on PIN joint or a 2-DOF force on a UJOINT.



TBD: Add the Doxygen link above

2.6.2.4.6. FuelManifoldAssembly

2.6.2.4.6.1. Introduction

A fuel manifold model.

2.6.2.4.6.2. Signal Information

| Signal Name             | Length | Type   | Description                                     |
|-------------------------|--------|--------|---|
| FuelUsedByManifold      | 2      | double | Fuel line draws from tanks by manifold manifold |
| FuelRemainingInManifold | 1      | double | Fuel available from manifold manifold           |

2.6.2.4.6.3. Param Class Information

2.6.2.4.6.4. Model Information

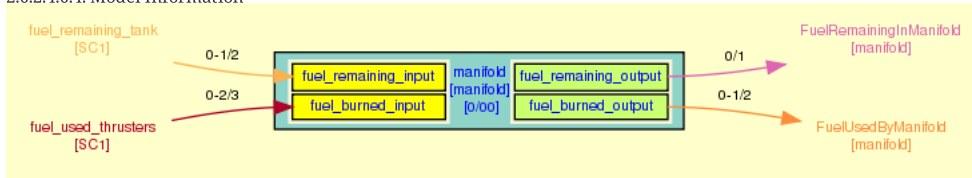


Figure 31. Model Data Flow

2.6.2.4.6.5. FuelManifold

Models a simple fuel manifold.

- **Keywords:**
- **Class:** actuator

The FuelManifold model accepts two inputs which denotes the amount of fuel burned/drawn from a set of sinks and the amount of fuel available from a set of sources. The inputs are variable sized so that several thrusters/sinks and several tanks can share a single FuelManifold model. It outputs two array of doubles: fuel\_burned\_output, the sum of the input fuel draws divided by the size of flowOuts0→fuel\_burned, and fuel\_remaining\_output, the sum of input fuel availability replicated on all slices.

Note: when the fuel\_remaining is zero on any line the draw rates for the other lines will be adjusted by their ratios to give a total flow rate of 1.0.

2.6.2.4.6.5.1. FlowIns

| Name                 | Units | Length | Description                                    |
|----------------------|-------|--------|--|
| fuel_burned_input    |       | 3      | Total amount of fuel burned from each input    |
| fuel_remaining_input |       | 2      | Total amount of remaining fuel from each input |

2.6.2.4.6.5.2. Parameters

| Name              | Units | Length | Description  |
|-------------------|-------|--------|--|
| fuel_draw_weights |       | 2      | Fuel draw weighting for the fuel_burned_output lines |

2.6.2.4.6.5.3. Continuous States

None

2.6.2.4.6.5.4. Scratch

None

2.6.2.4.6.5.5. States

| Name                      | Units | Length | Description   |
|---------------------------|-------|--------|---|
| current_fuel_draw_weights |       | 2      | The current fuel draw weighting for the fuel_burned_output lines            |
| fuel_burned_output        |       | 1      | The total amount of fuel burned for each tank as far as this manifold knows |
| num_burned_inputs         |       | 1      | size of flowIns0→fuel_burned_input  |
| num_burned_outputs        |       | 1      | size of flowOuts0→fuel_burned_output  |

| Name                        | Units | Length | Description          |
|-----------------------------|-------|--------|----------------------|
| <i>total_fuel_burned</i>    |       | 1      | total fuel burned    |
| <i>total_fuel_remaining</i> |       | 1      | total remaining fuel |

#### 2.6.2.4.6.5.6. FlowOuts

| Name                         | Units | Length | Description  |
|------------------------------|-------|--------|--|
| <i>fuel_burned_output</i>    |       | 2      | Sum of thruster fuel burned divided by num_outputs multiplied by fuel_draw_weighting |
| <i>fuel_remaining_output</i> |       | 1      | Sum of fuel remaining in tanks connected to manifold                                 |

#### 2.6.2.4.6.6. Related Regression Tests

##### Test the FuelManifoldAssembly with two tanks and four thrusters

► [Click to see the DshellCommon/test/test\\_Ndarts/test\\_FuelManifoldAssembly/script.py script](#)

#### 2.6.2.4.6.7. FuelManifoldAssembly Class API Documentation



##### Note

For Doxygen documentation for the underlying model, please see: [FuelManifold](#)

#### Fuel Manifold Assembly class for DshellCommon Simulation models



**TBD:** Add Doxygen link for FuelManifold model above

#### 2.6.2.4.7. GeneralGravityActuatorAssembly

##### 2.6.2.4.7.1. Class Documentation

###### 2.6.2.4.7.1.1. Introduction

The GeneralGravityActuatorAssembly ?

###### 2.6.2.4.7.1.2. Construction Options

- config - GeneralGravityActuatorAssembly has one optional config setting. Inherited from its base class, GravityActuatorAssembly:

- **'node'**

The name of the node that the gravity is to be applied to. Defaults to the name of the assembly if not provided.

- context

GeneralGravityActuatorAssembly has one required config setting. Inherited from its base class, GravityActuatorAssembly:

- **'body'**

The name of the body that the gravity is to be applied to.

- signalTies - None

- param

GeneralGravityActuatorAssembly has one optional parameter that it inherits from its base class, GravityActuatorAssembly:

- **'Node'**

[Optional] This should point to a `NodeParam` parameter class object. This is usually stored in a file in the library of your run script such as: `./library/bodies.py`. This information is used to initialize the relative position of the node with respect to the body reference coordinates.

###### 2.6.2.4.7.1.3. Example Construction Syntax

```

config = {
  'SC1' : {
    'class' : 'VehicleAssembly',
    ...
    'assemblies' : {
      'grav' : {
        'class' : 'GeneralGravityActuatorAssembly',
        'context' : { 'body' : 'CapsuleBase0' }
      }
    }
  },
  ...
}

```

This configuration block would normally appear in the 'assemblies' section for a VehicleAssembly.

###### 2.6.2.4.7.1.4. Signals

##### FlowIns

The GeneralGravityActuatorAssembly has two flowin signals for user to command the desired linear and angular gravitational acceleration. These are inherited from the GravityActuatorAssembly base class.

- **LinAccelIn** - 3-vector of desired linear gravitational acceleration

- **AngAccelIn** - 3-vector of desired rotational gravitational acceleration

## FlowOuts

Inherited from base class GravityActuatorAssembly.

- **LinAccel** - 3-vector of linear gravitational acceleration
- **AngAccel** - 3-vector of rotational gravitational acceleration

### 2.6.2.4.7.1.5. Related Regression Tests

- Vehicle with target body and gravity Create a vehicle with 3 bodies and a planetary target body. Set up simple gravity.

### 2.6.2.4.7.2. GeneralGravityActuatorAssembly Class API Documentation



#### Note

For Doxygen documentation, please see: `GeneralGravityActuatorAssembly<GeneralGravityActuatorAssembly: :GeneralGravityActuatorAssembly>`

GeneralGravity Force Actuator Assembly class for EDL simulation models

This gravity assembly sets up the GeneralGravity model which takes constant linear and angular acceleration terms specified by signals by the user (or other source) and applies the gravity to the node to which the gravity model is attached. It also includes flowouts of the linear and angular accelerations used.

### 2.6.2.4.8. GravityActuatorAssembly

#### 2.6.2.4.8.1. Introduction

The GravityActuatorAssembly is the base assembly for all gravitational assemblies.

#### 2.6.2.4.8.2. Construction Options

- config - GravityActuatorAssembly has one optional config setting:

- **'node'**

The name of the node that the gravity is to be applied to. Defaults to the name of the assembly if not provided.

- context - GravityActuatorAssembly has one required config setting:

- **'body'**

The name of the body that the gravity is to be applied to.

- signalTies - None

- param - GravityActuatorAssembly has one optional parameter:

- **'Node'**

This should point to a `NodeParam` parameter class object. This is usually stored in a file in the library of your run script such as: `./library/bodies.py`. This information is used to initialize the relative position of the node with respect to the body reference coordinates.

#### 2.6.2.4.8.3. Signals

##### 2.6.2.4.8.3.1. FlowIns

None.

##### 2.6.2.4.8.3.2. FlowOuts

The GravityActuatorAssembly has two flowout signals for user inspection of the applied gravity force and torque:

- **LinAccel** - 3-vector of linear gravitational acceleration
- **AngAccel** - 3-vector of rotational gravitational acceleration

#### 2.6.2.4.8.4. Popular GravityActuatorAssembly Functions

Some of the most common functions for the GravityActuatorAssembly are:

#### Functions related to activating the gravity model:

| Function  |
|---|
| <a href="https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/DshellCommon/html/classpython_1_1assemblies_1_1gravity_1_1GravityActuatorAssembly_1_1GravityActuatorAssembly.html#ad6c7f5a8b32bda9b755dbaf">gravity.activate()</a><br>(https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/DshellCommon/html/classpython_1_1assemblies_1_1gravity_1_1GravityActuatorAssembly_1_1GravityActuatorAssembly.html#ad6c7f5a8b32bda9b755dbaf)       |
| <a href="https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/DshellCommon/html/classpython_1_1assemblies_1_1gravity_1_1GravityActuatorAssembly_1_1GravityActuatorAssembly.html#acd543b333f8b07246af2c60b">gravity.deactivate()</a><br>(https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/DshellCommon/html/classpython_1_1assemblies_1_1gravity_1_1GravityActuatorAssembly_1_1GravityActuatorAssembly.html#acd543b333f8b07246af2c60b)   |
| <a href="https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/DshellCommon/html/classpython_1_1assemblies_1_1gravity_1_1GravityActuatorAssembly_1_1GravityActuatorAssembly.html#a78a4ad0356b090106b42df1e">gravity.gravityModel()</a><br>(https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/DshellCommon/html/classpython_1_1assemblies_1_1gravity_1_1GravityActuatorAssembly_1_1GravityActuatorAssembly.html#a78a4ad0356b090106b42df1e) |

### 2.6.2.4.8.5. GravityActuatorAssembly Class API Documentation



**Note**

For Doxygen documentation, please see: [GravityActuatorAssembly<GravityActuatorAssembly::GravityActuatorAssembly>](#)

Gravity Force Actuator Assembly base class for DshellCommon simulation models

[2.6.2.4.9. LinkAssembly](#)

**2.6.2.4.9.1. Class Documentation**

A single link assembly class.

**2.6.2.4.9.2. Usage Scenario**

This is a simple usage of the Link Assembly. Start with the standard imports and import the Assembly. Create the Sim with SimulationExecutive.

```

"""
Test the LinkAssembly (with default params)
=====
(Adapted from test/test_Ndarts/test_LinkAssembly/script1.py)
>>> from DshellCommon.utils.sphinxutils import generateRST

Create the simulation (with no assemblies)
-----

>>> from DshellCommon.SimulationExecutiveNdarts import SimulationExecutiveNdarts

>>> sim = SimulationExecutiveNdarts(banner=False)

Load the parameter libraries

>>> sys.path.append('./library')

>>> import actuators
>>> import bodies

Define the simulation configuration
-----

>>> from DshellCommon.assemblies import LinkAssembly
>>> from DshellCommon.assemblies import VehicleAssembly

>>> config = {
...     'SC1_Alternate1' : {
...         'class' : 'VehicleAssembly',
...         'basename' : 'baseBody',
...         'params' : {
...             'Bodies' : bodies.Library['SC1_Alternate1']['Bodies']
...         },
...         'signals' : {
...             'Segment_set_input' : {
...                 'type' : 'double', 'length' : 1,
...                 'comment' : 'Commanded joint angle',
...             },
...             'Segment_actual_angle' : {
...                 'type' : 'double', 'length' : 1,
...                 'comment' : 'Actual joint angle',
...             },
...         },
...         'signal_ties' : {
...             'Segment_set_input' : [
...                 { 'child_signal_name' : 'Segment_set_input',
...                   'child_assembly_name' : 'Segment' },
...             ],
...             'Segment_actual_angle' : [
...                 { 'child_signal_name' : 'Segment_angle_truth',
...                   'child_assembly_name' : 'Segment' },
...             ],
...         },
...         'assemblies' : {
...             'Segment' : {
...                 'class' : 'LinkAssembly',
...                 'motorType' : 'PrescribedPin',
...                 'context' : { 'parentBody' : 'baseBody' },
...                 'params' : { 'Link' : actuators.Library['Link']['JointStop'],
...                               'Body' : bodies.Library['SC1_Alternate1']['Bodies'],
...                 }
...             }
...         },
...     },
... }

Create all the assemblies
-----

>>> scs = sim.createAssemblies(config)

>>> sim.bindState()

>>> sim.resetState(0.0)

>>> link = sim.assembly('Segment', 0, True)

```

### 2.6.2.4.9.3. Usage

#### 2.6.2.4.9.3.1. Introduction

A single link assembly class.

#### 2.6.2.4.9.3.2. Signal Information

There are no signals in this assembly.

#### 2.6.2.4.9.3.3. Param Class Information

##### 2.6.2.4.9.3.3.1. BodyParam

 The **BodyParam** class provides all parameters required to specify a single Darts body.



Required Parameters

| Name        | Type   | Length | Quantity | Units | Description   |
|-------------|--------|--------|----------|-------|---------------|
| <i>mass</i> | double | 1      | Mass     | kg    | The body mass |

Optional Parameters

| Name  | Type   | Length                              | Quantity  | Units | Description  |
|---|--|-------------------------------------|---|-------|--|
| <i>bodyToCM</i>   | vector3  | 1                                   | Length  | m     | The 3-vector to the body CM from the body frame.   |
| <i>bodyToJoint</i><br><i>bodyToJointQuat</i> <i>cmInertia</i><br><i>description</i> <i>gearRatio</i><br><i>geometry</i>   | vector3 double double<br>string double \{\}                      | 1 4 None 1 1 None                   | Length Quaternion<br>MomentsOfInertia N/A<br>Dimensionless N/A                        | m     | The 3-vector from the body frame to the body hinge The quaternion from the body frame to the pnode for the body hinge The 3x3 body inertia matrix about the body CM (do not specify both inertia and cmInertia) DEPRECATED: Body description. IGNORED. Gear ratio for PIN joints Geometry to be loaded into DScene.  |
| <i>inbToJoint</i> <i>inbToJointQuat</i><br><i>inertia</i> <i>jointAxes</i> <i>jointLimits</i><br><i>jointType</i><br><i>negativeIntegralSense</i><br><i>partGraphics</i> <i>prescribed</i><br><i>prescribedType</i> <i>subhinge</i> | vector3 double double<br>vector3 double string bool \{\}<br>bool | 1 4 None ** 1 1 None * None<br>None | Length Quaternion<br>MomentsOfInertia<br>Dimensionless N/A N/A N/A<br>N/A N/A N/A N/A | m     | The 3-vector from the inboard body frame to the body hinge The quaternion from the inboard body frame to the body hinge The 3x3 body inertia matrix about the body frame (do not specify both inertia and cmInertia) The hinge rotation/translation axis Limits for range of joint generalized coordinate [min, max] The hinge type (string). Supported: BALL, COMPOSITE-TRANSLATIONAL, FULL6DOF, FULL6DOF_INERTIAL, GIMBAL, LOCKED, PIN, PLANAR, SLIDER, TRANSLATIONAL, UJOINT, CUSTOM Body products of inertia values expressed in NEGATIVE inertia integral sense (defaults to True) DEPRECATED: Geometry to be loaded into DScene. Whether the joint is prescribed or free (scalar boolean or array of booleans for each subhinge) DEPRECATED: Type of prescribed joint. WARNING: Not permitted in Ndarts Subhinge specification for custom hinge. |

2.6.2.4.9.3.3.2. ControllerParam

“The attributes for the controller assembly

Required Parameters

| Name   | Type                           | Length  | Quantity        | Units | Description  |
|--|--------------------------------|---------|-----------------|-------|--|
| <i>kd</i> <i>ki</i> <i>kp</i> <i>maxOutput</i> | double double double<br>double | 1 1 1 1 | N/A N/A N/A N/A |       | Derivative gain Integral gain Proportional gain Maximum output |

Optional Parameters  
None

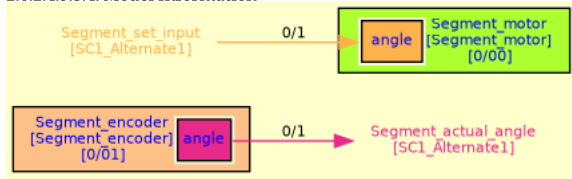
2.6.2.4.9.3.3.3. BaseMotorParam  
“The attributes for the link assembly.

Required Parameters  
None

Optional Parameters

| Name | Type   | Length | Quantity | Units | Description            |
|------|--------|--------|----------|-------|------------------------|
| Type | string | 1      | N/A      |       | String parameter field |

2.6.2.4.9.3.4. Model Information



2.6.2.4.9.3.4.1. PinEncoder  
Outputs the pin joint angular displacement

- *Keywords:* "Hinge!Pin"
- *Class:* encoder

Author: David Henriquez The PinEncoder encoder reports the angular displacement a pin joint hinge has undergone (i.e. angle). If PinEncoder is attached to a hinge with more than one degree of freedom, only the first degree of freedom will be reported; however, such a scenario may cause a system error (i.e. segmentation fault).

FlowIns  
None

Parameters  
None

Continuous States  
None

Scratch  
None

States

| Name  | Type   | Length | Quantity | Units | Value | Description |
|-------|--------|--------|----------|-------|-------|-------------|
| angle | double | 1      | Angle    | rad   | 0.0   |             |

FlowOuts

| Name  | Type   | Length | Quantity | Units | Value | Description       |
|-------|--------|--------|----------|-------|-------|-------------------|
| angle | double | 1      | Angle    | rad   | [0.0] | hinge angle value |

2.6.2.4.9.3.4.2. PrescribedPin  
Forces a hinge pin to follow prescribed angles (via flowins) for kinematics modes

- *Keywords:* "Hinge!Pin" "Hinge!Prescribed"
- *Class:* motor

If the prescribed angle exceeds specified min/max angles, they will be capped appropriately.

FlowIns

| Name  | Type   | Length | Quantity | Units | Value | Description                            |
|-------|--------|--------|----------|-------|-------|--|
| angle | double | 1      | Angle    | rad   | [0.0] | prescribed hinge angle value (radians) |

Parameters

| Name                           | Type       | Length | Quantity  | Units | Value  | Description   |
|--------------------------------|------------|--------|-----------|-------|--------|---|
| max_angle<br>max_angle_enabled | double int | 1 1    | Angle N/A | rad   | 10.0 1 | Joint angle limit maximum (radians, defaults to no limit) 1 if min angle limit is enabled, 0 if not |

| Name   | Type       | Length | Quantity  | Units | Value | Description   |
|--|------------|--------|-----------|-------|-------|---|
| <i>min_angle</i><br><i>min_angle_enabled</i> | double int | 1 1    | Angle N/A | rad   | 0.0 1 | Joint angle limit minimum (radians, defaults to no limit) 1 if min angle limit is enabled, 0 if not |

Continuous States  
None

Scratch

| Name                                       | Type    | Length | Quantity | Units | Value | Description   |
|--|---------|--------|----------|-------|-------|---|
| <i>at_max_angle</i><br><i>at_min_angle</i> | int int | 1 1    | N/A N/A  |       | 0 0   | 1 if angle is at maximum angle joint stop (0 otherwise) 1 if angle is at minimum angle joint stop (0 otherwise) |

States  
None

FlowOuts  
None

2.6.2.4.9.4. Related Regression Tests

**Test the LinkAssembly with PrescribedPin prescribed hinge motors**

▶ [Click to see the DshellCommon/test/test\\_Ndarts/test\\_LinkAssembly/script1.py script](#)

**Test the LinkAssembly with GeneralForcePin torque inputs**

▶ [Click to see the DshellCommon/test/test\\_Ndarts/test\\_LinkAssembly/script2.py script](#)

**Test the LinkAssembly (with spring damper motors)**

▶ [Click to see the DshellCommon/test/test\\_Ndarts/test\\_LinkAssembly/script3.py script](#)

2.6.2.4.9.5. Class API Documentation



*Note*

For Doxygen documentation, please see: {DshellCommon\_LinkAssembly\_class\_doxygen\_uri}[Link Assembly<assemblies::LinkAssembly::LinkAssembly>]

Linkbody assembly

2.6.2.4.10. NodePosVelAccelSensorAssembly

2.6.2.4.10.1. Introduction

A node truth sensor assembly

2.6.2.4.10.2. Signal Information

| Signal Name      | Length | Type   | Description  |
|------------------|--------|--------|--|
| <i>quat</i>      | 4      | double | Truth quaternion with respect to the inertial frame for node imu |
| <i>omega</i>     | 3      | double | gyro rates in inertial frame for node imu (rad/sec)              |
| <i>bodyOmega</i> | 3      | double | gyro rates in body frame for node imu (rad/sec)                  |
| <i>alpha</i>     | 3      | double | gyro angular accels in inertial frame for node imu (rad/sec^2)   |
| <i>pos</i>       | 3      | double | position relative to inertial frame for node imu (meters)        |
| <i>vel</i>       | 3      | double | velocity relative to inertial frame for node imu (m/sec)         |
| <i>bodyVel</i>   | 3      | double | velocity relative to body frame for node imu (m/sec)             |
| <i>accel</i>     | 3      | double | accelerations in inertial frame for node imu (m/sec^2)           |

| Signal Name            | Length | Type   | Description   |
|------------------------|--------|--------|---|
| <i>bodySensedAccel</i> | 3      | double | sensed accelerations in body frame for node imu (m/sec^2) |

### 2.6.2.4.10.3. Param Class Information

#### 2.6.2.4.10.3.1. NodeParam

Single body node parameters.

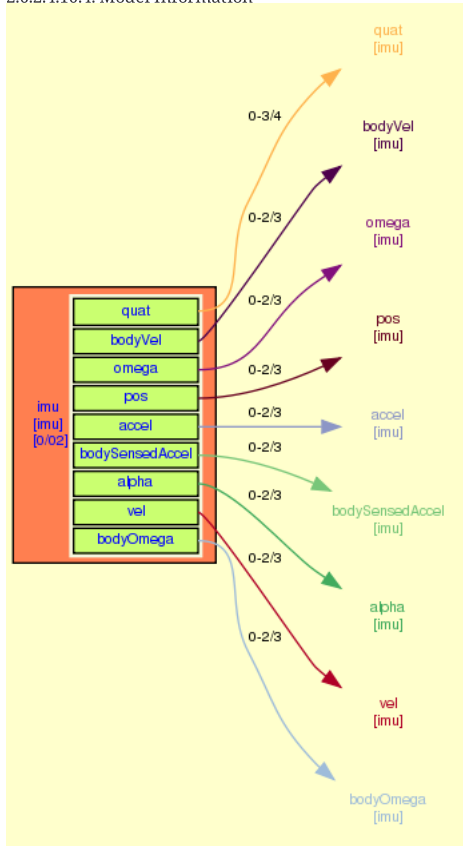
#### 2.6.2.4.10.3.2. Required Parameters

| Name              | Units | Description   |
|-------------------|-------|---|
| <i>bodyToNode</i> | m     | The 3-vector to the node frame from the body frame. |

#### 2.6.2.4.10.3.3. Optional Parameters

None

#### 2.6.2.4.10.4. Model Information



#### 2.6.2.4.10.5. NodePosVelAccelSensor

Model of a IMU truth sensor

- *Keywords*: "Inertial sensor!IMU" "Inertial sensor!Gyroscope" "Inertial sensor!Accelerometer"
- *Class*: sensor

Returns linear and angular rates and accelerations. Linear values are given in the inertial frame, while angular values are given in the local/node frame.

Partly based on RoverDynModels/RoverTruthIMU

#### 2.6.2.4.10.5.1. FlowIns

None

#### 2.6.2.4.10.5.2. Parameters

None

#### 2.6.2.4.10.5.3. Continuous States

None

#### 2.6.2.4.10.5.4. Scratch

| Name            | Units  | Length | Description   |
|-----------------|--------|--------|---|
| <i>accelMag</i> | m/s**2 | 1      | Linear acceleration magnitude (with respect to the inertial frame, m/sec^2) |

| Name                      | Units  | Length | Description  |
|---------------------------|--------|--------|--|
| <i>bodySensedAccelMag</i> | m/s**2 | 1      | Linear acceleration magnitude in body axis (with respect to the inertial frame, m/sec^2) |
| <i>bodyVelMag</i>         | m/s    | 1      | Linear inertial velocity magnitude (with respect to the body frame, m/sec)               |
| <i>nodeFrameAccel</i>     | m/s**2 | 6      | Total linear spatial acceleration written in node frame                                  |
| <i>posMag quat</i>        | m      | 1 4    | Position magnitude relative to inertial frame (m) Node attitude quaternion               |
| <i>velMag</i>             | m/s    | 1      | Linear inertial velocity magnitude (with respect to the inertial frame, m/sec)           |

2.6.2.4.10.5.5. States  
None

2.6.2.4.10.5.6. FlowOuts

| Name                   | Units    | Length | Description  |
|------------------------|----------|--------|--|
| <i>accel</i>           | m/s**2   | 3      | Linear accelerations (with respect to the inertial frame, m/sec^2)                               |
| <i>alpha</i>           | rad/s**2 | 3      | Angular accels (with respect to the inertial frame, rad/sec^2)                                   |
| <i>bodyOmega</i>       | rad/s    | 3      | Angular rates (with respect to the body frame, rad/sec)  |
| <i>bodySensedAccel</i> | m/s**2   | 3      | Linear accelerations in body axis (with respect to the inertial frame, m/sec^2)                  |
| <i>bodyVel</i>         | m/s      | 3      | Linear inertial velocity (with respect to the body frame, m/sec)                                 |
| <i>omega</i>           | rad/s    | 3      | Angular rates (with respect to the inertial frame, rad/sec)                                      |
| <i>pos quat</i>        | m        | 3 4    | Position relative to inertial frame (m) Attitude quaternion (with respect to the inertial frame) |
| <i>vel</i>             | m/s      | 3      | Linear inertial velocity (with respect to the inertial frame, m/sec)                             |

2.6.2.4.10.6. Related Regression Tests

**VehicleAssembly class with one vehicle and a truth sensor**

▶ [Click to see the DshellCommon/test/test\\_Ndarts/test\\_node\\_truth/script.py script](#)

2.6.2.4.10.7. NodePosVelAccelSensorAssembly Class API Documentation



*Note*

For Doxygen documentation for the underlying model, please see: `imu::NodePosVelAccelSensorAssembly`

NodePosVelAccelSensor Assembly

[2.6.2.4.11. SerialLinksAssembly](#)

2.6.2.4.11.1. Class Documentation  
A serial links assembly class.

2.6.2.4.11.2. Usage Scenario

This is a simple usage of the Serial Links Assembly. Start with the standard imports and import the Assembly. Create the Sim with SimulationExecutive.

▶ [Click to see the DshellCommon/usage/usage\\_SerialLinksAssembly.py script](#)

2.6.2.4.11.3. Usage

2.6.2.4.11.3.1. Introduction  
A serial links assembly class.

2.6.2.4.11.3.2. Signal Information

There are no signals in this assembly.

2.6.2.4.11.3.3. Param Class Information

2.6.2.4.11.3.3.1. ControllerParam

The attributes for the controller assembly

Required Parameters

| Name                      | Type                        | Length  | Quantity        | Units | Description  |
|---------------------------|-----------------------------|---------|-----------------|-------|--|
| <i>kd ki kp maxOutput</i> | double double double double | 1 1 1 1 | N/A N/A N/A N/A |       | Derivative gain Integral gain Proportional gain Maximum output |

Optional Parameters  
None

#### 2.6.2.4.11.3.3.2. BaseMotorParam

“The attributes for the link assembly.

Required Parameters  
None

Optional Parameters

| Name        | Type   | Length | Quantity | Units | Description            |
|-------------|--------|--------|----------|-------|------------------------|
| <i>Type</i> | string | 1      | N/A      |       | String parameter field |

#### 2.6.2.4.11.3.3.3. BodyParam

“The **BodyParam** class provides all parameters required to specify a single Darts body.

Required Parameters

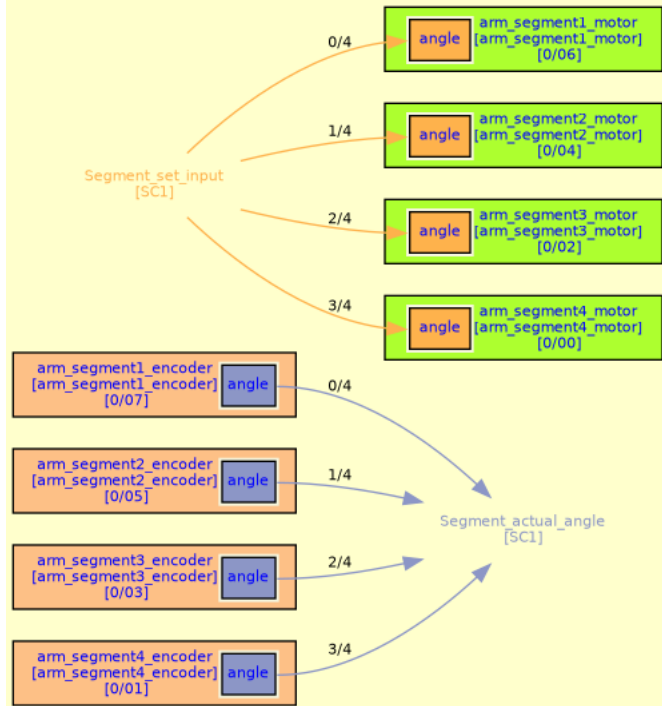
| Name        | Type   | Length | Quantity | Units | Description   |
|-------------|--------|--------|----------|-------|---------------|
| <i>mass</i> | double | 1      | Mass     | kg    | The body mass |

Optional Parameters

| Name  | Type                                       | Length            | Quantity   | Units | Description   |
|---|--|-------------------|--|-------|---|
| <i>bodyToCM</i>   | vector3                                    | 1                 | Length   | m     | The 3-vector to the body CM from the body frame.  |
| <i>bodyToJoint</i><br><i>bodyToJointQuat</i> <i>cmInertia</i><br><i>description</i> <i>gearRatio</i><br><i>geometry</i> | vector3 double double<br>string double \{} | 1 4 None 1 1 None | Length Quaternion<br>MomentsOfInertia N/A<br>Dimensionless N/A | m     | The 3-vector from the body frame to the body hinge The quaternion from the body frame to the pnode for the body hinge The 3x3 body inertia matrix about the body CM (do not specify both inertia and cmInertia) DEPRECATED: Body description. IGNORED. Gear ratio for PIN joints Geometry to be loaded into DScene. |

| Name  | Type  | Length                              | Quantity  | Units | Description  |
|---|---|-------------------------------------|---|-------|--|
| <i>inbToJoint</i> <i>inbToJointQuat</i><br><i>inertia</i> <i>jointAxes</i> <i>jointLimits</i><br><i>jointType</i><br><i>negativeIntegralSense</i><br><i>partGraphics</i> <i>prescribed</i><br><i>prescribedType</i> <i>subhinge</i> | vector3 double double<br>vector3 double string bool \           {} bool | 1 4 None ** 1 1 None * None<br>None | Length Quaternion<br>MomentsOfInertia<br>Dimensionless N/A N/A N/A<br>N/A N/A N/A N/A | m     | The 3-vector from the inboard body frame to the body hinge The quaternion from the inboard body frame to the body hinge The 3x3 body inertia matrix about the body frame (do not specify both inertia and cmInertia) The hinge rotation/translation axis Limits for range of joint generalized coordinate [min, max] The hinge type (string). Supported: BALL, COMPOSITE-TRANSLATIONAL, FULL6DOF, FULL6DOF_INERTIAL, GIMBAL, LOCKED, PIN, PLANAR, SLIDER, TRANSLATIONAL, UJOINT, CUSTOM Body products of inertia values expressed in NEGATIVE inertia integral sense (defaults to True) DEPRECATED: Geometry to be loaded into DScene. Whether the joint is prescribed or free (scalar boolean or array of booleans for each subhinge) DEPRECATED: Type of prescribed joint. WARNING: Not permitted in Ndarts Subhinge specification for custom hinge. |

2.6.2.4.11.3.4. Model Information



2.6.2.4.11.3.4.1. PinEncoder

Outputs the pin joint angular displacement

- Keywords: "Hinge!Pin"

- *Class:* encoder

Author: David Henriquez The PinEncoder encoder reports the angular displacement a pin joint hinge has undergone (i.e. angle). If PinEncoder is attached to a hinge with more than one degree of freedom, only the first degree of freedom will be reported; however, such a scenario may cause a system error (i.e. segmentation fault).

FlowIns  
None

Parameters  
None

Continuous States  
None

Scratch  
None

States

| Name         | Type   | Length | Quantity | Units | Value | Description |
|--------------|--------|--------|----------|-------|-------|-------------|
| <i>angle</i> | double | 1      | Angle    | rad   | 0.0   |             |

FlowOuts

| Name         | Type   | Length | Quantity | Units | Value | Description       |
|--------------|--------|--------|----------|-------|-------|-------------------|
| <i>angle</i> | double | 1      | Angle    | rad   | [0.0] | hinge angle value |

#### 2.6.2.4.11.3.4.2. PrescribedPin

Forces a hinge pin to follow prescribed angles (via flowins) for kinematics modes

- *Keywords:* "Hinge!Pin" "Hinge!Prescribed"
- *Class:* motor

If the prescribed angle exceeds specified min/max angles, they will be capped appropriately.

FlowIns

| Name         | Type   | Length | Quantity | Units | Value | Description                            |
|--------------|--------|--------|----------|-------|-------|--|
| <i>angle</i> | double | 1      | Angle    | rad   | [0.0] | prescribed hinge angle value (radians) |

Parameters

| Name   | Type       | Length | Quantity  | Units | Value  | Description   |
|--|------------|--------|-----------|-------|--------|---|
| <i>max_angle</i><br><i>max_angle_enabled</i> | double int | 1 1    | Angle N/A | rad   | 2.0 1  | Joint angle limit maximum (radians, defaults to no limit) 1 if min angle limit is enabled, 0 if not |
| <i>min_angle</i><br><i>min_angle_enabled</i> | double int | 1 1    | Angle N/A | rad   | -0.5 1 | Joint angle limit minimum (radians, defaults to no limit) 1 if min angle limit is enabled, 0 if not |

Continuous States  
None

Scratch

| Name                                       | Type    | Length | Quantity | Units | Value | Description   |
|--|---------|--------|----------|-------|-------|---|
| <i>at_max_angle</i><br><i>at_min_angle</i> | int int | 1 1    | N/A N/A  |       | 0 0   | 1 if angle is at maximum angle joint stop (0 otherwise) 1 if angle is at minimum angle joint stop (0 otherwise) |

States  
None

FlowOuts  
None

#### 2.6.2.4.11.3.4.3. PinEncoder

Outputs the pin joint angular displacement



- *Keywords:* "Hinge!Pin"
- *Class:* encoder

Author: David Henriquez The PinEncoder encoder reports the angular displacement a pin joint hinge has undergone (i.e. angle). If PinEncoder is attached to a hinge with more than one degree of freedom, only the first degree of freedom will be reported; however, such a scenario may cause a system error (i.e. segmentation fault).

FlowIns  
None

Parameters  
None

Continuous States  
None

Scratch  
None

States

| Name         | Type   | Length | Quantity | Units | Value | Description |
|--------------|--------|--------|----------|-------|-------|-------------|
| <i>angle</i> | double | 1      | Angle    | rad   | 0.0   |             |

FlowOuts

| Name         | Type   | Length | Quantity | Units | Value | Description       |
|--------------|--------|--------|----------|-------|-------|-------------------|
| <i>angle</i> | double | 1      | Angle    | rad   | [0.0] | hinge angle value |

#### 2.6.2.4.11.3.4.4. PrescribedPin

Forces a hinge pin to follow prescribed angles (via flowins) for kinematics modes

- *Keywords:* "Hinge!Pin" "Hinge!Prescribed"
- *Class:* motor

If the prescribed angle exceeds specified min/max angles, they will be capped appropriately.

FlowIns

| Name         | Type   | Length | Quantity | Units | Value | Description                            |
|--------------|--------|--------|----------|-------|-------|--|
| <i>angle</i> | double | 1      | Angle    | rad   | [0.0] | prescribed hinge angle value (radians) |

Parameters

| Name   | Type       | Length | Quantity  | Units | Value  | Description   |
|--|------------|--------|-----------|-------|--------|---|
| <i>max_angle</i><br><i>max_angle_enabled</i> | double int | 1 1    | Angle N/A | rad   | 2.0 1  | Joint angle limit maximum (radians, defaults to no limit) 1 if min angle limit is enabled, 0 if not |
| <i>min_angle</i><br><i>min_angle_enabled</i> | double int | 1 1    | Angle N/A | rad   | -0.5 1 | Joint angle limit minimum (radians, defaults to no limit) 1 if min angle limit is enabled, 0 if not |

Continuous States  
None

Scratch

| Name                                       | Type    | Length | Quantity | Units | Value | Description   |
|--|---------|--------|----------|-------|-------|---|
| <i>at_max_angle</i><br><i>at_min_angle</i> | int int | 1 1    | N/A N/A  |       | 0 0   | 1 if angle is at maximum angle joint stop (0 otherwise) 1 if angle is at minimum angle joint stop (0 otherwise) |

States  
None

FlowOuts  
None

#### 2.6.2.4.11.3.4.5. PinEncoder

Outputs the pin joint angular displacement

- *Keywords:* "Hinge!Pin"
- *Class:* encoder

Author: David Henriquez The PinEncoder encoder reports the angular displacement a pin joint hinge has undergone (i.e. angle). If PinEncoder is attached to a hinge with more than one degree of freedom, only the first degree of freedom will be reported; however, such a scenario may cause a system error (i.e. segmentation fault).

FlowIns  
None

Parameters  
None

Continuous States  
None

Scratch  
None

States

| Name         | Type   | Length | Quantity | Units | Value | Description |
|--------------|--------|--------|----------|-------|-------|-------------|
| <i>angle</i> | double | 1      | Angle    | rad   | 0.0   |             |

FlowOuts

| Name         | Type   | Length | Quantity | Units | Value | Description       |
|--------------|--------|--------|----------|-------|-------|-------------------|
| <i>angle</i> | double | 1      | Angle    | rad   | [0.0] | hinge angle value |

#### 2.6.2.4.11.3.4.6. PrescribedPin

Forces a hinge pin to follow prescribed angles (via flowins) for kinematics modes

- *Keywords:* "Hinge!Pin" "Hinge!Prescribed"
- *Class:* motor

If the prescribed angle exceeds specified min/max angles, they will be capped appropriately.

FlowIns

| Name         | Type   | Length | Quantity | Units | Value | Description                            |
|--------------|--------|--------|----------|-------|-------|--|
| <i>angle</i> | double | 1      | Angle    | rad   | [0.0] | prescribed hinge angle value (radians) |

Parameters

| Name   | Type       | Length | Quantity  | Units | Value  | Description   |
|--|------------|--------|-----------|-------|--------|---|
| <i>max_angle</i><br><i>max_angle_enabled</i> | double int | 1 1    | Angle N/A | rad   | 2.0 1  | Joint angle limit maximum (radians, defaults to no limit) 1 if min angle limit is enabled, 0 if not |
| <i>min_angle</i><br><i>min_angle_enabled</i> | double int | 1 1    | Angle N/A | rad   | -0.5 1 | Joint angle limit minimum (radians, defaults to no limit) 1 if min angle limit is enabled, 0 if not |

Continuous States  
None

Scratch

| Name                                       | Type    | Length | Quantity | Units | Value | Description   |
|--|---------|--------|----------|-------|-------|---|
| <i>at_max_angle</i><br><i>at_min_angle</i> | int int | 1 1    | N/A N/A  |       | 0 0   | 1 if angle is at maximum angle joint stop (0 otherwise) 1 if angle is at minimum angle joint stop (0 otherwise) |

States  
None

FlowOuts  
None

#### 2.6.2.4.11.3.4.7. PinEncoder

Outputs the pin joint angular displacement

- *Keywords:* "Hinge!Pin"
- *Class:* encoder

Author: David Henriquez The PinEncoder encoder reports the angular displacement a pin joint hinge has undergone (i.e. angle). If PinEncoder is attached to a hinge with more than one degree of freedom, only the first degree of freedom will be reported; however, such a scenario may cause a system error (i.e. segmentation fault).

FlowIns  
None

Parameters  
None

Continuous States  
None

Scratch  
None

States

| Name         | Type   | Length | Quantity | Units | Value | Description |
|--------------|--------|--------|----------|-------|-------|-------------|
| <i>angle</i> | double | 1      | Angle    | rad   | 0.0   |             |

FlowOuts

| Name         | Type   | Length | Quantity | Units | Value | Description       |
|--------------|--------|--------|----------|-------|-------|-------------------|
| <i>angle</i> | double | 1      | Angle    | rad   | [0.0] | hinge angle value |

#### 2.6.2.4.11.3.4.8. PrescribedPin

Forces a hinge pin to follow prescribed angles (via flowins) for kinematics modes

- *Keywords:* "Hinge!Pin" "Hinge!Prescribed"
- *Class:* motor

If the prescribed angle exceeds specified min/max angles, they will be capped appropriately.

FlowIns

| Name         | Type   | Length | Quantity | Units | Value | Description                            |
|--------------|--------|--------|----------|-------|-------|--|
| <i>angle</i> | double | 1      | Angle    | rad   | [0.0] | prescribed hinge angle value (radians) |

Parameters

| Name   | Type       | Length | Quantity  | Units | Value  | Description   |
|--|------------|--------|-----------|-------|--------|---|
| <i>max_angle</i><br><i>max_angle_enabled</i> | double int | 1 1    | Angle N/A | rad   | 2.0 1  | Joint angle limit maximum (radians, defaults to no limit) 1 if min angle limit is enabled, 0 if not |
| <i>min_angle</i><br><i>min_angle_enabled</i> | double int | 1 1    | Angle N/A | rad   | -0.5 1 | Joint angle limit minimum (radians, defaults to no limit) 1 if min angle limit is enabled, 0 if not |

Continuous States  
None

Scratch

| Name                                       | Type    | Length | Quantity | Units | Value | Description   |
|--|---------|--------|----------|-------|-------|---|
| <i>at_max_angle</i><br><i>at_min_angle</i> | int int | 1 1    | N/A N/A  |       | 0 0   | 1 if angle is at maximum angle joint stop (0 otherwise) 1 if angle is at minimum angle joint stop (0 otherwise) |

States  
None

FlowOuts

None

#### 2.6.2.4.11.4. Related Regression Tests

##### Test the LinkAssembly with prescribed hinges

▶ [Click to see the DshellCommon/test/test\\_Ndarts/test\\_SerialLinksAssembly/script.py script](#)

##### Test the LinkAssembly with torque inputs

▶ [Click to see the DshellCommon/test/test\\_Ndarts/test\\_SerialLinksAssembly/script2.py script](#)

##### Test the LinkAssembly (with spring damper motors)

▶ [Click to see the DshellCommon/test/test\\_Ndarts/test\\_SerialLinksAssembly/script3.py script](#)

#### 2.6.2.4.11.5. Class API Documentation



##### Note

For Doxygen documentation, please see: [Serial Links Assembly<assemblies::SerialLinksAssembly::SerialLinksAssembly](#)  
([https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/DshellCommon/html/classpython\\_1\\_1assemblies\\_1\\_1SerialLinksAssembly\\_1\\_1SerialLinksAssembly.html](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/DshellCommon/html/classpython_1_1assemblies_1_1SerialLinksAssembly_1_1SerialLinksAssembly.html))

#### 2.6.2.4.12. SiteAssembly

##### 2.6.2.4.12.1. Class Documentation

###### 2.6.2.4.12.1.1. Introduction

The SiteAssembly provides a way to specify a site location (such as a launch site) on the surface of a planetary target. The SiteAssembly acts as a frameProvider for the generated coordinate frame located at the specified site. SiteAssembly is typically a member assembly of the associated TargetAssembly.

###### 2.6.2.4.12.1.2. Construction Options

- config - None

- context

SiteAssembly has one required context setting:

- 'TargetAssembly'

The name of the TargetAssembly that the site is on.

- signalTies - None

- param

SiteAssembly has two required parameters:

- 'Site'

This should point to a SiteParam parameter class object.

- 'Target'

This should point to a TargetParam parameter class object.

###### 2.6.2.4.12.1.3. Example Construction Syntax

```
config = {  
  EARTH : {  
    CLASS : TargetAssembly,  
    PARAMS : { TARGET : target_params },  
    ASSEMBLIES : {  
      Site1 : {  
        CLASS : SiteAssembly,  
        CONTEXT : { TARGET_ASSEMBLY : EARTH },  
        PARAMS : { TARGET : target_params,  
                  SITE : site_params  
        }, #PARAMS  
      }, # Site1  
    } # ASSEMBLIES  
  } # EARTH
```

PYTHON

###### 2.6.2.4.12.1.4. Signals

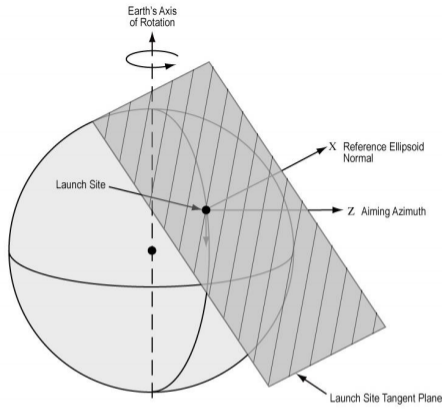
##### FlowIns

None

##### FlowOuts

None

###### 2.6.2.4.12.2. Visual Representation of Launch Site Frame



**TYPE:** Origin at launch site, plumb line, Earth-fixed

**ORIENTATION:**

- The x-axis coincides with the reference ellipsoid normal passing through the launch site, positive upward.
- The z-axis is parallel to the Earth-fixed aiming azimuth defined at Navigation Go-inertial, and is positive downrange.
- The y-axis completes a standard right-handed system.

**Note:** This system is identical to the Space-Fixed Launch Site System at Navigation Go-inertia.

It should be noted that the frame associated with any site (including a Landing Site) will have the same orientation as the Launch Site Frame shown above.

2.6.2.4.12.2.1. Related Regression Tests

**Unit test for the SiteAssembly Class.**

▶ [Click to see the `DshellCommon/test/test\\_Ndarts/test\\_SiteAssembly/script.py script](#)

**Unit test for the SiteAssembly Class with launch site.**

▶ [Click to see the `DshellCommon/test/test\\_Ndarts/test\\_SiteAssembly/script\\_LaunchSite.py script](#)

2.6.2.4.12.3. SiteAssembly Class API Documentation



**Note**

For Doxygen documentation, please see: [SiteAssembly<SiteAssembly::SiteAssembly>](#)  
([https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/DshellCommon/html/classpython\\_1\\_1assemblies\\_1\\_1SiteAssembly\\_1\\_1SiteAssembly.html](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/DshellCommon/html/classpython_1_1assemblies_1_1SiteAssembly_1_1SiteAssembly.html))

2.6.2.4.13. TargetAssembly

2.6.2.4.13.1. Class Documentation

The TargetAssembly class

2.6.2.4.13.2. TargetBaseAssembly Usage

2.6.2.4.13.2.1. Introduction

The TargetAssembly class

2.6.2.4.13.2.2. Signal Information

| Signal Name | Type   | Length | Quantity | Units | Description                              |
|-------------|--------|--------|----------|-------|--|
| MarsAngle   | double | 1      | Angle    | rad   | Truth data: rotation of target body Mars |

2.6.2.4.13.2.3. Param Class Information

**StatePosVelBaseParam**

State position (and velocity) knowledge/delivery specification for the start of a simulation.

The refVertical (see ref Reference\_Datums\_Verticals\_section) is used in the computation of initial attitude from flight-path and aero angle quantities.

Required Parameters

| Name | Type   | Length | Quantity | Units | Description      |
|------|--------|--------|----------|-------|------------------|
| name | string | 1      | N/A      |       | name of instance |

Optional Parameters

| Name | Type | Length | Quantity | Units | Description |
|------|------|--------|----------|-------|-------------|
|------|------|--------|----------|-------|-------------|

| Name               | Type   | Length | Quantity | Units | Description  |
|--------------------|--------|--------|----------|-------|--|
| <i>refVertical</i> | string | 1      | N/A      |       | direction defining vertical for bank angle 'RADIAL, ELLIPSOID_VERTICAL, ELLIPTIC_NORMAL', (defaults to RADIAL) |

#### StateAttitudeBaseParam

State attitude knowledge/delivery specification for the start of a simulation and attitude rate. When using this parameter class in a VehicleAssembly, the inertial frame in the descriptions for *initQ* and *initOmega* refers to the Planet-Centered Inertial (PCI) frame. When this class is used with the *stateInit* utility, the inertial frame refers to the frame provided by the assembly passed in as an argument.

#### Required Parameters

| Name         | Type   | Length | Quantity   | Units | Description  |
|--------------|--------|--------|------------|-------|--|
| <i>initQ</i> | double | 4      | Quaternion |       | initial attitude quaternion from body frame to inertial frame either directly specified or derived from angle data |
| <i>name</i>  | string | 1      | N/A        |       | name of instance   |

#### Optional Parameters

| Name                  | Type    | Length | Quantity        | Units | Description  |
|-----------------------|---------|--------|-----------------|-------|--|
| <i>initAdjustquat</i> | double  | 4      | Quaternion      |       | attitude quaternion adjustment                           |
| <i>initOmega</i>      | vector3 | 1      | AngularVelocity | rad/s | body angular velocity with respect to the inertial frame |
| <i>initOmegaBody</i>  | vector3 | 1      | AngularVelocity | rad/s | body angular velocity with respect to the body frame     |

#### TargetBaseParam

This class provides the parameters used to describe a central body or target.

This is the base class for all Target parameter classes.

#### Required Parameters

| Name                | Type   | Length | Quantity        | Units | Description   |
|---------------------|--------|--------|-----------------|-------|---|
| <i>mass</i>         | double | 1      | Mass            | kg    | Mass of target  |
| <i>name</i>         | string | 1      | N/A             |       | Name of the target body   |
| <i>rotationRate</i> | double | None   | AngularVelocity | rad/s | Rotation rate of the target. May be a single float or three element list of floats. |

#### Optional Parameters

| Name                 | Type   | Length | Quantity      | Units | Description  |
|----------------------|--------|--------|---------------|-------|--|
| <i>eccentricity</i>  | double | 1      | Dimensionless |       | Eccentricity of the target body ellipsoid (optional) |
| <i>flattening</i>    | double | 1      | Dimensionless |       | Ellipsoidal flattening of target body (optional)     |
| <i>radiusEquator</i> | double | 1      | Length        | m     | Equatorial radius of target body                     |
| <i>radiusPole</i>    | double | 1      | Length        | m     | Polar radius of target body (optional)               |
| <i>rotationAngle</i> | double | 1      | Angle         | rad   | Rotation angle of the target                         |

## BodyParam

The **BodyParam** class provides all parameters required to specify a single Darts body.

### Required Parameters

| Name        | Type   | Length | Quantity | Units | Description   |
|-------------|--------|--------|----------|-------|---------------|
| <i>mass</i> | double | 1      | Mass     | kg    | The body mass |

### Optional Parameters

| Name                         | Type    | Length | Quantity         | Units | Description  |
|------------------------------|---------|--------|------------------|-------|--|
| <i>bodyToCM</i>              | vector3 | 1      | Length           | m     | The 3-vector to the body CM from the body frame.   |
| <i>bodyToJoint</i>           | vector3 | 1      | Length           | m     | The 3-vector from the body frame to the body hinge   |
| <i>bodyToJointQuat</i>       | double  | 4      | Quaternion       |       | The quaternion from the body frame to the pnode for the body hinge   |
| <i>cmInertia</i>             | double  | None   | MomentsOfInertia |       | The 3x3 body inertia matrix about the body CM (do not specify both inertia and cmInertia)  |
| <i>description</i>           | string  | 1      | N/A              |       | DEPRECATED: Body description. IGNORED.   |
| <i>gearRatio</i>             | double  | 1      | Dimensionless    |       | Gear ratio for PIN joints  |
| <i>geometry</i>              | {}      | None   | N/A              |       | Geometry to be loaded into DScene.   |
| <i>inbToJoint</i>            | vector3 | 1      | Length           | m     | The 3-vector from the inboard body frame to the body hinge   |
| <i>inbToJointQuat</i>        | double  | 4      | Quaternion       |       | The quaternion from the inboard body frame to the body hinge   |
| <i>inertia</i>               | double  | None   | MomentsOfInertia |       | The 3x3 body inertia matrix about the body frame (do not specify both inertia and cmInertia)   |
| <i>jointAxes</i>             | vector3 | *      | Dimensionless    |       | The hinge rotation/translation axis generalized coordinate [min, max]  |
| <i>jointLimits</i>           | double  | *      | N/A              |       | Limits for range of joint generalized coordinate [min, max]  |
| <i>jointType</i>             | string  | 1      | N/A              |       | The hinge type (string). Supported: BALL, COMPOSITE-TRANSLATIONAL, FULL6DOF, FULL6DOF_INERTIAL, GIMBAL, LOCKED, PIN, PLANAR, SLIDER, TRANSLATIONAL, UJOINT, CUSTOM |
| <i>negativeIntegralSense</i> | bool    | 1      | N/A              |       | Body products of inertia values expressed in NEGATIVE inertia integral sense (defaults to True)  |

| Name                  | Type | Length | Quantity | Units | Description   |
|-----------------------|------|--------|----------|-------|---|
| <i>partGraphics</i>   | {}   | None   | N/A      |       | DEPRECATED: Geometry to be loaded into DScene.  |
| <i>prescribed</i>     | bool | *      | N/A      |       | Whether the joint is prescribed or free (scalar boolean or array of booleans for each subhinge) |
| <i>prescribedType</i> |      | None   | N/A      |       | DEPRECATED: Type of prescribed joint. WARNING: Not permitted in Ndarts                          |
| <i>subhinge</i>       |      | None   | N/A      |       | Subhinge specification for custom hinge.  |

### TargetAdditionalFrameParam

Parameters for additional target-centered frame specification.

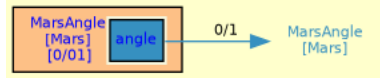
Required Parameters

| Name                  | Type   | Length | Quantity      | Units | Description  |
|-----------------------|--------|--------|---------------|-------|--|
| <i>frameTransform</i> | double | None   | Dimensionless |       | Frame transform of additional target-centered frame from the PCI frame |

Optional Parameters

None

2.6.2.4.13.2.4. Model Information



### PinEncoder

Outputs the pin joint angular displacement

- *Keywords:* "Hinge!Pin"
- *Class:* encoder

Author: David Henriquez The PinEncoder encoder reports the angular displacement a pin joint hinge has undergone (i.e. angle). If PinEncoder is attached to a hinge with more than one degree of freedom, only the first degree of freedom will be reported; however, such a scenario may cause a system error (i.e. segmentation fault).

FlowIns

None

Parameters

None

Continuous States

None

Scratch

None

States

| Name         | Type   | Length | Quantity | Units | Value | Description |
|--------------|--------|--------|----------|-------|-------|-------------|
| <i>angle</i> | double | 1      | Angle    | rad   | 0.0   |             |

FlowOuts

| Name | Type | Length | Quantity | Units | Value | Description |
|------|------|--------|----------|-------|-------|-------------|
|------|------|--------|----------|-------|-------|-------------|



| Name         | Type   | Length | Quantity | Units | Value | Description       |
|--------------|--------|--------|----------|-------|-------|-------------------|
| <i>angle</i> | double | 1      | Angle    | rad   | [1,0] | hinge angle value |

#### 2.6.2.4.13.3. Related Regression Tests

- Create a vehicle with 3 bodies and a planetary target body. Set up simple gravity.
- Simulate docking between 2 vehicles Create 2 vehicles. In the simulation one vehicle attaches to the other vehicle to roughly simulate docking a vehicle to the space station.
- Vehicle with aspherical gravity Create a vehicle with 3 bodies and a planetary target body. Set up an aspherical gravity tied to the planetary body.
- Vehicle with external disturbance Create a free-flying vehicle without gravity and apply a force and torque to test ExternalDisturbanceActuatorAssembly.

#### 2.6.2.4.13.4. TargetAssembly Class API Documentation



##### Note

For Doxygen documentation, please see: [TargetAssembly<TargetAssembly::TargetAssembly>](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/DshellCommon/html/classpython_1_1assemblies_1_1TargetAssembly_1_1TargetAssembly.html)  
([https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/DshellCommon/html/classpython\\_1\\_1assemblies\\_1\\_1TargetAssembly\\_1\\_1TargetAssembly.html](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/DshellCommon/html/classpython_1_1assemblies_1_1TargetAssembly_1_1TargetAssembly.html))

Derives from the TargetBaseAssembly class and requires body parameters by setting the Bodies config flag to true

#### 2.6.2.4.14. TargetBaseAssembly

##### 2.6.2.4.14.1. Class Documentation

The TargetBaseAssembly models some target.

The TargetBaseAssembly class is derived from the Assembly and FrameProvider classes and will be the parent class of all target assemblies.

##### 2.6.2.4.14.1.1. Construction Options

- **config** -
  - **'Bodies'**
- **context** - None
- **signalTies** - None
- **param**

TargetBaseAssembly has one required parameter:

- **'Target'**

This should point to a TargetBaseParam parameter class object. This is usually stored in a file in the library of your run script such as: './library/targets.py'. This information is used to initialize basic information about the target such as the initial rotational position of the target.

TargetBaseAssembly has additional required parameters depending on config type:

- **'Bodies'**

This should point to a dictionary of BodyParam parameter class objects. The TargetAssembly object create a 'PCI

TargetBaseAssembly has one optional parameter:

- *AdditionalFrames*

This must be a dictionary with string keys starting from '0' to the number of additional target-centered frames. The value associated with each string-based integer key must be an instance of TargetAdditionalFrameParam.

##### 2.6.2.4.14.2. Usage Scenario

This is a simple usage of the TargetBaseAssembly. Start with the standard imports and import the Assembly. Create the Sim with SimulationExecutive.

#### Simple TargetBaseAssembly usage regtest

- ▶ [Click to see the DshellCommon/usage/usage\\_TargetBaseAssembly.py script](#)

##### 2.6.2.4.14.3. Usage

##### 2.6.2.4.14.3.1. Introduction

The TargetAssembly class

##### 2.6.2.4.14.3.2. Signal Information

| Signal Name      | Type   | Length | Quantity | Units | Description                              |
|------------------|--------|--------|----------|-------|--|
| <i>MarsAngle</i> | double | 1      | Angle    | rad   | Truth data: rotation of target body Mars |

##### 2.6.2.4.14.3.3. Param Class Information

##### 2.6.2.4.14.3.3.1. StatePosVelBaseParam

“State position (and velocity) knowledge/delivery specification for the start of a simulation.

The *refVertical* (see *ref Reference\_Datums\_Verticals\_section*) is used in the computation of initial attitude from flight-path and aero angle quantities.

#### Required Parameters

| Name        | Type   | Length | Quantity | Units | Description      |
|-------------|--------|--------|----------|-------|------------------|
| <i>name</i> | string | 1      | N/A      |       | name of instance |

#### Optional Parameters

| Name               | Type   | Length | Quantity | Units | Description  |
|--------------------|--------|--------|----------|-------|--|
| <i>refVertical</i> | string | 1      | N/A      |       | direction defining vertical for bank angle 'RADIAL, ELLIPSOID_VERTICAL, ELLIPTIC_NORMAL', (defaults to RADIAL) |

#### 2.6.2.4.14.3.3.2. StateAttitudeBaseParam

“State attitude knowledge/delivery specification for the start of a simulation and attitude rate. When using this parameter class in a *VehicleAssembly*, the inertial frame in the descriptions for *initQ* and *initOmega* refers to the Planet-Centered Inertial (PCI) frame. When this class is used with the *stateInit* utility, the inertial frame refers to the frame provided by the assembly passed in as an argument.

##### Required Parameters

| Name              | Type          | Length | Quantity       | Units | Description   |
|-------------------|---------------|--------|----------------|-------|---|
| <i>initQ name</i> | double string | 4 1    | Quaternion N/A |       | initial attitude quaternion from body frame to inertial frame either directly specified or derived from angle data name of instance |

##### Optional Parameters

| Name                  | Type    | Length | Quantity        | Units | Description  |
|-----------------------|---------|--------|-----------------|-------|--|
| <i>initAdjustquat</i> | double  | 4      | Quaternion      |       | attitude quaternion adjustment                           |
| <i>initOmega</i>      | vector3 | 1      | AngularVelocity | rad/s | body angular velocity with respect to the inertial frame |
| <i>initOmegaBody</i>  | vector3 | 1      | AngularVelocity | rad/s | body angular velocity with respect to the body frame     |

#### 2.6.2.4.14.3.3.3. TargetBaseParam

“This class provides the parameters used to describe a central body or target.

*This is the base class for all Target parameter classes.*

##### Required Parameters

| Name                | Type          | Length | Quantity        | Units | Description   |
|---------------------|---------------|--------|-----------------|-------|---|
| <i>mass name</i>    | double string | 1 1    | Mass N/A        | kg    | Mass of target Name of the target body  |
| <i>rotationRate</i> | double        | None   | AngularVelocity | rad/s | Rotation rate of the target. May be a single float or three element list of floats. |

##### Optional Parameters

| Name                           | Type          | Length | Quantity                       | Units | Description  |
|--------------------------------|---------------|--------|--------------------------------|-------|--|
| <i>eccentricity flattening</i> | double double | 1 1    | Dimensionless<br>Dimensionless |       | Eccentricity of the target body ellipsoid (optional)<br>Ellipsoidal flattening of target body (optional) |
| <i>radiusEquator</i>           | double        | 1      | Length                         | m     | Equatorial radius of target body   |
| <i>radiusPole</i>              | double        | 1      | Length                         | m     | Polar radius of target body (optional)   |
| <i>rotationAngle</i>           | double        | 1      | Angle                          | rad   | Rotation angle of the target   |

#### 2.6.2.4.14.3.3.4. BodyParam

“The **BodyParam** class provides all parameters required to specify a single Darts body.

##### Required Parameters

| Name        | Type   | Length | Quantity | Units | Description   |
|-------------|--------|--------|----------|-------|---------------|
| <i>mass</i> | double | 1      | Mass     | kg    | The body mass |

##### Optional Parameters

| Name  | Type   | Length                              | Quantity  | Units | Description  |
|---|--|-------------------------------------|---|-------|--|
| <i>bodyToCM</i>   | vector3  | 1                                   | Length  | m     | The 3-vector to the body CM from the body frame.   |
| <i>bodyToJoint</i><br><i>bodyToJointQuat</i> <i>cmInertia</i><br><i>description</i> <i>gearRatio</i><br><i>geometry</i>   | vector3 double double<br>string double \{\}                      | 1 4 None 1 1 None                   | Length Quaternion<br>MomentsOfInertia N/A<br>Dimensionless N/A                        | m     | The 3-vector from the body frame to the body hinge The quaternion from the body frame to the pnode for the body hinge The 3x3 body inertia matrix about the body CM (do not specify both inertia and cmInertia) DEPRECATED: Body description. IGNORED. Gear ratio for PIN joints Geometry to be loaded into DScene.  |
| <i>inbToJoint</i> <i>inbToJointQuat</i><br><i>inertia</i> <i>jointAxes</i> <i>jointLimits</i><br><i>jointType</i><br><i>negativeIntegralSense</i><br><i>partGraphics</i> <i>prescribed</i><br><i>prescribedType</i> <i>subhinge</i> | vector3 double double<br>vector3 double string bool \{\}<br>bool | 1 4 None ** 1 1 None * None<br>None | Length Quaternion<br>MomentsOfInertia<br>Dimensionless N/A N/A N/A<br>N/A N/A N/A N/A | m     | The 3-vector from the inboard body frame to the body hinge The quaternion from the inboard body frame to the body hinge The 3x3 body inertia matrix about the body frame (do not specify both inertia and cmInertia) The hinge rotation/translation axis Limits for range of joint generalized coordinate [min, max] The hinge type (string). Supported: BALL, COMPOSITE-TRANSLATIONAL, FULL6DOF, FULL6DOF_INERTIAL, GIMBAL, LOCKED, PIN, PLANAR, SLIDER, TRANSLATIONAL, UJOINT, CUSTOM Body products of inertia values expressed in NEGATIVE inertia integral sense (defaults to True) DEPRECATED: Geometry to be loaded into DScene. Whether the joint is prescribed or free (scalar boolean or array of booleans for each subhinge) DEPRECATED: Type of prescribed joint. WARNING: Not permitted in Ndarts Subhinge specification for custom hinge. |

#### 2.6.2.4.14.3.3.5. TargetAdditionalFrameParam

 Parameters for additional target-centered frame specification.


##### Required Parameters

| Name                  | Type   | Length | Quantity      | Units | Description  |
|-----------------------|--------|--------|---------------|-------|--|
| <i>frameTransform</i> | double | None   | Dimensionless |       | Frame transform of additional target-centered frame from the PCI frame |

##### Optional Parameters

None

#### 2.6.2.4.14.3.4. Model Information

 Model data flow

#### 2.6.2.4.14.3.4.1. PinEncoder

Outputs the pin joint angular displacement

- *Keywords:* "Hinge!Pin"
- *Class:* encoder

Author: David Henriquez The PinEncoder encoder reports the angular displacement a pin joint hinge has undergone (i.e. angle). If PinEncoder is attached to a hinge with more than one degree of freedom, only the first degree of freedom will be reported; however, such a scenario may cause a system error (i.e. segmentation fault).

FlowIns  
None

Parameters  
None

Continuous States  
None

Scratch  
None

States

| Name         | Type   | Length | Quantity | Units | Value | Description |
|--------------|--------|--------|----------|-------|-------|-------------|
| <i>angle</i> | double | 1      | Angle    | rad   | 0.0   |             |

FlowOuts  
 ===== Name Type Length Quantity Units Value Description =====  
 =====

| Name         | Type   | Length | Quantity | Units | Value | Description       |
|--------------|--------|--------|----------|-------|-------|-------------------|
| <i>angle</i> | double | 1      | Angle    | rad   | [1.0] | hinge angle value |

#### 2.6.2.4.14.4. Class API Documentation



**Note**

For Doxygen documentation, please see: [Target Base Assembly<assemblies::TargetBaseAssembly::TargetBaseAssembly>](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/DshellCommon/html/classpython_1_1assemblies_1_1TargetBaseAssembly_1_1TargetBaseAssembly.html)  
([https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/DshellCommon/html/classpython\\_1\\_1assemblies\\_1\\_1TargetBaseAssembly\\_1\\_1TargetBaseAssembly.html](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/DshellCommon/html/classpython_1_1assemblies_1_1TargetBaseAssembly_1_1TargetBaseAssembly.html))

#### Target Base Assembly

This base assembly includes functions that create both topo dems and darts bodies as targets.

#### 2.6.2.4.15. TargetSpiceAssembly

##### 2.6.2.4.15.1. Class Documentation 2.6.2.4.15.1.1. Introduction

The TargetSpiceAssembly allows the user to create a TargetAssembly using celestial body/frame information from SPICE.

##### 2.6.2.4.15.1.2. Construction Options

- config - TargetSpiceAssembly has one optional config setting. Inherited from its base class, TargetBaseAssembly:
  - **'inertial\_frame'**  
The name of the frame that will be used as the reference inertial frame. Only one Target assembly can have the 'inertial\_frame' config set in a simulation.
- context - None
- signalTies - None
- param  
TargetSpiceAssembly has two required parameters:
  - **'Target'**  
This parameter is inherited from its base class, TargetBaseAssembly. This should point to a TargetSpiceParam parameter class object.
  - **'Epoch'**  
This should point to a EpochDateParam parameter class object.

##### 2.6.2.4.15.1.3. Example Construction Syntax

```
'Earth' : {
    'class' : 'TargetSpiceAssembly',
    'inertial_frame' : 'PCI',
    'params' : { 'Target' : params.TargetSpiceParam(
        params={'name' : 'Earth',
                'bodyID' : 399,
                'mass' : 3.986004415e14 / 6.67259e-11,
                'rotationRate' : 7.29211514646e-5, # rad/s
                'rotationAngle' : 4.11421137, # Rotation angle from midnight to launch time
                'radiusPole' : 6356784.283104,
                'radiusEquator' : 6378166.00, # 6378140.000616, #grav 6378136.3;
                'kernels' : [
                    '../common/spiceKernels/pck00009.tpc',
                    '../common/spiceKernels/Gravity.tpc',
                    '../common/spiceKernels/naif0009.tls',
                    '../common/spiceKernels/ofDate.fk',
                ],
                # 'additionalFrameNames' : ['J2000', 'B1950', 'TOD_I']
            }, # Target params
        source = None ),
        'Epoch' : params.EpochDateParam(
            params = {'name' : '1st Successful Run!',
                    'initDate' : "8/22/2012 08:01:04.184 (TDT)",})
        ), # Earth params
    }, #Earth
```

#### 2.6.2.4.15.1.4. Signals

The `TargetSpiceAssembly` has a `spiceTime` signal that is the time value from the Spice Frame Container.

#### 2.6.2.4.15.1.5. Related Regression Tests

- Vehicles with Spice targets. Create a vehicle with 3 bodies and a planetary target body initialized with Spice. Set up simple gravity.

#### 2.6.2.4.15.2. TargetSpiceAssembly Class API Documentation



##### Note

For Doxygen documentation, please see: [TargetSpiceAssembly<TargetSpiceAssembly>:TargetSpiceAssembly](#)

([https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/DshellCommon/html/classpython\\_1\\_1assemblies\\_1\\_1TargetSpiceAssembly\\_1\\_1TargetSpiceAssembly.html](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/DshellCommon/html/classpython_1_1assemblies_1_1TargetSpiceAssembly_1_1TargetSpiceAssembly.html))

#### Target body assembly using Spice data

The J2000 frame is centered at the solar system barycenter but is oriented to be parallel to Earth's ecliptic axes at the J2000 epoch time, eg, January 1, 2000 at 12:00:00 TDB. Z-J2000 is the spin axis of the Earth (northwards), and X-J2000 is the intersection of equatorial and ecliptic planes (called the vernal equinox), outwards from the center of the planet.

### 2.6.2.5. DshellCommon API Reference - Other

#### 2.6.2.5.1. Creating an Assembly Class

All Assembly classes should be derived from `BaseAssembly` or a base class derived from `BaseAssembly`. Here is a simplified example of how an assembly should be defined.

```
from DshellCommon.BaseAssembly import BaseAssembly
from DshellCommon.params.ExampleParam import ExampleParam

PARAM1 = 'param1'

class ExampleAssembly(BaseAssembly):
    """ExampleAssembly"""

    _param_types = { PARAM1 : ExampleParam
                    }

    def __init__(self, parent_asm, name, config, context, signal_ties=None, params=None):

        BaseAssembly.__init__(self, parent_asm, name, config, context, signal_ties)

        self.requiredConfigFields([])
        self.optionalConfigFields([])

        self.requiredContextFields([])
        self.optionalContextFields([])

        self.requiredParamFields([PARAM1])
        self.optionalParamFields([])

        # Save the parameters, if given
        if params:
            self.params(params)
```

Notice the constant 'PARAM1' defined at the top level of the module (line 5). This is used in the Assembly code in two locations (lines 9 and 22). Defining this constant at the module level makes it easier to import and use in run scripts or library files for run scripts.

The '`_param_types`' class variable dictionary contains the expected parameter keys and the corresponding parameter classes for each one. All parameters that could possibly be used in the assembly class must be defined here. As each parameter is processed, it is checked to verify that is the same type (or of a derived type) as is specified in the '`_param_types`' dictionary.

Depending on the signals, models, and assemblies this assembly needs to create, it will be necessary to define additional member functions such as '`addSignals()`', '`addModels()`', and '`addAssemblies()`'.

#### 2.6.2.5.1.1. Derived Assembly Classes

Derived classes can add additional parameters as needed. Suppose we derive a new assembly class 'NewAssembly' from the 'ExampleAssembly' Class. This excerpt would show how to do that:

```
CONST1 = 'param2'
```

```
class NewAssembly(ExampleAssembly):  
    _params = ExampleAssembly.addParams({  
        CONST1 : ParamInfo( required = True,  
                            units = UNITLESS,  
                            description = 'const2',  
                            data_type = float  
                        )  
    })  
    etc...
```

### 2.6.2.5.2. DshellCommonFSM

#### 2.6.2.5.2.1. Class Documentation

##### 2.6.2.5.2.1.1. Introduction

The DshellCommonFSM class is the primary class for finite state machine operations.

- Derived from the C++ Assembly class
- Provides a layer over the Python FSM class (from DshellEnv/python/Dfsm.py) for any needed additional functionality.
- Etc (TBD)

##### 2.6.2.5.2.1.2. Related Regression Tests

- Standalone FSM regtest Create a finite state machine based on DshellCommonFSM and exercise its functionality in a stand-alone test (without a simulator).
- FSM within a simulation regtest Create a finite state machine based on DshellCommonFSM and exercise its functionality in a simulation. Check basic transitions and termination during simulation.
- FSM in a simulation with transitions regtest Create a finite state machine based on DshellCommonFSM and exercise its functionality in a simulation. Check basic transitions and termination during simulation using the 'generalTrigger' trigger function from FsmUtils.py

##### 2.6.2.5.2.1.3. DshellCommonFSM Class API Documentation



#### Note

For Doxygen documentation, please see: [DshellCommonFSM<DshellCommonFSM::DshellCommonFSM>](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/DshellCommon/html/classpython_1_1fsm_1_1DshellCommonFSM_1_1DshellCommonFSM.html)  
([https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/DshellCommon/html/classpython\\_1\\_1fsm\\_1\\_1DshellCommonFSM\\_1\\_1DshellCommonFSM.html](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/DshellCommon/html/classpython_1_1fsm_1_1DshellCommonFSM_1_1DshellCommonFSM.html))

DshellCommonFSM finite state machine class

### 2.6.2.5.3. FsmUtils

#### 2.6.2.5.3.1. Function and Class Documentation

##### 2.6.2.5.3.1.1. Introduction

The FsmUtils module provides various utility functions and classes in support of finite state machine operations.

- generalTrigger() is a class whose instances support FSM triggering associated with predefined thresholds (including slope of value trend).

##### 2.6.2.5.3.1.2. DshellCommonFSM Class API Documentation

Finite State Machine utility functions

### 2.6.2.5.4. Knobs

#### 2.6.2.5.4.1. Introduction

Contains specifications of input values represented by "knob" objects. For example, to represent an input value taken from the sequence 1,2,3:

```
knob1 = SequenceKnob('knob1_name', [1,2,3])
```

"knob1" is a unique id to identify the knob within the "Knobs" section.

"knob1\_name" is a unique string id used to retrieve the current Knob value from within testDsends.py.

DMonteCarlo passes the current knob values as a Knobs dictionary object so, in testDsends.py, you would retrieve the current knob value using "Knobs['knob1\_name']"

Knobs are usually defined in Knob configuration files. See DshellKnobsConfigFiles for more details.

##### 2.6.2.5.4.1.1. Available Types of Knobs

Input values may be specified through a python Knob class object.

The following Knob types are available:

#### SequenceKnob

The SequenceKnob object returns a value from a specified list in sequence order.

Each job takes a value from the specified in sequential order. No two jobs will use the same list value. The maximum number of jobs is therefore equal to the length of the SequenceKnob with the shortest list.

```
param name
  A unique id string to identify this Knob object

param List
  The list of values
```

### SequenceReuseKnob

The SequenceReuse object returns a value from a specified list.

Each job will “step” through each list value in the sequential order. After the end of the list is reached, the list will restart from the beginning. The list is reused as necessary and does not constrain the number of overall jobs executed.

```
param name
  A unique id string to identify this Knob object

param List
  The list of values
```

### ListKnob

The ListKnob object returns a value from a specified list.

It will generate all possible combinations of list values so the same ListKnob value can be used in more than one job.

```
param name
  A unique id string to identify this Knob object

param List
  A Python list of values
```

### GaussianKnob

The GaussainKnob object returns a random number using a gaussian distribution.

```
param name
  A unique id string to identify this Knob object

param mean
  Mean value (floating point)

param sigma
  Standard deviation (floating point)

param firstVal
  First value that will be returned by this Knob

param seed
  Random generator seed
```

### RandomKnob

The RandomKnob object returns a value randomly selected from a list.

```
param List
  List of values

param firstVal
  First value that will be returned by this Knob

param seed
  Random generator seed
```

### RandomGaussianVectorKnob

The RandomGaussianVectorKnob returns lists of random gaussian numbers.

```
param name
  A unique id string to identify this Knob object

param meanList
  Mean values of gaussian vector

param sigmaList
  Standard deviation of vector

param firstVal
  First value that will be returned by this Knob

param seed
  Random generator seed
```

## UniformKnob

The UniformKnob object returns a random number using a uniform distribution within a specified range.

```
param name
  A unique id string to identify this Knob object

param rmin, rmax
  Range of values (inclusive)

param firstVal
  First value that will be returned by this Knob

param seed
  = random generator seed
```

## CovarianceKnob

The CovarianceKnob object produces values that are part of a covariance dispersion.

```
param name
  A unique id string to identify this Knob object

param covarianceGroup
  Reference to the CovarianceDispersion instance that creates this knob. See CovarianceDispersion for more details.
```

## ConstantKnob

The ConstantKnob object returns the same value for each job.

```
param name
  A unique id string to identify this Knob object

param value
  Constant value to that this knob returns

param firstVal
  First value that will be returned by this Knob. If firstVal != None, the first value generated
  will be firstVal, the subsequent generated values will be constantValue
```

### 2.6.2.5.4.1.2. CREATING NEW KNOB CLASSES

You may create new Knob classes by deriving a new class from the base class Knobs.

The base Knob construction (*init*) method requires two parameters: the name of the Knob object and a Python generator function. The Knob object calls the generator function to generate the "next" value required by a job.

For example, here's the source code that defines a Knob that returns a value from a list of values in sequential order:

```
class SequenceKnob(Knob)::
  def __init__(self,name,sequence)::
    Knob.__init__(self,name,lambda: iter(sequence))
```

PYTHON

### 2.6.2.5.4.1.3. Related Regression Tests

### 2.6.2.5.4.2. Knob API Documentation

#### Note

For Doxygen documentation, please see:

- Knob<utils::Knobs::Knob>
- MultiRun<utils::MultiRun::MultiRun>
- SequenceKnob<utils::Knobs::SequenceKnob>
- SequenceResuseKnob<utils::Knobs::SequenceReuseKnob>
- ListKnob<utils::Knobs::ListKnob>
- GaussianKnob<utils::Knobs::GaussianKnob>
- RandomKnob<utils::Knobs::RandomKnob>
- RandomGaussianVectorKnob<utils::Knobs::RandomGaussianVectorKnob>
- UniformKnob<utils::Knobs::UniformKnob>
- CovarianceKnob<utils::Knobs::CovarianceKnob>
- ConstantKnob<utils::Knobs::ConstantKnob>



The following detailed API documentation includes additional information

Knob class.

### 2.6.2.5.5. Knob Configuration Files



### 2.6.2.5.5.1. Introduction

Knobs are usually defined in knob configuration files. These are 'configObj' files that define the knobs, how many runs are expected, and optionally some overrides.

### 2.6.2.5.5.2. How to Define Knobs with Knob Configuration Files

The user's run script would execute the following commands (where 'run\_number' comes into the script from its invocation, eg, via command line argument):

```
from DshellCommon.utils.Knobs import Knob, createKnobsFromFile
from DshellCommon.utils.MultiRun import MultiRun
listOfKnobs = createKnobsFromFile('ParamSweep.cfg')
multiRun = MultiRun(output_directory='results', run=run_number,
                    config_file_name='ParamSweep.cfg')
```

PYTHON

where the MultiRun class helps the knobs get the correct values for a specific run when many runs are being executed as part of a parameter sweep or Monte Carlo simulation.

Then the knob values for this run can be accessed like this:

```
print Knob.value('ListKnob1')
```

PYTHON

### 2.6.2.5.5.2.1. Example Knob Configuration File (ParamSweep.cfg)

```
# Example Parameter Sweep Configuration File

[System]

maxjobs = 16

[Knobs]

ListKnob1 = '''ListKnob(
    "ListKnob1", [ 18.0, 20.0, 22.0, 24.0 ]
)'''

GaussianKnobWithSeed = '''GaussianKnob(
    "GaussianKnobWithSeed", mean=100.0,sigma=2.0,
    firstVal = 100.0, seed = 1
)'''
```

CFG

Note that the config-file labels or names on the left of each knob equals sign are not significant but must not be repeated. Each label must be distinct in order for the format of the configuration file to be valid, but the actual names of the knobs are always taken from the first constructor argument. In other words, the first knob could have been defined as follows:

```
...

[Knobs]

k1 = '''ListKnob(
    "ListKnob1", [ 18.0, 20.0, 22.0, 24.0 ]
)'''
```

CFG

In this case, the knob name will be 'ListKnob1' (not 'k1').

### Optional 'frozen' or 'unfrozen' sections

In some cases you would like to freeze one (or more) of the knobs to their first value. (Where the first value is defined using the 'firstVal' option in the knob definition or is the first item in the value list if 'firstVal' is omitted). To do that, you may add an 'Override' section (where the case of the 'Override' is important) at the end of the knobs configuration file:

```
... (same as above)

[Override]

frozen = ['GaussianKnobWithSeed']
```

CFG

where `frozen` is a comma-separated list of names of knobs to freeze. In this case, the GaussianKnobWithSeed knob will remain frozen to its 'firstVal' but the other knobs will vary normally.

On the other hand, if you want to freeze everything except a set of knobs, you can use the `unfrozen` option:

```
[Override]

unfrozen = ['GaussianKnobWithSeed']
```

CFG

In this case, the GaussianKnobWithSeed knob will vary normally, but all other knobs will be frozen. Again, `unfrozen` could be a comma-separated list of knobs that you want to vary while leaving all the others frozen.



#### Warning

You cannot specify BOTH `frozen` and `unfrozen` sections in the knobs configuration file.

### 2.6.2.5.5.2.2. Related Regression Tests

#### 2.6.2.5.6. CovarianceDispersion

### 2.6.2.5.6.1. Introduction

`./regtests/test_MultiRun` `./regtests/test_MultiRun_frozen`

A utility for creating and accessing a covariance based dispersion.

#### 2.6.2.5.6.2. Example Usage Syntax

To create a CovarianceDispersion, you could use code like this:

```
covar = CovarianceDispersion('4x4Dispersion',          # name
                              [['value1', 2.0], ['value2', 4.0], ['value3', 6.0], ['value4', 8.0]], # initialValues
                              [[25, 15, -5, 1],        # covarianceMatrix
                               [18, 0, 1],
                               [11, 1],
                               [1]],
                              'UNIFORM',              # distribution
                              2                       # seedVal
                              )
```

PYTHON

Here is some more information about the CovarianceDispersion class and its construction arguments:

A covariance dispersion is created by instantiating a CovarianceDispersion object.

Initializes the data members, checks for input argument errors, and creates the knobs that are used to access the values.

The object constructor takes the following arguments:

```
param name
    The name of the dispersion instance being created

param initial_values
    A list of variable names and their initial (undispersed) values.

param covariance_matrix
    The covariance matrix input as an upper triangle.

param distribution
    The desired distribution type, either 'NORMAL', 'UNIFORM', or 'EXPONENTIAL'.

param seedVal
    The desired seed value. This is optional and will ensure that any given run always has the same value.
```

Here is an example of using a one of the dispersed variables defined in a the example CovarianceDispersion knob in a parameter library file:

```
'defaultActuator' : {
  'class' : 'BaseActuatorAssembly',
  'context' : {'bodyName': 'testBody'},
  'params' : {
    'actuator' : ActuatorBaseParam({
      'actuatorType' : 'PERFECT',
      'initialDefl' : Knob.value('value3'),
      'actuatorTau' : 20,
      'dampingRatio' : 0.7,
      'naturalFrequency' : 150,
      'cycleTime' : 0.00001,
      'deflectionBias' : 0.0, }),
    } # params
  }, # actuator
```

PYTHON

To use the dispersed data in a simulation, reference it as you would a regular knob.

#### 2.6.2.5.6.3. CovarianceDispersion Class API Documentation



##### Note

For Doxygen documentation, please see: `CovarianceDispersion<utils::CovarianceDispersion::CovarianceDispersion>`

A class for creating a covariance based dispersion.

#### 2.6.2.5.7. ExecutionSummary

##### 2.6.2.5.7.1. Introduction

A utility for creating an Execution Summary report file. The purpose of the Execution Summary report file is to have a record of every simulation run that can be used for debugging by both developers and users, and to have a record for posterity of run times, etc of various simulations across various COMPASS versions and host machines.

##### 2.6.2.5.7.2. Contents Description

###### 2.6.2.5.7.2.1. Simulation Info

COMPASS Version: CompassPkg-R#-### Host used: host\_name / CPU Utilization: ## Script used: script\_name Output data location: output\_directory

COMPASS Version will tie a given run to a version for debugging purposes. Host used will identify the host machine that was used in execution, and give the CPU usage level prior to execution. Script used lists the name of the executed script. Output data location gives the directory where any output files that are generated will appear.

###### 2.6.2.5.7.2.2. Execution Time

Sim Start: mm/dd/yyyy hh:mm:ss (clock time) Sim End: mm/dd/yyyy hh:mm:ss (clock time) Duration: hh:mm:ss (clock time) Processor time: hh:mm:ss

##### Execution Time Breakdown

Initialization: hh:mm:ss (clock time) Sim Object Creation: hh:mm:ss (clock time) Execution: hh:mm:ss (clock time)

Time information will measure various aspects of execution timing, for the purposes of debugging and performance analysis.

#### 2.6.2.5.7.2.3. Error / Warning / Phase Messages

Sim Time, Category, Message

Error, Warning, and Phase messages will be logged for debugging purposes.

#### 2.6.2.5.7.2.4. User Inputs

A summary of user inputs to the simulation is included for debugging purposes. This section can be turned on or off, and is set to 'off' by default because most sims will have a massive amount of input data.

Example File:



```

0.0,
0.0,
1.0],
'cmInertia': [[1,
0,
0],
[0,
1,
0],
[0,
0,
1]],
'inbToJoint': [0.0,
0.0,
0.0],
'inbToJointQuat': [0.0,
0.0,
0.0,
1.0],
'jointAxes': False,
'jointLimits': False,
'jointType': 'FULL6DOF_INERTIAL',
'mass': 133.21,
'negativeIntegralSense': True,
'prescribed': False}},
'Position': {'initPos_PCI': [-3900969.33545,
2509122.52581,
4698777.04841],
'initVel_PCI': [-1539.89613644,
-7234.69190295,
2465.47711765],
'name': 'ECI',
'refVertical': 'RADIAL'}},
'signals': {}},
'Earth': {'config': {'bodies': True},
'context': {},
'models': {'EarthAngle': {'flowOut': {'angle': [-3.4028234663852886e+38]},
'state': {'angle': 0.0}}},
'param': {'Target': {'eccentricity': 0.08176382867096284,
'flattening': 0.0033482672863778184,
'mass': 5.973698990946545e+24,
'name': 'Earth',
'radiusEquator': 6378140.000616,
'radiusPole': 6356784.283104,
'rotationAngle': 3.571677927,
'rotationRate': 7.2921151e-05}},
'signals': {'EarthAngle': [-3.4028234663852886e+38]}},
'TargetVehicle': {'config': {'basename': 'TargetBody'},
'context': {},
'gravity': {'config': {},
'context': {'body': 'TargetBody'},
'models': {'gravity': {'flowOut': {'ang_accel': [-3.4028234663852886e+38,
-3.4028234663852886e+38,
-3.4028234663852886e+38],
'lin_accel': [-3.4028234663852886e+38,
-3.4028234663852886e+38,
-3.4028234663852886e+38]}},
'param': {'BodyMass': 0.0,
'BodyName': '',
'EquatRadius': 0.0,
'G': 6.67259e-20,
'J2': 0.0,
'J3': 0.0,
'UseGradient': 0},
'scratch': {'Accel': [0.0,
0.0,
0.0,
0.0,
0.0,
0.0]},
'state': {'BodyPosition': [0.0,
0.0,
0.0],
'BodyVelocity': [0.0,
0.0,
0.0]}},
'param': {'Gravity': {'BodyName': '',
'G': 6.67259e-11,
'J2': 0.0010826260682438,
'J3': 0.0,
'Rgrav': 6378136.3,
'UseGradient': 0},
'Target': {'eccentricity': 0.08176382867096284,
'flattening': 0.0033482672863778184,
'mass': 5.973698990946545e+24,
'name': 'Earth',
'radiusEquator': 6378140.000616,
'radiusPole': 6356784.283104,
'rotationAngle': 3.571677927,
'rotationRate': 7.2921151e-05}},
'signals': {'AngAccel': [-3.4028234663852886e+38,
-3.4028234663852886e+38,
-3.4028234663852886e+38],
'LinAccel': [-3.4028234663852886e+38,
-3.4028234663852886e+38,
-3.4028234663852886e+38]}},

```

```

'models': {},
'param': {'Attitude': {'initAngleAttack': 0.0,
                        'initBank': 0.0,
                        'initOmega': [0, 0, 0],
                        'initSideslip': 0.0,
                        'name': 'LVLH'},
          'Bodies': {'TargetBody': {'bodyToCM': [0.0,
                                                  0.0,
                                                  0.0],
                                    'bodyToJoint': [0.0,
                                                    0.0,
                                                    0.0],
                                    'bodyToJointQuat': [0.0,
                                                       0.0,
                                                       0.0,
                                                       1.0],
                                    'cmInertia': [[1,
                                                  0,
                                                  0],
                                                  [0,
                                                  1,
                                                  0],
                                                  [0,
                                                  0,
                                                  1]],
                                    'inbToJoint': [0.0,
                                                    0.0,
                                                    0.0],
                                    'inbToJointQuat': [0.0,
                                                       0.0,
                                                       0.0,
                                                       1.0],
                                    'jointAxes': False,
                                    'jointLimits': False,
                                    'jointType': 'FULL6DOF_INERTIAL',
                                    'mass': 17.39,
                                    'negativeIntegralSense': True,
                                    'prescribed': False}},
          'Position': {'initPos_PCI': [3748996.1088,
                                       -3862101.2928000004,
                                       -4103766.24],
                       'initVel_PCI': [2298.6744984,
                                       6264.8592,
                                       -3785.616],
                       'name': 'ECI',
                       'refVertical': 'RADIAL'}},
'signals': {}},
'config': {},
'context': {},
'models': {'adv_dyn': {}},
'signals': {}

```

=====

ERROR / WARNING / PHASE MESSAGES

=====

| Sim Time | Category | Message   |
|----------|----------|---|
| 536.000  | PHASE    | Phase 1: 1st Phase, triggered with delta: -8.000000                                   |
| 566.000  | PHASE    | Phase 120: Target Sequence - 1st Phase, triggered with delta: 0.000000                |
| 616.000  | PHASE    | Phase 30: 2nd Phase, triggered with delta: 0.000000                                   |
| 616.000  | PHASE    | Phase 20: 3rd Phase, triggered with delta: 0.000000                                   |
| 616.000  | PHASE    | Phase 50: 4th Phase, triggered with delta: 0.000000                                   |
| 616.000  | PHASE    | Phase 10: 5th Phase, triggered with delta: 0.000000                                   |
| 636.000  | PHASE    | Phase 60: 6th Phase, triggered with delta: -5.000000                                  |
| 666.000  | PHASE    | Phase 130: Target Sequence - 2nd Phase, triggered with delta: 0.000000                |
| 666.000  | PHASE    | Phase 140: Target Sequence - 3rd Phase, triggered with delta: -30.000000              |
| 676.000  | PHASE    | Phase 145: Target Sequence - Optional Triggered Phase, triggered with delta: 0.000000 |
| 1116.000 | PHASE    | Phase 210: Floating Sequence - 1st Phase, triggered with delta: 0.000000              |
| 1146.000 | PHASE    | Phase 200: Floating Sequence - 2nd Phase, triggered with delta: 0.000000              |
| 1166.000 | PHASE    | Phase 220: Floating Sequence - 3rd Phase, triggered with delta: 0.000000              |
| 1266.000 | PHASE    | Phase 230: Floating Sequence - 4th Phase, triggered with delta: 0.000000              |
| 1266.000 | PHASE    | Phase 240: Floating Sequence - 5th Phase, triggered with delta: -50.000000            |
| 1266.000 | PHASE    | Phase 250: Floating Sequence - 6th Phase, triggered with delta: -100.000000           |
| 1326.000 | PHASE    | Phase 300: CID Sequence - 1st Phase, triggered with delta: 0.000000                   |
| 1515.954 | PHASE    | Phase 150: Chaser Sequence - 1st Phase, triggered with delta: 0.046054                |
| 1517.458 | PHASE    | Phase 100: 7th Phase Regula-Falsi, triggered with delta: 0.041521                     |
| 1517.458 | PHASE    | Phase 105: 8th Phase, triggered with delta: 0.790000                                  |
| 1520.950 | PHASE    | Phase 160: Chaser Sequence - 2nd Phase, triggered with delta: 0.049529                |
| 1536.712 | PHASE    | Phase 110: 9th Phase Regula, triggered with delta: 0.037500                           |
| 1566.000 | PHASE    | Phase 2: Last Phase, triggered with delta: 0.000000                                   |

### 2.6.2.5.8. MultiRun Integration in DshellCommon

Goal is to minimize the complexity of running distributed monte-carlo simulations. This includes making it easy for user specification of distributed machines on the network, and reducing the number of files and actions necessary to actually perform the monte-carlo.

The base MultiRun class only provides a framework to execute runs, but does not actually provide a means to execute the runs.

The IPythonMultiRun class is derived from the MultiRun class, but adds functionality necessary to run the simulation in two modes:

#### 1. Monte-Carlo

- Intended to distribute simulation runs across the network.

- Executes a large number of simulation runs where knowledge of run results is not critical to future runs.
- Currently only provides basic introspection to running engines. Future improvement could provide full introspection but need to be able to restart engines (since we currently can't re-create a simulation instance within one python interpreter). This would be useful to investigate why certain runs have failed or have become stalled.

## 2. (Future) Targeting/Optimization

- Used to execute a smaller number of runs specifically for targeting or optimization.
- Provides high degree of introspection with executing engines, i.e. input of controls, and output of desired result metrics.
- Basic functionality created, but needs more work.

### 2.6.2.5.8.1. Typical inputs

User specifies `--multirun=IPython` in command line args.

User specifies `multirun_config_file` (typically named `MonteCarlo.cfg`) in constructor.

`MonteCarlo.cfg` file specifies Knobs used in simulation, max number of jobs to be run, and the machine / # of nodes where runs are to be executed.

The `MultiRun` Class processes the knobs, instead of having the user perform this function.

Currently two distribution approaches are available via the `IPythonMultiRun` input for `launcher_class`:

- *Local* where the IPython engines are started on the local machine where the script is being run from.
- *SSH* where the IPython engines are started on remote machines specified in the `multirun` config file.

### 2.6.2.5.8.2. Monte-Carlo Execution

Upon construction, `IPythonMultiRun` creates an IPython cluster on the machines specified. The cluster creates an IPython controller on local host, along with all the IPython engines. The engines are created on the local host if the `launcher_config` option is set to *Local*, or on each of the specified machines if the *SSH* option is used. If a cluster is already executing for that set of monte-carlo runs, it is re-used.

A load balanced view is created for the IPython controller. All the monte-carlo keys are executed via the view. Basically, for a given set of engines across the cluster, monte-carlo runs are automatically dispatched to available engines. The load balanced view continues running these jobs on the engines until they are all completed.

### 2.6.2.5.8.3. Targeting/Optimization Execution

Not implemented yet.

### 2.6.2.5.8.4. Limitations

The current implementation using SSH launchers requires all controllers, engines and views to be hosted on a system that has common NFS access to all files used for execution.

Also, since ssh is used to launch all IPython engines, proper automatic ssh login keys must be available and the hosts specified must be in the ssh known hosts file.

### 2.6.2.5.8.5. Discussion

1. TBD ???Add multirun and run inputs to `script_args` in `argparser.py`???
2. Should we have tighter integration of `MultiRun` with `SimulationExecutive::` \* If multirun in `script.args`, have sim create an instance of `MultiRun`. \* Removes need to add configuration code to run script. \* Would need to add `MonteCarlo.cfg` file arg to `SimulationExecutive`.
3. Some legacy code in `DshellCommon.SimulationExecutive` (are these still used?)

TODO: Add to code (depricated), factor out or replace DCOW env vars.

```
# dwatch logging related code

def resultSpecs(self, specs=None)::
    "Result specifications for sim run" if specs: self._result_specs =
    specs return self._result_specs

# batch job related code @staticmethod def batchJob(): "Batch id number
if applicable" # the env var is defined in DshellEnv/DMonteCarlo.py
return os.getenv('DCOW_JOBID')

@staticmethod def resultsDir(): "Directory where overall results go"
return os.getenv('DCOW_RESULTS_DIR', os.getenv('PWD'))

@staticmethod def runDir(pdir=None):
```

PYTHON

## 2.6.2.5.9. Utils

### 2.6.2.5.9.1. Function and Class Documentation

#### 2.6.2.5.9.1.1. Introduction

The `Utils` modules provide various utility functions and classes in support of finite state machine operations, docstring, state initialization, etc.

#### 2.6.2.5.9.1.2. Utils Class API Documentation

##### 2.6.2.5.9.2. phasing.py

This module is similar to the FSM that controls events called phases. The types of phases currently implemented are Fixed, Free, Floating, and Optional phases. Phases are used to control simulation input. When a phase triggers based on user-specified criteria, user-specified commands are executed.

##### 2.6.2.5.9.3. cubicspline.py

Provides a `CubicSpline` class for interpolation.

##### 2.6.2.5.9.4. doctestutils.py

Utility functions for regtests

##### 2.6.2.5.9.5. stateInit.py

Provides the user the capability to initialize the states of vehicles in a particular simulation using information from the simulation assemblies.

##### 2.6.2.5.9.6. spiceUtils.py

Provides the user the capability to access certain SPICE functions directly from Python.

#### 2.6.2.5.10. Simulation Definition Block BNF Grammar Definition

A DshellCommon Simulation Definition Block is simply an Assembly Definition which describes the assemblies to be created for the top-level simulation assembly. In the rest of this page, we define the exact grammar for simulation definition blocks using Backus–Naur Form for defining grammars (typically used for computer languages):

```
<assembly-definition-block> ::=
  '{' <assembly-definition> [, <assembly-definition> ] }'

<assembly-definition> ::=
  <assembly-name> : <assembly-definition-dict>

<assembly-definition-dict> ::=
  '{' <required-assembly-items> [, <optional-assembly-item> ] }'

<required-assembly-definition-items> := <assembly-class>

<optional-assembly-item> ::= <params> | <context> | <subassemblies> |
  <signals> | <signal-ties> | <config-items>

<assembly-class> ::= 'class' : <assembly-class-name>

<params> ::= 'params' : ( <param-object> | <param-dict> )
  <param-dict> ::= '{' <param-spec> [, <param-spec> ] }'
  <param-spec> ::= <param-name> : <param-object>

<context> ::= 'context' : <context-spec>

<subassemblies> ::= 'assemblies' : '{' <assembly-definition-list> }'

<signals> ::= 'signals' : '{' <signal-definition> [, <signal-definition> ] }'

<signal-definition> ::= <signal-name> : <signal-definition-dict>

<signal-definition-dict> ::=
  '{' <signal-definition-item> [, <signal-definition-item> ] }'

<signal-definition-item> ::=
  <signal-type> | <signal-length> | <signal-nature> | <signal-comment>

<signal-type> ::= 'type' : ( 'double' | 'int' | 'bool' | 'string' )

<signal-length> ::= 'length' : <non-zero unsigned integer>

<signal-nature> ::=
  'physicalType' : ( 'POSITION' | 'LINEAR VELOCITY' | 'LINEAR ACCELERATION' |
    'ATTITUDE QUATERNION' | 'ANGULAR VELOCITY' |
    'ANGLE', 'UNKNOWN' )

<signal-comment> ::= 'comment' : <arbitrary quoted python string>

<signal-ties> ::= '{' <signal-ties-definition> [, <signal-ties-definition> ] }'

<signal-ties-definition> ::=
  <parent-signal-name> : '[' <signal-ties-items-list> ]'

<signal-ties-items-list> ::=
  <required-signal-ties-item-list> [, <optional-signal-ties-item-list> ]

<required-signal-ties-item-list> ::= <child-signal-name>

<optional-signal-ties-item-list> ::=
  <child-assembly-name> | <child-assembly-class> | <parent-slice>

<child-signal-name> ::= 'child-signal-name' : <signal-name>

<child-assembly-name> ::= 'child-assembly-name' : <assembly-name>

<child-assembly-class> ::= 'child-assembly-class' : <assembly-class-name>

<parent-slice> ::= 'parent-slice' : '[' <unsigned int> ',' <unsigned int> ]'
```

Notes:

- This grammar description uses regular Backus-Naur Form (BNF) notation with a few helpful syntax extensions from Extended Backus-Naur Form:
  - Optional items enclosed in square brackets (without quotes) can appear zero or more times. Note that brackets (and other characters) in quotes represent characters that would actually be in the python script (for instance delimiting a python list).
  - Exactly one item should appear from a list of items enclosed in parentheses and separated by vertical bars.
  - For example, the following specification can represents various expressions as shown on the right (separated by 'or'):

```
1 [, (2 | 3) ] ==> 1 or 1, 3 or 1, 2 or 1, 3, 2 or ...
```

- **<assembly-class-name>** : A quoted python string for the exact name of the assembly class.
- **<param-obj>** : A parameter type object/instance derived from BaseParam



- **<context-spec>** : The value of the 'context' input dictionary needed for the assembly being defined. Since this is assembly-specific, listing all the possibilities here was impractical. This should appear in exactly the form that the Assembly constructor requires.
- **<config-items>** : All items that are not <params>, <context>, <signals>, <signal-ties>, or <subassemblies>, are lumped together into a new 'config' dictionary and given as the 'config' dictionary for the assembly being created.
- **physicalType** : The value of 'physicalType' is primarily for documentation purposes: any string can be given. The only case where it affects program operation is for **physicalType='ATTITUDE QUATERNION'** (where it enables appropriate operation of quaternion signal initialization via the setToNominalValue() function call).

### 2.6.2.5.11. Sharing Frames in Models and Assemblies

#### 2.6.2.5.11.1. Introduction

This section will describe the approach of using Frame UUIDs to share frames between models and assemblies.

This approach was developed by the JSC MOD team.

##### 2.6.2.5.11.1.1. Assembly code

In the following description, we will deal with two assemblies:

- an assembly that is providing a frame, and
- an assembly that is using the frame in one of its models.

In some cases, an assembly may perform both roles.

Some functionality that simplifies dealing with frames has been aggregated in the FrameProvider class. In order to use it, the assembly should inherit from the FrameProvider class. First import FrameProvider near the top of the Python assembly code:

```
from DshellCommon.assemblies.FrameProvider import FrameProvider
```

In the class definition for the assembly class, add the additional parent class (this example is from TargetAssembly in DshellCommon):

```
class TargetAssembly(BaseAssembly, FrameProvider):
```

In the initialization method of the assembly class, right before the BaseAssembly class is initialized, initialize the FrameProvider class:

```
FrameProvider.__init__(self)
```

#### Assembly code for the frame provider

The basic idea of the frame provider assembly is that it will create the nodes and frames necessary to provide to other assemblies. For example, in the case of TargetAssembly, it would provide PCI and PCR frames.

In the assembly's addModels() method, use the FrameProvider method addFrame() function to add the frames that are created:

```
self.addFrame('MYFRAME', self._my_frame)
```

#### Assembly code for the frame user

In the assembly that needs the frame, the basic idea is that we get the frame from the frame provider assembly and then use it to initialize the UUID model parameter of one of the models created by the frame user assembly.

Since we need flexibility in naming a frame provider; normally the frame user assembly will define two required config variables in the initializer function:

```
self.requiredContextFields(['frameProviderAssembly', 'myFrameName'])
```

In the bindParams() method of the assembly code, we need to get a handle for the frame provider assembly assembly with some like this:

```
# Get the assembly from the simulation assemblies that will provide the trajectory frame.
fp_assembly = self.findAssembly(self.context()['frameProviderAssembly'])
```

Next we need to get a handle for the frame itself:

```
frame_name = self.context()['myFrameName']
try:
    self._my_frame = fp_assembly.getFrame(frame_name)
except AttributeError:
    raise AttributeError('Failed to obtain frame called %s' % frame_name)
```

Finally, we can get the UUID and set the necessary model parameter:

```
# Set required parameters
self.modelParamSet(self.name(), 'FRAME_UUID', self._my_frame.uuidInt())
```

#### 2.6.2.5.11.2. Model Code

In order to share frames via UUIDs, several additions are needed for the model code.

#### 2.6.2.5.11.2.1. Model .mdl code

The frame UUID is passed into a model using a special parameter. For example, a model needs a frame "X". First the parameter for this frame needs to be defined in the model definition (.mdl) file:

```
# class parameters
[params]
[[FRAMEX_UUID]]
Type = int
Length = -
Brief description = ""**Not for user input** - UUID for the frame.""
Long description = ""**Not for user input** - UUID for the frame.""
```

Note that the parameter is not set by the user in the normal config script. In general, bodies, frames, and models are created during the `addModels()` assembly calls. The assembly will set the frame parameter during `bindParams()`. Since `bindParams()` is called after all `addModels()` functions have been called, the code to define the UUID parameter can be confident of having real frames and UUIDs to deal with.

Don't forget to delete the auto directory and rebuild the model C++ auto code.

#### 2.6.2.5.11.2.2. Model .h code

Since the goal in the model code is to retrieve a frame from some other assembly, the user should set up a variable to hold the frame.

Either include "DFrame/Frame.h" or forward declare the frame class in the model's header file:

```
// Forward declarations
class DFrame::Frame;
```

Define a variable to hold the frame (normally in a private or protected section):

```
const DFrame::Frame *frame;
```

#### 2.6.2.5.11.2.3. Model .cc code

The model code should include the following Frame-related header files:

```
#include "DFrame/FrameContainer.h"
#include "DFrame/Frame.h"
```

In the model's constructor in the C++ code, the UUID parameter should be initialized:

```
params()->FRAME_UUID = 0;
```

The frame is retrieved in the model's `setup()` function. At this point, the initial model `bindParams()` call has been done and the frame UUID is known and available. The code would look something like this:

```
frame = &(simulation().frameContainer().getFrame(static_cast<UUIDTYPE>(params()->FRAME_UUID));
```

It would also be a good idea to add the following checks to make sure the frame object is valid:

```
// Check the dynamic cast converted the NdartsBaseObject to a Frame
if ( frame == 0L )
{
    ss << "FrameHelper::getFrame() --> Failed cast for frame UUID =" << id << " !\n";
    throw std::runtime_error(ss.str());
}

// Verify the frame is registered
if ( frame->isRegistered() != true )
{
    ss << "FrameHelper::getFrame() --> " << frame->name() << " not registered!\n";
    throw std::runtime_error(ss.str());
}
```

Once the frame is obtained, `Frame2Frame` objects can be created and saved as necessary.

## 3. DIntegrator

### 3.1. Background

#### 3.1.1. Reference & Source material

- [DIntegrator Doxygen docs](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/DIntegrator/html/) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/DIntegrator/html/)
- [Team talks](https://dlabportal.jpl.nasa.gov/resources/darts-lab-talks/) (https://dlabportal.jpl.nasa.gov/resources/darts-lab-talks/)
- [2019, 2020 course slides](https://dlabportal.jpl.nasa.gov/documentation/coursetutorial-links/) (https://dlabportal.jpl.nasa.gov/documentation/coursetutorial-links/)
- [Sphinx docs](https://dartslab.jpl.nasa.gov/dlabdocs/) (https://dartslab.jpl.nasa.gov/dlabdocs/)
- [Jupyter notebooks](https://dlabnotebooks.jpl.nasa.gov/hub/login) (https://dlabnotebooks.jpl.nasa.gov/hub/login)
- [DARTS Lab QA site](https://dartslab.jpl.nasa.gov/qa/) (https://dartslab.jpl.nasa.gov/qa/)

### 3.2. Design

### 3.3. Usage



TBD: Add Doxygen link for FuelManifol model above

### 3.4. Software

### 3.5. Raw documents

### 3.6. Sphinx documentation

DIntegrator is a set of classes that implement or interface to different integration schemes.

The base class is the DIntegrator class. This is an abstract class from which different instance integrator classes are derived. Current integration schemes are the DIntegrator\_euler, DIntegrator\_rk4, DIntegrator\_rkf4, the DIntegrator\_CVode and the DIntegrator\_noop.

- DIntegrator\_euler is a first order simple Euler integration scheme
- DIntegrator\_rk4 is the 4th order Runge-Kutta scheme.
- DIntegrator\_rkf4 is the Runge-Kutta-Fehlberg 4-5 scheme with the correction step.
- DIntegrator\_CVode is an interface to use the LLNL CVODES package for variable step integration.
- DIntegrator\_noop is a no-operation or dummy class that does not integrate, but provides the interface to create a dummy integrator

Each integration class has the following functions:

- **integrate**: This is the function call to solve the ODE. This function takes a function pointer for calculating the RHS of the equation  $y_{\dot{}} = \text{RHS}$ . In our case, this function call is the Darts++ solve equations of motion call. It also takes as arguments number of states, arrays for the state and the state derivative, the desired integration time step, minimum integration time step and the star time.
- **reset\_states**: This is a function call that sets the "reset\_requested" flag. This flag is used internally in the ref integrate function to regenerate the states. An example of this is when Darts++ states are changed run time due to change in the values of the states or a change in the number of states.
- There are several functions to set and get the different tolerances, i.e. the relative and absolute error tolerance, as well as for printing information about the integrator statistics and states.
- There are functions to create and delete instances of these objects.

#### 3.6.1. Reference



MISSING DOXYGEN LINK: DIntegrator module doxygen documentation



*Note*

For doxygen module documentation for DIntegrator; see: [DIntegrator Module<index.html>](#)

##### 3.6.1.1. DIntegrator

###### 3.6.1.1.1. Introduction

Add some intro material to DIntegrator ref doc?

###### 3.6.1.1.2. DIntegrator Class API Documentation



*Note*

For Doxygen documentation, please see: [DIntegrator](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/DIntegrator/html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/DIntegrator/html)

## 4. CCode

### 4.1. Background

#### 4.1.1. Reference & Source material

- [CCode Doxygen docs](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/CCode/html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/CCode/html)
- [Team talks](https://dlabportal.jpl.nasa.gov/resources/darts-lab-talks/) (https://dlabportal.jpl.nasa.gov/resources/darts-lab-talks/)
- [2019, 2020 course slides](https://dlabportal.jpl.nasa.gov/documentation/coursetutorial-links/) (https://dlabportal.jpl.nasa.gov/documentation/coursetutorial-links/)
- [Sphinx docs](https://dartslab.jpl.nasa.gov/dlabdocs/) (https://dartslab.jpl.nasa.gov/dlabdocs/)
- [Jupyter notebooks](https://dlabnotebooks.jpl.nasa.gov/hub/login) (https://dlabnotebooks.jpl.nasa.gov/hub/login)
- [DARTS Lab QA site](https://dartslab.jpl.nasa.gov/qa/) (https://dartslab.jpl.nasa.gov/qa/)

### 4.2. Design

### 4.3. Usage



TBD: CCode documentation TBD.

### 4.4. Software

### 4.5. Raw documents

#### 4.5.1. Sphinx documentation

##### 4.5.1.1. Introduction

The CCode module houses the CCODES package from LLNL as well as interface code to tie the integrators into the DIntegrator format for use with ROAMS. CCode is a variable step integrator that is adapted from the package CCODES. Documentation regarding the CCODES package can be found in the section Other CCode/CCODES Documentation.

There is a single class, **DIntegrator\_ccode**, defined in DIntegrator\_ccode.cc and DIntegrator\_ccode.h in this module, which exposes the CCODES package to the Darts/Dshell environment. This class sets up the CCODES integrator object, exposes functions to set the tolerances, weights and other settings, as well as to print debugging information. The DIntegrator\_ccode class is derived from the generic DIntegrator base class which manages the different integrators used by Darts/Dshell.

Here is an extract from the CCODES manual:

```
CCODES is part of a software family called sundials: SUite of Nonlinear and
Differential/Algebraic equation Solvers. This suite consists of ccode, kinsol,
and ida, and variants of these. CCODES is a solver for stiff and nonstiff
initial value problems for systems of ordinary differential equation
(ODEs). In addition to solving stiff and nonstiff ODE systems, CCODES has
sensitivity analysis capabilities, using either the forward or the adjoint
methods.
```

Here is an extract from the title block of the ccodes.c source code:

```
File      : ccodes.c
Programmers : Scott D. Cohen, Alan C. Hindmarsh, Radu Serban
            and Dan Shumaker @ LLNL
Version of : 27 June 2002

-----
Copyright (c) 2002, The Regents of the University of California
Produced at the Lawrence Livermore National Laboratory
All rights reserved
For details, see sundials/ccodes/LICENSE
-----

This is the implementation file for the main CCODES integrator
with sensitivity analysis capabilities.
```

##### 4.5.1.2. Other CCode/CCODES Documentation

See [https://computing.llnl.gov/sites/default/files/cvs\\_guide.pdf](https://computing.llnl.gov/sites/default/files/cvs_guide.pdf) for the latest CCODES documentation. A copy of this document is in the module 'doc' directory.

See <https://computing.llnl.gov/projects/sundials> for more information on SUNDIALS, the "SUite of Nonlinear and Differentail/Algebraic equation Solvers" from Lawrence Livermore Laboratories. CCODES is part of that library.

##### 4.5.1.3. CCode Doxygen Documentation



#### Note

For doxygen module documentation for CCode, see: [CCode Module](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/CCode/html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/CCode/html)

## 5. DshellEnv

### 5.1. Background

#### 5.1.1. Reference & Source material

- [DshellEnv Doxygen docs](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/DshellEnv/html/) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/DshellEnv/html/)
- [Team talks](https://dlabportal.jpl.nasa.gov/resources/darts-lab-talks/) (https://dlabportal.jpl.nasa.gov/resources/darts-lab-talks/)
- [2019, 2020 course slides](https://dlabportal.jpl.nasa.gov/documentation/coursetutorial-links/) (https://dlabportal.jpl.nasa.gov/documentation/coursetutorial-links/)
- [Sphinx docs](https://dartslab.jpl.nasa.gov/dlabdocs/) (https://dartslab.jpl.nasa.gov/dlabdocs/)
- [Jupyter notebooks](https://dlabnotebooks.jpl.nasa.gov/hub/login) (https://dlabnotebooks.jpl.nasa.gov/hub/login)
- [DARTS Lab QA site](https://dartslab.jpl.nasa.gov/qa/) (https://dartslab.jpl.nasa.gov/qa/)

### 5.2. Design

#### 5.2.1. Dclick option handling

##### 5.2.1.1. DClick option handling requirements

- Should be able load options from a config file
  - Should have support for multiple config file types, e.g., YAML, INI, etc. The file type should be inferred from its extension.
- Should be able to save current set of options to a config file
- Option handling should support all combinations environment variables, config files, command line option values and built in defaults
  - command line values take precedence (per run)
  - which take precedence over environment variables (per user)
  - which take precedence over over config files (project use cases)
  - which take precedence over built-in defaults (software)
- Should be able to support sub-commands (eg. for adding a rover group of options, adding a topo)
- Should be able to specify same sub-command multiple times (to add multiple rovers or topos)
- Should be able to specify multiple different sub-commands (to add rovers and topos)
- Each subcommand should have its own set of options (rover options are different from ones for topos)
- Should be able to tailor options handling to specific context choices (eg. allowed config values for a rover depend on the rover class chosen,)

##### 5.2.1.2. Dclick validation

Dclick uses the `pydantic` module to validate input from config files.

The validator is created by parsing through the `Dclick.cli` group's commands and their associated options. `pydantic` allows nesting validators, so a validator is created for each command. For each command, the associated option's information is extracted, and used to dynamically create a `pydantic` validator via the `create_model` function. The validator checks the type of each option, and adds extra validation functions if needed. For example, if the type is a `FloatRange` or `IntRange`, validation functions are added that check the input is within the associated min/max values. `pydantic` supports complex types, e.g., `Tuple[Tuple[str, float], int]`, but min/max values are not checked when `FloatRange` or `IntRange` is nested. For example, if the type was `Tuple[FloatRange, IntRange]` the validator would check that the incoming type was `Tuple[float, int]`, but it would not check the min/max values of the incoming float and int. It should be noted here that the functions for checking min/max are not built into `pydantic`, but rather are added via custom functions that use the validation decorator. If such sophisticated checking is required in the future, we can create custom functions for this. However, since there is not a use case for it currently, it has not yet been added.

In addition, `pydantic` validators are used to prep values for output. For example, in `Dclick.OutputValidator`, variables of type `DclickPath` have their values compressed, so rather than seeing the full file path for `YAM_ROOT`, one will simply see `$YAM_ROOT`, e.g., `/home/leake/Documents/roamsPkg/etc/` becomes `$YAM_ROOT/etc`.

##### 5.2.1.3. Dclick modes

Oftentimes there are cases where one is interested in displaying different sets of commands/options: let's call these different sets of options "modes". For example, consider a development vs. delivery mode where a full set of options/commands is wanted/needed during development, but a smaller set of options/commands is desired for delivery. This scenario can be implemented in `Dclick` with the use of the `mode` keyword in both `Dclick.option` and `Dclick.cli.command`.

The following script shows an example of a mode option used for development vs. delivery.

```
#!/usr/bin/env python

from Dutils import Dclick
import click

@Dclick.cli.command()
@click.pass_context
@click.option("--opt1", type=float, default=1, show_default=True, help="")
def command_all(ctx, **kwargs):
    """
    This command is used by the development and delivery audiences and should appear the same for both.
    """
    Dclick.add_options("command-all", Dclick.cli.commands["command-all"], kwargs)

@Dclick.cli.command()
@click.pass_context
@Dclick.option("--opt1",
               type=float,
               default=1,
               show_default=True,
               help="This is a help message intended to be used by the delivery audience.",
               mode="delivery",
               )
@Dclick.option("--opt1",
               type=float,
               default=1,
               show_default=True,
               help="This is a help message intended to be used by the developer audience.",
               mode="developer",
               )
def different_help(ctx, **kwargs):
    """
    Some options.
    """
    Dclick.add_options("different-help", Dclick.cli.commands["different-help"], kwargs)

@Dclick.cli.command(mode="developer")
@click.pass_context
@click.option("--opt1", type=float, default=1, show_default=True, help="Option 1")
def development_only(ctx, **kwargs):
    """
    Command that appears only when in development mode.
    """
    Dclick.add_options("development-only", Dclick.cli.commands["development-only"], kwargs)

Dclick.set_default_mode("delivery")

try:
    cfgobj, ctxobj = Dclick.cli(standalone_mode=False)
    print(cfgobj)
except:
    # help was called so exit
    import sys
    sys.exit(1)
```

If the "developer" mode is chosen, then the "development-only" command is added, and the help message of the "opt1" option of the "different-help" command is set accordingly. If the "delivery" mode is chosen, then the "development-only" command is not added, and the help message of the "opt1" option of the "different-help" command is set accordingly.

Suppose we have this in a script called test. Then, here is the help message when we run everything with default settings, i.e., in delivery mode using `srun ./test -h`. Note the addition of the option `--mode`:

```
Usage: test [OPTIONS] COMMAND1 [ARGS]... [COMMAND2 [ARGS]...]...

  A command line interface tool for Dshell simulations.

Options:
  --in-cfg TEXT           Full or partial input config file.
  --full-out-cfg PATH    Full output config file.
  --delta-out-cfg PATH   Delta output config file.
  --write-default-cfg    Flag to write a default config file
                        (config.ini) using the master configspec.
  --write-configspec     Flag to output a default master configspec file
                        (master_configspec.ini).
  --delta-options        Flag to output command line options
                        corresponding the current options.
  --file-exec PATH       A file to execute after simulation is locked
                        and ready to go.
  --version              Show the version and exit.
  --mode [developer|delivery] Dclick mode. [default: delivery]
  -h, --help            Show this message and exit.

Commands:
  command-all  This command is used by the development and delivery...
  different-help  Some options.
```

Here is the help message when we run using development mode, i.e., using `srun ./test --mode developer -h`,

```
Usage: test [OPTIONS] COMMAND1 [ARGS]... [COMMAND2 [ARGS]...]...

A command line interface tool for Dshell simulations.

Options:
  --in-cfg TEXT           Full or partial input config file.
  --full-out-cfg PATH     Full output config file.
  --delta-out-cfg PATH    Delta output config file.
  --write-default-cfg     Flag to write a default config file
                          (config.ini) using the master configspec.
  --write-configspec      Flag to output a default master configspec file
                          (master_configspec.ini).
  --delta-options         Flag to output command line options
                          corresponding the current options.
  --file-exec PATH        A file to execute after simulation is locked
                          and ready to go.
  --version               Show the version and exit.
  --mode [developer|delivery] Dclick mode. [default: delivery]
  -h, --help              Show this message and exit.

Commands:
  command-all           This command is used by the development and delivery...
  development-only       Command that appears only when in development mode.
  different-help         Some options.
```

As the help message shows, enabling developer mode adds in the "development-only" command. Moreover, if we look at the help messages for the "opt1" option of the "different-help" command, we can see the help message has changed as desired. For delivery mode, `srun ./test different-help -h`,

```
Usage: test different-help [OPTIONS]

Some options.

Options:
  --opt1 FLOAT This is a help message intended to be used by the delivery
               audience. [default: 1.0]
  -h, --help   Show this message and exit.
```

and for developer mode, `srun ./test --mode developer different-help -h`,

```
Usage: test different-help [OPTIONS]

Some options.

Options:
  --opt1 FLOAT This is a help message intended to be used by the developer
               audience. [default: 1.0]
  -h, --help   Show this message and exit.
```

Note that an arbitrary number of modes are supported. For example, we may have "delivery", "delivery-advanced", and "developer". If we want to activate an option/command for more than one mode, we can set the `mode` keyword to be a tuple of the valid modes:

```
@Dclick.cli.command(mode=("delivery-advanced", "developer"))
@click.pass_context
@click.option("--opt1", type=float, default=1, show_default=True, help="Option 1")
def advanced_and_dev(ctx, **kwargs):
    """
    Command that appears only when in delivery-advanced or developer mode.
    """
    Dclick.add_options("advanced-and-dev", Dclick.cli.commands["advanced-and-dev"], kwargs)
```

PYTHON

#### 5.2.1.4. Dclick version

Each Dclick application comes with a version number. By default, this version number is set to 1.0. The version number can be updated using

```
Dclick.update_version(versionNumber)
```

PYTHON

where `versionNumber` is new version number. Behind the scenes, Dclick uses a hidden command to ensure that the version number is printed to each output config file, which will look like this

```
[version]
version = 1.0
```

However, this command will not be visible or directly modifiable to users: it is only indirectly modifiable via the `update_version` method shown above. If an input config file has a version number that is different than the Dclick application's, this will trigger an error.

The top-level `--version` option can be used on the command line to display the version and exit, e.g., `srun ./my-dclick-app --version`.

#### 5.2.1.5. Filtering dictionaries with Dclick

Dclick includes a `filter_dict` function, which can be used to filter a nested dictionary. The filters are functions of type `Callable[[Any, Any], bool]`, which take in the a key-value pair (the `[Any, Any]` inputs) from the dictionary and return a `bool`. If the return value is `True`, then the key-value pair should be kept. If the return value is `False`, then it should be removed. The filter functions are passed to the `filter_dict` in a dictionary whose keys are integers, which specify the layer of the dictionary that the corresponding filter functions should be used on,

and the values are the filter functions themselves. The values can be either a single filter function, or a list of filter functions. In addition, a keyword argument, `in_place`, specifies whether `filter_dict` should filter the dictionary in-place, or whether it should modify and return a copy of the input dictionary. For more details on this function, see the corresponding usage section.

#### 5.2.1.6. `modelParams.*` commands



**TBD:** Fill in this section once `modelParams.*` API has been fleshed out.

Dclick has a helper function called `getModelParamsDict`, which can be used to get all `modelParams`. **command information from the Dclick dictionary, or all `modelParams`.** command information that regex matches a given ID. For more information, see the corresponding usage section.

#### 5.2.1.7. Tab-based completion

##### 5.2.1.7.1. Background on shell tab-completion

Various shells, e.g., `bash`, `zsh`, etc., have tab-based completion systems built in. Each completion system is slightly different, but they do have some similarities. One important similarity, that will be leveraged here, is the ability to register a completion function that completes the tab-completion suggestions for a given command. In general, we can use this concept to generate shell functions that will provide the tab-completion suggestions for our Dclick applications.

##### 5.2.1.7.2. Python `click` tab-completion

The `click` Python package has tab-completion built in: see [here](https://click.palletsprojects.com/en/8.1.x/shell-completion/) (<https://click.palletsprojects.com/en/8.1.x/shell-completion/>). The scheme provided `click` can be used to generate shell scripts with the tab-completion functions and registration calls. For example, in `bash` for a `click` application `foo-bar` one can use

```
_FOO_BAR_COMPLETE=bash_source foo-bar > foo-bar-complete.bash BASH
```

to create a script called `foo-bar-complete.sh` that registers a tab-completion function for the `foo-bar` command. In DARTS, we have a make target called `dclick_complete_files-module` that runs these commands for all Dclick application scripts in the `DCLICK_SOURCE` variable, and adds them to the `<YAM_ROOT>/etc/Dclick_complete` folder.

##### 5.2.1.7.3. DARTS tab-completion

Once the tab-completion functions for the Dclick applications have been created, all that is left to do is source them and call them. To do this, the `srun` tab-completion function is used. In the `DshellEnv`, there are two scripts, `srun-complete.bash` and `srun-complete.zsh`, that contain the tab-completion for the `srun` command in `bash` and `zsh` respectively. The `srun` tab-completion will source the appropriate file for the incoming command, and then call the associated tab-completion function. Source the tab-completion code is custom code written by us, and calling the associated tab-completion function follows from the functions used by the `xargs` command to accomplish a similar feat.

The `srun` tab-completion functions are bit complicated, since they also need to set up system environment variables, e.g., `PATH`, `LD_LIBRARY_PATH`, etc. For more details, please see the code for `srun` tab-completion functions in `DshellEnv`.

#### 5.2.1.8. Exporting Dclick application help to Markdown

Oftentimes, it is convenient to include Dclick application help strings in the appendix of customer documentation. This provides a place for the customer to easily reference all of the Dclick application options. The `Dclick` module comes with a `exportHelpToMarkdown` function that can be used for this purpose.

This function works by modifying `sys.argv` (`sys.argv` is modified within a context that ensures its original state is returned once finished) and running the Dclick application with `--help` specified for the top-level and for all commands of the application. During this time, `stdout` is redirected via a context manager to capture the help output in strings or files. This function also contains the ability to optionally write the command-level help strings into collapsible sections using markdown extensions.

## 5.3. Usage

### 5.3.1. Dclick Usage

#### 5.3.1.1. Multi-command defaults

Sometimes, we want to have a default instance of a multicommand invoked if no instances have been invoked by the user. One can think of this as a multi-command default: a default instance of the multi-command that gets used if no other instances have been specified. If a user wants this default instance, they can create it using the `Dclick.add_multi_cmd_default` function. This function accepts the following:

- `cmd` (*Dclick multi-command*) - The Dclick multi-command to add the default to.
- `name` (*str*) - This argument specifies the name to give the default instance of the multi-command.
- `kwargs` - Any extra keyword arguments are used to modify the defaults of the default multi-command instance.

Reg tests that utilize this functionality can be found in `DshellEnv/test/test_Dclick/test_multi_cmd_default`.

#### 5.3.1.2. Filtering dictionaries

Dclick has the ability to filter dictionaries via the `filter_dict` function. This function accepts the following inputs: `* d_orig` (*dict*) - Dictionary to be filtered. This can be a nested dictionary. `* filters` (*Dict[int, Union[Callable]]*) - A dictionary of filter functions. The key, an integer, specifies the layer (level of nesting) of the dictionary that the filter will be applied to. Each layer can have one or more filter functions. The filter functions, of type `Callable[[Any, Any], bool]`, take in the a key-value pair from the dictionary and return a `bool`. If the return value is `True`, then the key-value pair should be kept. If the return value is `False`, then it should be removed. `* in_place` (*bool, optional*) - This keyword argument specifies whether the dictionary will be modified in place or not. (default = `False`)

Reg tests that utilize this functionality (indirectly) can be found in `DshellEnv/test/test_Dclick/test_dict_filter`.

#### 5.3.1.3. `getModelParamsDict`



Dclick has a helper function called `getModelParamsDict` that utilizes the dictionary filtering functions to extract all of the `modelParams`. **command information for all IDs that regex match a user-specified string; if no string is supplied, then it returns all** `modelParams`. command information. The function accepts the following inputs: `*idFilter (str, optional)` - This string will be used as a filter, by regex matching it, on level 1 of the dictionary. For example, if one used "roverA.\*", then the function will only return IDs that start with roverA. (default = None) `*d (dict, optional)` - This is the dictionary that will be filtered. If None, then the output of `Dclick.cli` is used. (default = None)

Reg tests that utilize this functionality can be found in `DshellEnv/test/test_Dclick/test_dict_filter`.

#### 5.3.1.4. Tab-based autocompletion

If you are on a DLAB machine, use the following steps to setup tab-completion for your `Dclick` application; note, this will only work if you are using either the `bash` or `zsh` shells. 1. Add the scripts you care about to the `DCLICK_SOURCE` variable in the associated module's Makefile 2. Run `make dclick_complete_files-module`. You should now see an associated completion file in the `YAM_ROOT/etc/Dclick_complete` folder.

If you are on a personal machine, you will also need to source the associated `srun-complete` file (either `srun-complete.bash` or `srun-complete.zsh`) from the `DshellEnv` module.

### 5.3.2. Typing

#### 5.3.2.1. Quick links

Follow the links if you want to...

- learn the basics about typing in python (A, B) and using `Dutils.typing` (C).
- enforce type checking in a function or code you are writing (A).
- find an appropriate type hint for a specific input (A).
- write advanced (DARTS-flavored) checks (A, B).
- learn about built-in `FieldCheck` (A).
- learn about built-in advanced (DARTS-flavored) types (A).

#### 5.3.2.2. Basics of python typing

Typing (or type hinting) is a > 3.5 python feature that allows developers to **annotate** their code with the type that a variable / attribute / function argument **should** be at runtime. This does not have any effects at runtime: a developer may annotate a value as a type yet be a completely different type at runtime.

The purpose of type hinting thus is not safety, but documenting. When we annotate the types in a function signature, we make its usage clearer:

```
def my_list_generator(name: str, reps: int) -> List[str]:
    return [name] * reps
```

PYTHON

As we can see in the snippet above, we are declaring that `name` should be a string and `reps` an integer; and that the function will return a list of strings.

Similarly, we can annotate classes so specify the types of its attributes:

```
class Dog:
    name: str
    age: int
    ...
```

PYTHON

In this way, users of the class `Dog` will now that they can expect the attributes `name` and `age` to be of the types `str` and `int`.

Finally, we can annotate simple variables. This is useful when a variable is used in a long piece of code where it is easy to lose control on what each variable represented:

```
def print_names():
    names: List[str] = []
    ...
```

PYTHON

In the case above, we see that we are specifying that `names` is a list of strings, which might not be obvious from its initialization value (an empty list).

See more on typing on the python docs: <https://docs.python.org/3/library/typing.html>.

#### 5.3.2.3. Dutils typing

In the above snippets you may have noticed that type annotations can be made both with regular types (`str`, `int` ...) and other special notation (for example, `List[str]`). This special notation can be used to represent complex typing patterns. For example, the `List[str]` annotation not only says that the type must be a list, but also that all its elements must be strings.

In order to use this special syntax, developers must import the relevant annotations from the `typing` module:

```
from typing import List

def test(foo: List[int]):
    ...
```

PYTHON

However, the `typing` module does not always remain consistent across versions. For example, python 3.9 saw the deprecation (and eventual removal) of the `List[...]` notation in favor of `list[...]`. If we were to write DARTS code with `List[...]` (which is necessary to support python < 3.9), we would find that when `List` gets removed we would need to fix all our type annotations. To prevent this, we have the `Dutils.typing` module, which acts as a version-agnostic drop-in replacement for `typing`:

```
from Dutils.typing import List

def test(foo: List[int]):
    ...
```

Additionally, `Dutils.typing` also backports all typing functionality implemented in newer versions of python. For example, `Annotated` was included in `typing` in 3.9, but it is available in `Dutils.typing` for earlier versions.

#### ► Using pyright (VSCode, pylance)

##### 5.3.2.4. Typing examples

There are many useful `typing` constructs in python. In this section, we show an example of a function that takes in arguments of many different types, and we explain what values would be acceptable for each type:

```
from Dutils.typing import (
    Optional,
    Union,
    Literal,
    List,
    Tuple,
    Set,
    Dict,
    Sequence,
    Iterable,
    Mapping,
    Any,
)

class MyType:
    ...

ListOfAgencies = List[Literal["NASA", "ESA", "JAXA", "Roscosmos"]]

def my_complex_function(
    a_type: MyType
    b_union: Union[str, bool]
    c_optional: Optional[str]
    d_literal: Literal["foo", True, 42]
    e_list: List[str]
    f_tuple: Tuple[int, ...]
    g_tuple_fixed: Tuple[int, str, float]
    h_set: Set[str]
    i_dict: Dict[str, int]
    j_sequence: Sequence[float]
    k_iterable: Iterable[int]
    l_mapping: Mapping[str, int]
    m_combinations: Union[List[Union[str, float]], Optional[Set[int]]]
    n_alias: ListOfAgencies
    o_any: Any
):
    ...
```

Brief description on what is acceptable for each of the fields:

- **a\_type**: when you specify a python class (including built-ins like `str` or `int`) the value must be an instance of the type. This takes subclassing into account: if `Child` is a subclass of `Parent`, then any `Child` object will be acceptable in a `Parent`-type field. The opposite is not true. Note that `int` are always acceptable where `float` are accepted.
- **b\_union**: the notation `Union[X, Y]` means that a value is valid if it is either a `X` or a `Y`. An arbitrary number of types is supported: `Union[X, Y, Z]` is also valid. For `Union[str, bool]`, both `"bar"` and `False` will be valid options, for example.
- **c\_optional**: `Optional[X]` means that the value can be either `X` or `None`.
- **d\_literal**: `Literal[X, Y]` means that the value can be exactly `X` or `Y`. Instead of types, `Literal` takes instances of `str`, `int`, `bool` or enum and uses the equality operator `==` to determine whether a value is acceptable. Much like `Union`, `Literal` takes an arbitrary number of arguments. For `Literal["foo", True, 42]`, for example, we can only use `"foo"`, `True`, or `42`, but not any other value.
- **e\_list**: `List[X]` means that the value must be a list and that every element of the list must of type `X`. For `List[str]`, `["foo", "bar"]` would be accepted, but not `("foo", "bar")` or `["foo", 42]`.
- **f\_tuple**: `Tuple[X, ...]` acts like `List[X]`, except that the value must be tuple instead of a list. For `Tuple[int, ...]`, `(1, 42)` would be accepted, but not `[1, 42]`.
- **g\_tuple\_fixed**: `Tuple[X, Y, Z]` means that the input must be a tuple with exactly three values, and that the first one must be of type `X`, the second one of type `Y`, etc. An arbitrary number of required inputs is allowed. For `Tuple[int, str, float]`, the value `(32, "test", 4.3)` would be valid.
- **h\_set**: `Set[X]` requires that the input be a python set with only types `X`. For `Set[str]`, the value `{"foo", "bar"}` would be permitted.
- **i\_dict**: `Dict[X, Y]` requires that the input be a dictionary, that every key in the dictionary be of type `X`, and that every value be of type `Y`. For `Dict[str, int]`, the value `{"foo": 1, "bar": 2}` would be valid, but not `{"foo": 1.5, "bar": 2}` or `{"foo": 1, 42: 2}`.
- **j\_sequence**: `Sequence[X]` means that any value that inherits from the abstract class `Sequence` is valid as long as all its members are of type `X`. Classes that inherit from `Sequence` support accessing its elements as an array. Most notably, this includes both tuples and lists. For `Sequence[float]`, both `(1, 2.3, 4)` and `[1, 2.3, 4]` would be acceptable, but not `{1, 2.3, 4}` as one cannot access the elements of a set.
- **k\_iterable**: `Iterable[X]` allows any value that is iterable and whose values are all of type `X`. This means that they support something like: `for i in my_iterable: ...`. This supports lists and tuples, but also sets and dictionaries, as they are all iterable.

- **`l_mapping`**: `Mapping[X, Y]` requires that the input be any object that inherits from the abstract collection class `Mapping`, that every key (`my_mapping.keys()`) be of type `X`, and that every value (`my_mapping.values()`) be of type `Y`. Of course, this allows regular dictionaries, but also other dict-like classes such as `OrderedDict` or even `InputDict`.
- **`m_combinations`**: arbitrary combinations of the previously mentioned types are possible. In the provided example (`Union[List[Union[str, float]], Optional[Set[int]]]`), the value may be of two types: either a list of strings or float, or an "optional" set of integers. Since the set of integers is "optional", it means that `None` is also an accepted value.
- **`n_alias`**: given that type definitions can be arbitrarily complex (as we saw in `m_combinations`), sometimes it is useful to create a type alias. In `n_alias` we use `ListOfAgencies`, which we previously defined to be `List[Literal["NASA", "ESA", "JAXA", "Roscosmos"]]`. Using the alias or the value has the same effect. Aliases are useful because they improve reusability of complex types and improve clarity.
- **`o_any`**: finally, the special word `Any` means that any value is accepted. Users should avoid using this type as an easy and lazy catch-all, and restrict its usage to instances where truly any value is acceptable.

### 5.3.2.5. Type checkers

While type annotations do not have any effects at runtime, there are certain 3rd party tools that can process the information in type annotations and notify users when they are violating these annotations.

For example, by using Intellisense in VSCode (install the Python extension pack in VSCode), and changing "Type checking mode" to "basic", we will be notified when we are using a function, class or variable incorrectly:

(IMAGES)

These types of hints allow you to catch errors before even running python and encourage you to consider all edge cases.

### 5.3.2.6. Runtime type checking

While python typing was designed to have no impact at runtime, sometimes we would prefer to enforce the types we annotated in order to increase the safety of our program.

Note that "there ain't no such thing as a free lunch": when you perform runtime type checking you are paying a cost in performance. Type checking is better suited for public functions that users will interact with during set-up, while using type checking on private functions that must be run many times is probably a bad idea.

Currently, we can enforce type annotations in three ways: with `Dutils.typing.utils.type_check`, `Dutils.typing.utils.type_check_fn`, and `DshellCommon.input_dict.input_dict.InputDict`.

These docs will cover the first two ways, while `InputDict` has a separate doc entry.

#### 5.3.2.6.1. `Dutils.typing.utils.type_check`

`type_check` is a function that takes in two arguments, a value and its expected type. This function will raise a `TypeError` if the value does not correspond to the type. Otherwise, it will return the input value unchanged:

```
from Dutils.typing.utils import type_check

value = "test"
value = type_check(value, str) # ok

value = type_check(value, int) # raises TypeError
```

The `type_check` function also supports the greater part of the typing constructs (all the examples shown in Typing examples can be type checked at runtime):

```
from Dutils.typing import List
from Dutils.typing.utils import type_check

value = ["foo", "bar"]
value = type_check(value, List[str]) # ok

value = type_check(value, List[int]) # raises TypeError
```

This function acts similar to `typing.cast`: it will narrow down the type of the given value to the given type (or it will raise an error in the attempt).

`type_check` can also return a boolean instead of raising an exception if `raises=False` is passed to the function:

```
from Dutils.typing import List
from Dutils.typing.utils import type_check

value = ["foo", "bar"]

if type_check(value, List[str], raises=False):
    print(", ".join(value))
else:
    print(", ".join( str(v) for v in value ))
```

For regular types (`int`, `str`, `MyObject`), `isinstance` is preferred as it is faster. For generics (`List[int]`, `Dict[str, int]`, ...) is required, as these are not supported by `isinstance`.

#### 5.3.2.6.2. `Dutils.typing.utils.type_check_fn`

`type_check_fn` is a function decorator that automatically type checks the inputs to a function according to the type hints in its signature:

```

from Dutils.typing import List
from Dutils.typing.utils import type_check_fn

@type_check_fn
def my_fun(foo: int, bar: List[str], *args: str, **kargs: float):
    ...

my_fun(1, ["J","P","L"], "test", "test2", key_test_1=1, key_test_2=2) # ok

my_fun("invalid", ["J","P","L"], "test", "test2", key_test_1=1, key_test_2=2) # TypeError because `foo` is not `int`

my_fun(1, ["J","P","L"], "test", 42, key_test_1=1, key_test_2=2) # TypeError because the second variadic argument is not `str`

my_fun(1, ["J","P","L"], "test", "test2", key_test_1=1, key_test_2="2") # TypeError because the keyword argument `key_test_2` is not `float`

```

The above function definition is equivalent to:

```

from Dutils.typing import List
from Dutils.typing.utils import type_check

def my_fun(foo: int, bar: List[str], *args: str, **kargs: float):

    type_check(foo, int)
    type_check(bar, List[str])
    for arg in args:
        type_check(arg, str)
    for karg in kargs.values():
        type_check(karg, float)

    ...

```

Note that default values are not type checked. To exclude certain inputs from type checking, specify `exclude` in the decorator:

```

from Dutils.typing import List
from Dutils.typing.utils import type_check_fn

@type_check_fn(exclude={"bar", "args"})
def my_fun(foo: int, bar: List[str], *args: str, **kargs: float):
    ...

my_fun(1, "INVALID", 42, keyword=4.0) # ok

```

### 5.3.2.6.3. Extending runtime checking: FieldCheck

Sometimes, we have constraints for inputs that go beyond their type. For example, we might need that an input be a positive integer. In this case, type checking that the input is an integer is a necessary step, but we also need to check that the value is greater than zero.

In this case, we need to "extend" a type to include additional checks. We use the `Dutils.typing.Annotated` and `Dutils.typing.field_check.FieldCheck` classes to do this:

```

from Dutils.typing.field_check import FieldCheck
from Dutils.typing import Annotated
from Dutils.typing.utils import type_check_fn

class PositiveCheck(FieldCheck):

    def check(self, value):
        return 0 < value

@type_check_fn
def print_age(age: Annotated[int, PositiveCheck()]):
    print(age)

print_age(42) # ok

print_age(-1)
# TypeError: Incorrect type for argument 'age' in function 'print_age': Value -1 failed check PositiveCheck

```

In the previous snippet, we are creating a subclass of `FieldCheck` and defining the method `check` to return True only if the value is acceptable. Then, we are annotating the regular "core" type `int` with an instance of `PositiveCheck`.

In essence, by using `Annotated` with `FieldCheck`, we are telling the type checker to not only ensure the value is of the correct "core" type, but also that it should meet all the annotating `FieldCheck` to be considered a valid type. We extend the definition of a type to include custom checks through `FieldCheck`. This is particularly powerful when we combine it with type alias, as it allows us to easily create custom types and hide the implementation details:

```

PositiveInt = Annotated[int, PositiveCheck()]

@type_check_fn
def print_age(age: PositiveInt):
    print(age)

```

The function definition in the above snippet is equivalent to the previous definition, but it is cleaner and more readable.

We might also perform several improvements to the `PositiveCheck` class by defining the `fail_message` method and thus specifying a clearer error message:

```

class PositiveCheck(FieldCheck):

    def check(self, value):
        return 0 < value

    def fail_message(self, value):
        return f"{value} is not positive"

PositiveInt = Annotated[int, PositiveCheck()]

@type_check_fn
def print_age(age: PositiveInt):
    print(age)

print_age(-1)
# TypeError: Incorrect type for argument 'age' in function 'print_age': -1 is not positive

```

We could also make `PositiveCheck` more reusable by allowing users to specify the minimum value, instead of hardcoding it to 0:

```

from Dutils.typing.field_check import FieldCheck
from Dutils.typing import Annotated
from Dutils.typing.utils import type_check_fn

class GreaterThanCheck(FieldCheck):

    def __init__(self, minimum):
        self.minimum = minimum

    def check(self, value):
        return self.minimum < value

    def fail_message(self, value):
        return f"{value} is not greater than {self.minimum}"

PositiveInt = Annotated[int, GreaterThanCheck(0)]

@type_check_fn
def print_age(age: PositiveInt):
    print(age)

@type_check_fn
def print_adult_age(age: Annotated[int, GreaterThanCheck(17)]):
    print(age)

print_age(-1)
# TypeError: Incorrect type for argument 'age' in function 'print_age': Value -1 failed check PositiveCheck

print_adult_age(16)
# TypeError: Incorrect type for argument 'age' in function 'print_adult_age': 16 is not greater than 17

print_adult_age(21) # ok

```

#### 5.3.2.6.3.1. Notes on Annotated

We can provide multiple `FieldCheck` in a single annotation (e.g. `Annotated[X, MyFieldCheck(), MyOtherFieldCheck()]`) and all of them will need to pass for the type to be considered correct. Annotations can also be nested, and are equivalent to providing all annotations in the same object: `Annotated[X, MyFieldCheck(), MyOtherFieldCheck()]` is the same as `Annotated[Annotated[X, MyFieldCheck()], MyOtherFieldCheck()]`.

Annotated types can also be used in complex types: `List[PositiveInt]`, for example, will check that the value is a list and that every element in the list be an integer that meets the `GreaterThanCheck(0)` check. Annotating `Union` acts as annotating each of the elements in the union. `Annotated[Union[X, Y], MyFieldCheck()]` is equivalent to `Union[Annotated[X, MyFieldCheck()], Annotated[Y, MyFieldCheck()]]`.

Whenever we type check a value, we checker will first check that the "core" type is correct. Then, it will check each `FieldCheck` from "left" to "right". For example, when we type check `Annotated[int, MyFieldCheck(), MyOtherFieldCheck()]`, we first check that the value is an `int`, then we call `MyFieldCheck().check`, and finally `MyOtherFieldCheck().check`.

This means that you can always assume that the previous checks have passed before reaching the next check function. This allows us to design faster and shorter `FieldCheck` classes. For example, we can design a `PositiveCheck` that assumes that the value supports the `>` operator because we expect this `PositiveCheck` to be used only with numerics.

#### 5.3.2.6.3.2. Boolean operators with FieldCheck

It is possible to manipulate `FieldCheck` using the operators `&`, `|`, and `^`, which will generate a new `FieldCheck`:

- `&`: the "and" operator. Combines two `FieldCheck` into a single `FieldCheck` that passes only if both `FieldCheck` pass independently. Note that usage of this operator is discouraged over simply providing both checks separately. `validators=[MyFieldCheck(), MyOtherFieldCheck()]`, or `Annotated[X, MyFieldCheck(), MyOtherFieldCheck()]` is preferable over `validators=[MyFieldCheck() & MyOtherFieldCheck()]`, or `Annotated[X, MyFieldCheck() & MyOtherFieldCheck()]`. Both syntax are equivalent when type checking, but keeping the checks separated facilitates analysis of the types and is necessary for certain systems to work.
- `|`: the "or" operator. Combines two `FieldCheck` into a single `FieldCheck` that passes if at least one of the `FieldCheck` passes (`MyFieldCheck() | MyOtherFieldCheck()`).
- `^`: the "inversion" operator. Modifies a single `FieldCheck` (`^MyFieldCheck()`). The generated `FieldCheck` will only pass if the original check would not have passed.

Combinations of these operators are possible, e.g. `^(check_a & (check_b | check_c))`.

#### 5.3.2.6.3.3. Built-in FieldCheck

There are several `FieldCheck` already defined in "Dutils.typing.field\_check" that cover most use cases. For a detailed description on each of them, the user is encouraged to read the extensive documentation written in "Dutils.typing.field\_check". In this section, we will briefly cover them:

- **SizeCheck:** A FieldCheck that passes if the given iterable has the correct size. For example, `SizeCheck(7)` will check that the given value is a vector of size 7. Similarly, `SizeCheck(3, 4)` will check that the value is a matrix with shape 3x4. In place of an integer, `None` can be specified to denote that any length is acceptable. `SizeCheck(3, None)` will check that the value is a matrix with with 3 rows, but any number of columns. `SizeCheck(None)` will check that the value is a vector (is 1-dimensional), while `SizeCheck(None, None)` will check that the value is a matrix (2D). A correctly formatted string can also be used to define inequality checks:
  - `SizeCheck("<7")` ensures size is less than 7
  - `SizeCheck(">=2")` ensures size is greater/equal than 2
  - `SizeCheck(">1 < 5")` ensures size is between 1 and 5
- **QuantityCheck:** A FieldCheck that passes if the value has the correct quantity. For example, `QuantityCheck("Length")` will check that the given value has units of length (meters, feet...). It is possible to specify whether values without units (i.e. regular floats) are acceptable. By default, this is `True`, as most DARTS code accepts unitless values by assuming the system default units. To forbid unitless values use `QuantityCheck(..., unspecified_ok=False)`.
- **RangeCheck:** A FieldCheck that passes if the input is within a range. For example, `RangeCheck(gt=0, lt=1)` will check that the value is greater than zero and lower or equal than 1. `RangeCheck(gt=0, lt=1)` will check that the value is greater or equal than zero and lower than 1. Omitting one of the bounds is possible: `RangeCheck(gt=0)` will only check that the value is greater than zero. Mathematical interval notation is supported: `RangeCheck(gt=0, lt=1)` is equivalent to `RangeCheck((0, 1))`. `RangeCheck(gt=0, lt=1)` is equivalent to `RangeCheck([0, 1])`. Units are supported in `RangeCheck`, but the quantity must be provided explicitly: `RangeCheck([1*ft, 5*ft], quantity="Length")` will first check that the value has units of quantity "Length". Then, it will ensure that the value is between one and five feet. If the given value has no units, then the system default are used in the comparison. `RangeCheck` can also validate (nested) collections of values: if a list is provided, for example, it will only pass if every element is within the range.
- **ChoicesCheck:** Forces the value to be in the given container. For example, `ChoicesCheck([1, 2])` and `ChoicesCheck({1, 2})` are equivalent and will only allow the values 1 or 2. `ChoicesCheck` has two built-in subclasses, `ModelEnumCheck` and `KnownTopoCheck`. `ModelEnumCheck` will check that the value is a valid enum defined in a `Model`, for example `ModelEnumCheck("GeneralModels.ExternalDisturbance", "DisturbanceMode")` will only accept the values "BODY" or "INERTIAL". `KnownTopoCheck()`, on the other hand, will only accept the names of known topos, for example "SphericalSun" or "EarthWGS84". All `ChoicesCheck` classes can also validate (nested) collections of values: if a list is provided, for example, it will only pass if every element is a valid choice.
- **RealCheck:** used to check floats or collections of floats (lists, numpy arrays, nested lists...) and forbid the special values "NaN", "+inf", and "-inf". However, these values may be permitted by using special keywords. `RealCheck(pos_inf_ok=True)` will fail for "NaN" and "-inf", but not for "+inf".
- **PathCheck:** Performs certain checks to strings or collections of strings to ensure they are valid paths. Existence can be checked, as well as whether the path points to a file or a directory, and whether these are writable or readable.
- **NDArrayTypeCheck:** checks that a numpy array has the correct type: `NDArrayTypeCheck(int)` will fail for `np.array([1.2, 3.4])` but pass for `np.array([1, 3])` and `np.array([True, False])` (as `bool` are subclasses of `int`).
- **Math-related:** performs certain checks on vectors or matrices. There are: `MonotonicCheck`, `NormCheck`, `SymmetricCheck`, `DeterminantCheck`, `DefinitenessCheck`.

#### 5.3.2.6.3.4. Built-in type alias

"`Dutils.typing.Dtyping`" provides a series of built-in type alias that cover most use cases in DARTS. Inspecting the definition of these types is a great exercise to understand how typing and `FieldCheck` work. In this section we provide a brief look at some of these types:

- **Real:** A float that cannot be "NaN", "+inf", or "-inf". Should be used over float for most use cases.
- **Array[T]:** A list or tuple of unknown size whose elements should be of type `T`. `T` is a `TypeVar`, which means that it can be substituted for an actual type. For example, the type `Array[str]` will permit tuples and lists of strings.
- **FloatArray, RealArray, IntArray, BoolArray:** A list, tuple, or numpy array of float, real (float not "NaN", "+inf", or "-inf"), integers, or boolean types. The numpy arrays have additional `FieldCheck` that forces them to be one dimensional. These can be used with `SizeCheck` to request arrays of specific sizes. For example, `Annotated[FloatArray, SizeCheck(3), RangeCheck([0, 1])]` could represent a color specification where the R, G, B values are given as floats between 0 and 1. Moreover, `SOA_Py.SOAVectorBase` is accepted for `FloatArray` and `RealArray`.
- **Matrix[T]:** Similar to `Array[T]`, except that this accepts lists of lists, lists of tuples, tuples of lists, and tuples of tuples. For example, `Matrix[MyObject]` would require nested sequences of `MyObject`. Note that the nested lists/tuples must all have the same length.
- **FloatMatrix, RealMatrix, IntMatrix, BoolMatrix:** matrix (as defined above) or numpy arrays of float, real, integers or boolean types. Similarly to their array equivalents, these are commonly used along side `SizeCheck`. For example, "`Dtyping`" includes a `RotationMatrix` type that is defined as `Annotated[RealMatrix, SizeCheck(3, 3), DeterminantCheck(1)]`. Moreover, `SOA_Py.SOAMatrixBase` is accepted for `FloatMatrix` and `RealMatrix`.
- **PositiveFloat, NegativeFloat, NonPositiveFloat, NonNegativeFloat, PositiveReal...:** floats, reals, and integers types that must be greater/lower/lower or equal/greater or equal to zero.
- **FilePath, DirPath:** strings that must represent valid paths to an existing file or directory.
- **Quaternion:** Either a 4-length array with norm 1 or a `SOA_Py.SOAQuaternion`.
- **Time, Mass, Length...:** reals that may optionally have units of quantity "Time", "Mass", "Length"... For example, `1.32` and `1.32 * ft` are acceptable values for the type `Length`, but not `1.32 * s`.
- **TimeArray, MassArray, LengthArray...:** real arrays that may optionally have units of quantity "Time", "Mass", "Length"... For example, `[1.32, 2.3]` and `[1.32, 23] * ft` are acceptable values for the type `LengthArray`, but not `[1.32, 2.3] * s`. Much like `RealArray`, these types are usually annotated with `SizeCheck`. For example, we might define `Position = Annotated[LengthArray, SizeCheck(3)]` to create a position vector type.
- **TimeMatrix, MassMatrix, LengthMatrix...:** real matrix that may optionally have units of quantity "Time", "Mass", "Length"... For example, `[[1.32, 2.3], [4, 5]]` and `[[1.32, 2.3], [4, 5]] * ft` are acceptable values for the type `LengthMatrix`, but not `[[1.32, 2.3], [4, 5]] * s`.
- **StrictTime, StrictTimeArray, StrictTimeMatrix, StrictMass...:** Like their non-strict equivalents, except that the values *must* have units. For example, `1.32 * ft` is an acceptable value for the type `StrictLength`, but not `1.32` or `1.32 * s`.

#### 5.3.2.7. Type hint decision tree

This section will help you find an appropriate type hint for any input you need. A decision tree will present you with questions about said input, and you can traverse it through your answers until you arrive at a square box, which contains the type hint you need.

If the type hint you find through the tree is not restricting enough, you can add additional constraints through FieldChecks, either custom-built to your needs or using one of the many built-in ones (which should cover most use cases).

Before using the tree, you might consider this list of commonly used types which are not present in the tree:

- List of built-in commonly used types

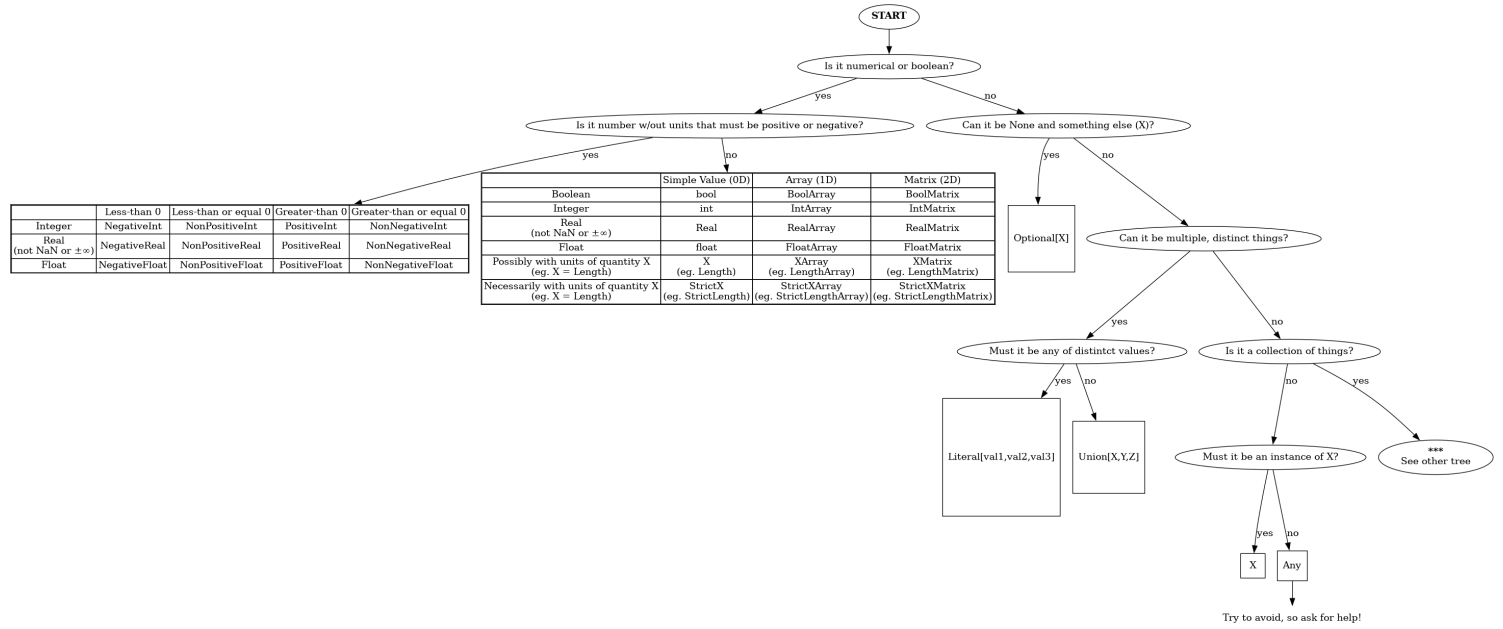


Figure 32. The decision tree

The decision tree (continued)

Failed to generate image: dot failed: Error: <stdin>: syntax error in line 2 near 'star\_2'

```
star_2 [label=<<b>***</b>>]

dictish [label="Does have to resemble a 'dict'? (it has keys and values)"]
keys_apriori [label="Are the keys strings and\you know the keys a priori?"]
  input_dict [shape=square, label="InputDict"]
change_keys [label="Are you going to change the values stored?\n('my_map[\"test\"] = 2)"]
  mapping [shape=square, label="Mapping[K,V]"]
mapping_dict [label="Does it necessarily have to be a 'dict'?"]
  dict [shape=square, label="Dict[K,V]"]
  mutable_mapping [shape=square, label="MutableMapping[K,V]"]
repeated [label="Can it have repeated values?"]
  set [shape=square, label="Set[X]"]
by_index [label="Do you need to access the\nelements by index ('obj[1]')?"]
  iterable [shape=square, label="Iterable[X]"]
list_or_tuple [label="Does it necessarily have\nto be a list or tuple?"]
  sequence [shape=square, label="Sequence[X]"]
maybe_list [label="Can it be a list?"]
  can_tuple [label="Can it be a tuple?"]
  array [shape=square, label="Array[X]"]
  list [shape=square, label="List[X]"]
specific_tuple [label="Must the tuple have specific\ntypes at each index?"]
  tuple_dddodt [shape=square, label="Tuple[X,...]"]
  tuple [shape=square, label="Tuple[X,Y,Z]"]

star_2 -> dictish

dictish -> keys_apriori [label="yes"]
dictish -> repeated [label="no"]

keys_apriori -> input_dict [label="yes"]
keys_apriori -> change_keys [label="no"]

change_keys -> mapping_dict [label="yes"]
change_keys -> mapping [label="no"]

mapping_dict -> dict [label="yes"]
mapping_dict -> mutable_mapping [label="no"]

repeated -> by_index [label="yes"]
repeated -> set [label="no"]

by_index -> list_or_tuple [label="yes"]
by_index -> iterable [label="no"]

list_or_tuple -> maybe_list [label="yes"]
list_or_tuple -> sequence [label="no"]

maybe_list -> can_tuple [label="yes"]
maybe_list -> specific_tuple [label="no"]

can_tuple -> array [label="yes"]
can_tuple -> list [label="no"]

specific_tuple -> tuple [label="yes"]
specific_tuple -> tuple_dddodt [label="no"]
```

Note that:

- The recommended type hints can be imported from `Dutils.typing` or `Dutils.typing.Dtyping`.
- To decide what the type hint for the placeholders `X`, `Y`, `Z`, `K`, `V` should be, go back to the start of the tree.
- `Literal`, `Union`, and `Tuple` can take as many arguments as you need: `Union[X, Y]`, `Union[X, Y, Z]`, `Union[X, Y, Z, W]` ... To find what should
- In `Mapping`, `MutableMapping`, and `Dict`, `K` refers to the type of the keys and `V` to the type of the values.
- `Literal` should only take **instances** of string, integers, None, or enums. For example: `Literal["a", None, 3, 5, Color.RED]`.

## 5.4. Software

### 5.5. Raw documentation



TBD: Need scrubbing before integration.

#### 5.5.1. Handling different versions of typing and typing\_extension Python modules



TBD: Needs scrubbing. Notes brought over from [issue](#)

([https://dartsgitlab-internal.jpl.nasa.gov/darts-lab/jpl-internal/all-access/development/simulation-framework/dshellcommon/-/issues/29#note\\_9371](https://dartsgitlab-internal.jpl.nasa.gov/darts-lab/jpl-internal/all-access/development/simulation-framework/dshellcommon/-/issues/29#note_9371)).

While this particular issue has been solved, it is possible that similar errors might appear in the future. These issues happen because `typing` is a relatively recent development of the python standard (as of 2022) and is continuously improving.



It is best if we are able to use the latest features of the `typing` system, as they are handy and facilitate development. However, we need to keep backward compatibility. This compatibility should be maintained through the `Dutils.typing` module. Internally, the `Dutils.typing` module tries to import type annotation constructs from the standard `typing` module, but if it cannot do so it will use the `typing_extensions` library. This library backports new `typing` features to older python versions (so even if `Annotated` is only available in 3.8, one can access it from `typing_extensions` in 3.7).

If a type hint is failing in a machine but not in another:

- If the type hint construct is being imported from `Dutils.typing`: check that the `typing_extensions` library is updated. If `typing_extensions` is updated to the latest version, then `Dutils.typing` may not be able to ensure backward compatibility in this particular case **yet**. You will need to update `Dutils.typing` so that the versions are handled appropriately.
- If the type hint construct is being imported from a module other than `Dutils.typing`: you will have to manually introduce a backward compatibility mechanism (see below for a possible solution).

If the type hint is only being used as an annotation (i.e. is not being used at runtime to type check, like with `InputDict` or the `@type_check_fn` decorator), then you can do:

```
from Dutils.typing import TYPE_CHECKING, Any

if TYPE_CHECKING:
    from my_module import MyProblematicAnnotation
else:
    MyProblematicAnnotation = Any
```

this will make your IDE believe that the `MyProblematicAnnotation` is coming from `my_module` (so that you get appropriate hints), but at runtime, it will actually be `Any`, which is always available.

## 5.6. Sphinx documentation

### 5.6.1. Introduction

### 5.6.2. DshellEnv Classes

#### 5.6.2.1. Mathematical Constants

The file `mathConstants.h` provides a set of predefined values for common mathematical constants. These constants are defined to allow code to be more portable since the values of some of these numbers varies based on a number of factors such as machine precision, etc. Using these predefined constants reduce the impact of such variations and should make our code more robust and portable.

The following math constants are available from C++ as defined in the `DshellEnv/mathConstants.h` header file. These constants are also available via python by importing them:

```
>>> from Math import MathConstants_Py
```

and using the same constants via python.

Note that the use of these constants is entirely optional. All these constants are in the `DMath` namespace.

#### Source for the `DshellEnv/mathConstants.h` header file

- ▶ [Click to see the `DshellEnv/mathConstants.h` script](#)

#### 5.6.2.1.1. Related Regression Tests

#### MathConstants unit tests

- ▶ [Click to see the `DshellEnv/test/test\_MathFunctions/script.py` script](#)

### 5.6.2.2. Math Functions

#### 5.6.2.2.1. Introduction

Add `mathFunctions` introduction.

#### 5.6.2.2.2. `mathFunctions` API Documentation



#### Note

For Doxygen documentation of the `DMath` namespace, including functions in `mathFunctions`, please see: [DMath](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/DshellEnv/html/namespaceDMath.html) (<https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/DshellEnv/html/namespaceDMath.html>) namespace Doxygen documentation.

The following math functions are available from C++ as defined in the `DshellEnv/mathFunctions.h` header file. These functions are also available via python by importing them:

```
>>> from Math import MathFunctions_Py
```

and using the same functions via python.

#### 5.6.2.2.1. This is the source code of the `DshellEnv/mathFunctions.h` header file

#### Source for the `DshellEnv/mathFunctions.h` header file

- ▶ [Click to see the `DshellEnv/mathFunctions.h` script](#)

#### 5.6.2.2.3. Related Regression Tests

#### MathFunctions unit tests

► [Click to see the DshellEnv/test/test\\_MathFunctions/script.py script](#)

### 5.6.2.3. DataAccessUtils

#### 5.6.2.3.1. Introduction

Add DataAccessUtils introduction.

#### 5.6.2.3.2. Related Regression Tests

##### Test DAO Constants

► [Click to see the DshellEnv/test/test\\_DataAccessUtils/test\\_DoubleDAO.py script](#)

##### DAO unit test

► [Click to see the DshellEnv/test/test\\_DataAccessUtils/unitTest.py script](#)

#### 5.6.2.3.3. DoubleDAO API Documentation



##### Note

For Doxygen documentation of the module, please see: [DoubleDAO <DataAccessUtils\\_Py::DoubleDAO>](#)  
([https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/DshellEnv/html/classDataAccessUtils\\_Py\\_1\\_1DoubleDAO.html](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/DshellEnv/html/classDataAccessUtils_Py_1_1DoubleDAO.html))

```
class Dshell.DataAccessUtils_Py.DoubleDAO(DoubleDAO self) → DoubleDAO
class Dshell.DataAccessUtils_Py.DoubleDAO(DoubleDAO self, double val) → DoubleDAO
```

PYTHON

The DoubleDAO class provides for creation of DAO objects based upon a constant double value.

This class is intended as a concrete implementation that will be extended via Python.

C++ includes: DoubleDAO.h

Typical constructor that accepts a double value to return as the DAO result.

- val  
is a double data value to return for this DAO.

noindex

#### 5.6.2.3.4. DvarDoubleDAO API Documentation



##### Note

For Doxygen documentation of the module, please see: [DvarDoubleDAO <DataAccessUtils\\_Py::DvarDoubleDAO>](#)  
([https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/DshellEnv/html/classDataAccessUtils\\_Py\\_1\\_1DvarDoubleDAO.html](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/DshellEnv/html/classDataAccessUtils_Py_1_1DvarDoubleDAO.html))

```
class Dshell.DataAccessUtils_Py.DvarDoubleDAO(DvarDoubleDAO self) → DvarDoubleDAO
class Dshell.DataAccessUtils_Py.DvarDoubleDAO(DvarDoubleDAO self, std::string in_string) → DvarDoubleDAO
class Dshell.DataAccessUtils_Py.DvarDoubleDAO(DvarDoubleDAO self, DVar::DoubleLeaf * dvar) → DvarDoubleDAO
```

PYTHON

The DvarDoubleDAO class provides for creation of DAO objects based upon a Dvar string.

This capability is especially useful for providing interpolation tables with DAO objects that obtain their value from a Dvar object. Note at this time only double data types are supported.

C++ includes: DvarDAO.h

Typical constructor that accepts a Dvar specification string and assigns the result to the class held dvar variable.

- spec\_string  
is the valid Dvar specification for a double data type parameter

noindex

#### 5.6.2.3.5. DvarDoubleVectorDAO API Documentation



##### Note

For Doxygen documentation of the module, please see: [DvarDoubleVectorDAO <DataAccessUtils\\_Py::DvarDoubleVectorDAO>](#)  
([https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/DshellEnv/html/classDataAccessUtils\\_Py\\_1\\_1DvarDoubleVectorDAO.html](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/DshellEnv/html/classDataAccessUtils_Py_1_1DvarDoubleVectorDAO.html))

```
class Dshell.DataAccessUtils_Py.DvarDoubleVectorDAO(DvarDoubleVectorDAO self) → DvarDoubleVectorDAO
class Dshell.DataAccessUtils_Py.DvarDoubleVectorDAO(DvarDoubleVectorDAO self, std::string spec_string) → DvarDoubleVectorDAO
class Dshell.DataAccessUtils_Py.DvarDoubleVectorDAO(DvarDoubleVectorDAO self, DVar::DoubleVectorLeaf * dvar) → DvarDoubleVectorDAO
class Dshell.DataAccessUtils_Py.DvarDoubleVectorDAO(DvarDoubleVectorDAO self, std::string in_string, long inSlice) → DvarDoubleVectorDAO
```

PYTHON

The DvarDoubleVectorDAO class provides for creation of DAO objects based upon a Dvar string.

This capability is especially useful for providing interpolation tables with DAO objects that obtain their value from a Dvar object. Note at this time only double data types are supported.

C++ includes: DvarVectorDAO.h

Typical constructor that accepts a Dvar specification string and assigns the result to the class held dvar variable.

- spec\_string

is the valid Dvar specification for a double vector data type parameter

noindex

#### 5.6.2.3.6. iDataAccessObjectDouble API Documentation



##### Note

For Doxygen documentation of the module, please see: [iDataAccessObject <DataAccessUtils\\_Py::iDataAccessObjectDouble>](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/DshellEnv/html/classDataAccessUtils_Py_1_1iDataAccessObjectDouble.html)  
([https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/DshellEnv/html/classDataAccessUtils\\_Py\\_1\\_1iDataAccessObjectDouble.html](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/DshellEnv/html/classDataAccessUtils_Py_1_1iDataAccessObjectDouble.html))

```
class Dshell.DataAccessUtils_Py.iDataAccessObjectDouble(*args, **kwargs)
```

PYTHON

C++ includes: iDataAccessObject.h

noindex

#### 5.6.2.3.7. IndependentArray API Documentation



##### Note

For Doxygen documentation of the module, please see: [IndependentArray <DataAccessUtils\\_Py::IndependentArray>](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/DshellEnv/html/classDataAccessUtils_Py_1_1IndependentArray.html)  
([https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/DshellEnv/html/classDataAccessUtils\\_Py\\_1\\_1IndependentArray.html](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/DshellEnv/html/classDataAccessUtils_Py_1_1IndependentArray.html))

```
class Dshell.DataAccessUtils_Py.IndependentArray(IndependentArray self) → IndependentArray
class Dshell.DataAccessUtils_Py.IndependentArray(IndependentArray self, DoubleVector arg2) → IndependentArray
class Dshell.DataAccessUtils_Py.IndependentArray(IndependentArray self, int arg2, double * arg3) → IndependentArray
class Dshell.DataAccessUtils_Py.IndependentArray(IndependentArray self, IndependentArray arg2) → IndependentArray
```

PYTHON

This class defines an independent data array used for interpolating a dimension of an interpolation table.

The class owns the array memory that is operated on and will copy data into the memory as required.

C++ includes: IndependentArray.h

Constructor that allocates memory for the data array and copies data from the input vector:

- data
- is data container to copy data from for this independent array instance.

noindex

#### 5.6.2.3.8. IntegrationTimeDAO API Documentation



##### Note

For Doxygen documentation of the module, please see: [IntegrationTimeDAO <DataAccessUtils\\_Py::IntegrationTimeDAO>](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/DshellEnv/html/classDataAccessUtils_Py_1_1IntegrationTimeDAO.html)  
([https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/DshellEnv/html/classDataAccessUtils\\_Py\\_1\\_1IntegrationTimeDAO.html](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/DshellEnv/html/classDataAccessUtils_Py_1_1IntegrationTimeDAO.html))

```
class Dshell.DataAccessUtils_Py.IntegrationTimeDAO(IntegrationTimeDAO self) → IntegrationTimeDAO[source]
class Dshell.DataAccessUtils_Py.IntegrationTimeDAO(IntegrationTimeDAO self, DVar::DoubleLeaf * dvar) → IntegrationTimeDAO
```

PYTHON

C++ includes: IntegrationTimeDAO.h

Constructor that accepts the Dvar object.

- dvar
- is the valid Dvar object for sim time

noindex

#### 5.6.2.3.9. InterpolationTable API Documentation



##### Note

For Doxygen documentation of the module, please see: [InterpolationTable <DataAccessUtils\\_Py::InterpolationTable>](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/DshellEnv/html/classDataAccessUtils_Py_1_1InterpolationTable.html)  
([https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/DshellEnv/html/classDataAccessUtils\\_Py\\_1\\_1InterpolationTable.html](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/DshellEnv/html/classDataAccessUtils_Py_1_1InterpolationTable.html))

```
class Dshell.DataAccessUtils_Py.InterpolationTable(InterpolationTable self, int arg2) → InterpolationTable
class Dshell.DataAccessUtils_Py.InterpolationTable(InterpolationTable self, InterpolationTable arg2) → InterpolationTable
```

PYTHON

Purpose: This is the class that does all of the number crunching for N-dimensional interpolation tables.

Assumptions and Limitations: The data in the dependent variable array(s) should be structured so that the order of the independent variable(s) in the table matches the order of the indices in the array. The data in the independent variable array must be ordered in ascending magnitude. The only way to get a value returned from this table is to supply the value of the independent variable used to index into the table.

Description: The approach used is to recursively solve the equation.

$$d = ( d[j+1] - d[j] ) * \frac{ ( i - i[j] ) }{ ( i[j+1] - i[j] ) } + d[j]$$

Where  $i$  represents a value for the independent variable and  $d$  represents a value for the dependent variable.  $j$  represents the index in the independent variable array such that  $i[j] = i < i[j+1]$  except for the special cases of  $i < i[0]$  or  $i > i[n]$  where  $n$  is the largest index in the array.

The problem is solved in two parts. The first part is to determine the value of  $j$  and to compute the value of the  $i$  factor in the equation for each independent variable. The second part is to recursively compute the values of  $d$  for each independent variable.

An additional complexity arises because the dependent variable table must be treated as a linear array. A multiplicative offset scheme is used to navigate through the array. As the  $i$  values are being determined for each independent variable, an offset multiplier, whose initial value is 1, is multiplied by the size of the independent variable array. To move to the correct point in the table while solving for the values of  $d$ , the index is multiplied by the multiplier. As the recursive interpolator moves down through each independent variable, the multiplier is divided by the size of the array for that independent variable.

History: The logic and source code for this class was derived from `interp.c` used in STAMPS, and later used in ANTARES. Originally the logic was programmed in C, and has since been converted to C++ for use in COMPASS. More C → C++ refactoring could certainly be performed, but was not accomplished due to schedule constraints.

C++ includes: `InterpolationTable.h`

Constructor used for interpolation tables that requires the dimension or number of independent variables to be specified at construction.

- `numIndVars`  
is the number of independent variables or dimensions of the interpolation table.

req COMPASS-05.02-REQ-00001

noindex

#### 5.6.2.3.10. InterpolationTableError API Documentation



##### Note

For Doxygen documentation of the module, please see: [InterpolationTableError <DataAccessUtils\\_Py::InterpolationTableError>](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/DshellEnv/html/classDataAccessUtils_Py_1_1InterpolationTableError.html)  
([https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/DshellEnv/html/classDataAccessUtils\\_Py\\_1\\_1InterpolationTableError.html](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/DshellEnv/html/classDataAccessUtils_Py_1_1InterpolationTableError.html))

```
class Dshell.DataAccessUtils_Py.InterpolationTableError(InterpolationTableError self, std::string const & func="", std::string const & msg="") → InterpolationTableError
```

PYTHON

C++ includes: `InterpolationTable.h`

noindex

#### 5.6.2.3.11. InvalidDimensionError API Documentation



##### Note

For Doxygen documentation of the module, please see: [InvalidDimensionError <DataAccessUtils\\_Py::InvalidDimensionError>](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/DshellEnv/html/classDataAccessUtils_Py_1_1InvalidDimensionError.html)  
([https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/DshellEnv/html/classDataAccessUtils\\_Py\\_1\\_1InvalidDimensionError.html](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/DshellEnv/html/classDataAccessUtils_Py_1_1InvalidDimensionError.html))

```
class Dshell.DataAccessUtils_Py.InvalidDimensionError(InvalidDimensionError self, size_t const used_dimension, size_t const table_dimension) → InvalidDimensionError
```

PYTHON

C++ includes: `InterpolationTable.h`

noindex

#### 5.6.2.3.12. Table API Documentation



##### Note

For Doxygen documentation of the module, please see: [Table <DataAccessUtils\\_Py::Table>](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/DshellEnv/html/namespaceDataAccessUtils_Py.html)  
([https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/DshellEnv/html/namespaceDataAccessUtils\\_Py.html](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/DshellEnv/html/namespaceDataAccessUtils_Py.html))

```
class Dshell.DataAccessUtils_Py.Table(Table self) → Table  
class Dshell.DataAccessUtils_Py.Table(Table self, double Value) → Table
```

PYTHON

Purpose: The `Table` class is used primarily as a base for `InterpolationTables` and `independent_arrays`, and provides core functionality needed by all tables.

While this class is mostly used as a base class, it can be instantiated directly to represent constant value tables. All tables will implement the `iDataAccessObject` interface, and hence can be used directly as a `DataAccessObject`.

Assumptions and Limitations: None

C++ includes: Table.h

This constructor is for a table that only consists of a constant value.

noindex

### 5.6.2.4. Interpolation Tables

#### 5.6.2.4.1. Class Documentation

##### 5.6.2.4.1.1. Introduction

The InterpolationTable class provides the capability to define and use 10-dimensional interpolation tables.

##### 5.6.2.4.1.2. Related Regression Tests



**Note**

Test cases are TBD

##### 5.6.2.4.2. Interpolation Table API Documentation



**Note**

For Doxygen documentation, please see: `InterpolationTable<utils_Py::InterpolationTable>`

For C++ API documentation, please see: `Interpolation Table Component<InterpolationTable>`

```
class Dshell.DataAccessUtils_Py.InterpolationTable(InterpolationTable self, int arg2) → InterpolationTable
class Dshell.DataAccessUtils_Py.InterpolationTable(InterpolationTable self, InterpolationTable arg2) → InterpolationTable
-----
```

Purpose: This is the class that does all of the number crunching for N-dimensional interpolation tables.

Assumptions and Limitations: The data in the dependent variable array(s) should be structured so that the order of the independent variable(s) in the table matches the order of the indices in the array. The data in the independent variable array must be ordered in ascending magnitude. The only way to get a value returned from this table is to supply the value of the independent variable used to index into the table.

Description: The approach used is to recursively solve the equation.

$$d = ( d[j+1] - d[j] ) * \frac{( i - i[j] )}{( i[j+1] - i[j] )} + d[j]$$

Where i represents a value for the independent variable and d represents a value for the dependent variable. j represents the index in the independent variable array such that  $i[j] = i < i[j+1]$  except for the special cases of  $i < i[0]$  or  $i > i[n]$  where n is the largest index in the array.

The problem is solved in two parts. The first part is to determine the value of j and to compute the value of the i factor in the equation for each independent variable. The second part is to recursively compute the values of d for each independent variable.

An additional complexity arises because the dependent variable table must be treated as a linear array. A multiplicative offset scheme is used to navigate through the array. As the i values are being determined for each independent variable, an offset multiplier, whose initial value is 1, is multiplied by the size of the independent variable array. To move to the correct point in the table while solving for the values of d, the index is multiplied by the multiplier. As the recursive interpolator moves down through each independent variable, the multiplier is divided by the size of the array for that independent variable.

History: The logic and source code for this class was derived from interp.c used in STAMPS, and later used in ANTARES. Originally the logic was programmed in C, and has since been converted to C++ for use in COMPASS. More C → C++ refactoring could certainly be performed, but was not accomplished due to schedule constraints.

C++ includes: InterpolationTable.h

Constructor used for interpolation tables that requires the dimension or number of independent variables to be specified at construction.

numIndVars is the number of independent variables or dimensions of the interpolation table.

req COMPASS-05.02-REQ-00001

noindex

### 5.6.2.5. Dvar Data Access Objects (DAO)

#### 5.6.2.5.1. Class Documentation

##### 5.6.2.5.1.1. Introduction

The DvarDoubleDAO class provides for creation of DAO objects based upon a Dvar string. This capability is especially useful for providing interpolation tables with DAO objects that obtain their value from a Dvar object. Note at this time only double data types are supported.

##### 5.6.2.5.1.2. Example Construction Syntax

```
from Dshell.DataAccessUtils_Py import DvarDoubleDAO
myDAO = DvarDoubleDAO('some.valid.spec.string')
myDAO()
```

##### 5.6.2.5.1.3. Related Regression Tests



**Note**

Test cases are TBD

5.6.2.5.2. Dvar Data Access Object API Documentation



**Note**

For Doxygen documentation of the module, please see: `DvarDoubleDAO<utils_Py::DvarDoubleDAO>`

```

class Dshell.DataAccessUtils_Py.DvarDoubleDAO(DvarDoubleDAO self) → DvarDoubleDAO
class Dshell.DataAccessUtils_Py.DvarDoubleDAO(DvarDoubleDAO self, std::string in_string) → DvarDoubleDAO
class Dshell.DataAccessUtils_Py.DvarDoubleDAO(DvarDoubleDAO self, DVar::DoubleLeaf * dvar) → DvarDoubleDAO
-----

```

PYTHON

The DvarDoubleDAO class provides for creation of DAO objects based upon a Dvar string.

This capability is especially useful for providing interpolation tables with DAO objects that obtain their value from a Dvar object. Note at this time only double data types are supported.

C++ includes: DvarDAO.h

Typical constructor that accepts a Dvar specification string and assigns the result to the class held dvar variable.

- `spec_string`  
is the valid Dvar specification for a double data type parameter

noindex

5.6.2.6. Dwatch

5.6.2.6.1. Introduction

Add introductory material for DwatchLegacy

5.6.2.6.2. Related Regression Tests

**DwatchLegacy usage**

▶ [Click to see the DshellEnv/test/test/test\\_Dwatch/script.py script](#)

5.6.2.6.3. Dwatch Class API Documentation



**Note**

For Doxygen documentation, please see: `DwatchLegacy<DwatchLegacy::DwatchLegacy>`  
([https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/DshellEnv/html/classDwatchLegacy\\_1\\_1DwatchLegacy.html](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/DshellEnv/html/classDwatchLegacy_1_1DwatchLegacy.html))

File: DwatchLegacy.py Author: C. Lim (June 7, 2005)

This module contains classes to log simulation data. The main class is DwatchLegacy which manages groups (class DwatchLegacyGroup) of data items (class DwatchLegacyItem). DwatchLegacy reads its input parameters from a ConfigObj file. See test-DwatchLegacy for an example.

5.6.2.7. FSM

5.6.2.7.1. Introduction

Add introductory material for FSM

5.6.2.7.2. Related Regression Tests

**fsm usage**

▶ [Click to see the DshellEnv/test/test/test\\_fsm/script.py. script](#)

5.6.2.7.3. FSM Class API Documentation



**Note**

For Doxygen documentation, please see: `FSM <Dfsm::FSM>`

This module implements a Finite State Machine (FSM).

5.6.2.7.3.1. Example Usage:

```

>>> import Dfsm
>>> fs = Dfsm.FSM()
>>> fs.addState('init', None, None, None)
>>> fs.addState('s1', None, None, None)
>>> fs.addTransition(True, 'init', None, 's1')
>>> fs.setCurrentState('init')
>>> fs.step()
performing actions for state init
performing exit actions for state init
performing transition action: init -> s1
performing entry actions for state s1
>>> print(fs)
Current state = ['s1']
Event table = {'init': 0, 's1': 1}
>>> fs.getCurrentStateNames()

```

## 5.6.2.8. TrajectoryUtils

### 5.6.2.8.1. Introduction

Add TrajectoryUtils introduction.

### 5.6.2.8.2. Related Regression Tests

#### TrajectoryUtils Invariant unit tests

▶ [Click to see the DshellEnv/test/test/test\\_Invariant/script.py script](#)

#### TrajectoryUtils PlanetRelativeState unit tests

▶ [Click to see the DshellEnv/test/test/test\\_PlanetRelativeState/script.py script](#)

### 5.6.2.8.3. Invariant API Documentation



#### Note

For Doxygen documentation of the module, please see: [Invariant <TrajectoryUtils\\_Py::Invariant>](#)  
[\(https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/DshellEnv/html/classTrajectoryUtils\\_\\_Py\\_1\\_1Invariant.html\)](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/DshellEnv/html/classTrajectoryUtils__Py_1_1Invariant.html)

```
class Dutils.TrajectoryUtils_Py.Invariant(Invariant self, InvariantGravityTerms g) → Invariant
```

PYTHON

C++ includes: Invariant.h

Constructor that is defined using the Invariant Gravity Terms.

Copyright (c) 2013 National Aeronautics and Space Administration (NASA).

- g

The Invariant Gravity Terms.

All rights reserved. This software is owned by NASA.

Invariant.cc – C++ source file

PURPOSE: To convert between Osculating and Invariant orbital elements.

REFERENCE: Adapted from STAMPS 26 'models/invariant/invariant.c'

noindex

### 5.6.2.8.4. InvariantGravityTerms API Documentation



#### Note

For Doxygen documentation of the module, please see: [InvariantGravityTerms <TrajectoryUtils\\_Py::InvariantGravityTerms>](#)  
[\(https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/DshellEnv/html/classTrajectoryUtils\\_\\_Py\\_1\\_1InvariantGravityTerms.html\)](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/DshellEnv/html/classTrajectoryUtils__Py_1_1InvariantGravityTerms.html)

```
class Dutils.TrajectoryUtils_Py.InvariantGravityTerms(InvariantGravityTerms self) → InvariantGravityTerms
class Dutils.TrajectoryUtils_Py.InvariantGravityTerms(InvariantGravityTerms self, double mu, double omega, double radius, double radius_eq, double [5][5] cmn, double [5][5] snm) →
InvariantGravityTerms
```

PYTHON

C++ includes: InvariantGravityTerms.h

Constructor that is defined using the gravitational parameter, orbital rate, radius values and gravity terms.

- gravitational\_parameter

The 'mu' value for the planet.

- omega

The orbital rate.

- radius

The planet's radius.

- radius\_eq

The planet's equatoria radius.

- cnm  
C gravity coefficients (5x5 array).
- snm  
S gravity coefficients (5x5 array).

noindex

#### 5.6.2.8.5. OrbitalElements API Documentation



##### Note

For Doxygen documentation of the module, please see: [OrbitalElements <TrajectoryUtils\\_Py::OrbitalElements>](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/DshellEnv/html/classTrajectoryUtils_Py_1_1OrbitalElements.html)  
([https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/DshellEnv/html/classTrajectoryUtils\\_Py\\_1\\_1OrbitalElements.html](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/DshellEnv/html/classTrajectoryUtils_Py_1_1OrbitalElements.html))

```
class Dutils.TrajectoryUtils_Py.OrbitalElements(OrbitalElements self) → OrbitalElements
class Dutils.TrajectoryUtils_Py.OrbitalElements(OrbitalElements self, double gravitational_parameter, double radius_planet) → OrbitalElements
```

PYTHON

C++ includes: OrbitalElements.h

Constructor that is defined using the gravitational parameter and the planet radius.

- gravitational\_parameter  
The 'mu' value for the planet.
- radius\_planet  
The planet's radius.

noindex

#### 5.6.2.8.6. PlanetRelativeState API Documentation



##### Note

For Doxygen documentation of the module, please see: [PlanetRelativeState <TrajectoryUtils\\_Py::PlanetRelativeState>](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/DshellEnv/html/classTrajectoryUtils_Py_1_1PlanetRelativeState.html)  
([https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/DshellEnv/html/classTrajectoryUtils\\_Py\\_1\\_1PlanetRelativeState.html](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/DshellEnv/html/classTrajectoryUtils_Py_1_1PlanetRelativeState.html))

```
class Dutils.TrajectoryUtils_Py.PlanetRelativeState(PlanetRelativeState self) → PlanetRelativeState
class Dutils.TrajectoryUtils_Py.PlanetRelativeState(PlanetRelativeState self, double radiusEq, double radiusPol) → PlanetRelativeState
```

PYTHON

C++ includes: PlanetRelativeState.h

noindex

#### 5.6.2.8.7. RelativeState API Documentation



##### Note

For Doxygen documentation of the module, please see: [RelativeState <TrajectoryUtils\\_Py::RelativeState>](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/DshellEnv/html/classTrajectoryUtils_Py_1_1RelativeState.html)  
([https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/DshellEnv/html/classTrajectoryUtils\\_Py\\_1\\_1RelativeState.html](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/DshellEnv/html/classTrajectoryUtils_Py_1_1RelativeState.html))

```
class Dutils.TrajectoryUtils_Py.RelativeState(RelativeState self) → RelativeState[source]
```

PYTHON

C++ includes: RelativeState.h

Default constructor

Copyright (c) 2013 National Aeronautics and Space Administration (NASA).

Initializes an internal flag to track if the frames were initialized.

All rights reserved. This software is owned by NASA.

RelativeState.cpp – C++ source file

**PURPOSE:** The RelativeState class is a container object that holds the variables corresponding to the relative relationship between a primary/target and secondary/chaser object in space. The relative calculations are based primarily on the inertial positions of the two objects and their associated attitude and body attitude rates.

This class is used by both the VehicleRelativeStateSensor and the generic RelativeStateSensor models.

**COMMENTS:** 1) Suggestion to update 'quat' and 'htran' variables to use naming convention was considered but rejected because of the effect on the readability of the current code.

noindex



#### 5.6.2.8.8. VacuumImpactState API Documentation



##### *Note*

For Doxygen documentation of the module, please see: [VacuumImpactState <TrajectoryUtils\\_Py::VacuumImpactState>](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/DshellEnv/html/classTrajectoryUtils_Py_1_1VacuumImpactState.html)  
([https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/DshellEnv/html/classTrajectoryUtils\\_Py\\_1\\_1VacuumImpactState.html](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/DshellEnv/html/classTrajectoryUtils_Py_1_1VacuumImpactState.html))

```
class Dutils.TrajectoryUtils_Py.VacuumImpactState(VacuumImpactState self) → VacuumImpactState
class Dutils.TrajectoryUtils_Py.VacuumImpactState(VacuumImpactState self, double rad_eq, double rad_pol, double w, double mu) → VacuumImpactState
```

PYTHON

C++ includes: VacuumImpactState.h

noindex

#### 5.6.3. DshellEnv Reference

##### 5.6.3.1. DshellEnv Regression Tests

## 6. Ndarts

### 6.1. Background

#### 6.1.1. The multibody dynamics problem

##### 6.1.1.1. Requirements

- Speed
- ODE
- Reconfiguration
- Extend to closed chain, contact and flex body problem

##### 6.1.1.2. Minimal coordinate dynamics

##### 6.1.1.3. Solving the equations of motion

Using my substitution for Ndarts <https://ndarts>

#### 6.1.2. Reference & Source material

- Release notes appendices
- [DARTS paper](https://dartslab/References/pdf/2019-Darts.pdf) (<https://dartslab/References/pdf/2019-Darts.pdf>)
- [SOA algorithms papers](https://dartslab/References/index.php) (<https://dartslab/References/index.php>)
- [SOA book](https://dartslab.jpl.nasa.gov/SOABook/) (<https://dartslab.jpl.nasa.gov/SOABook/>)
- [SOA lecture series slides](https://dartslab.jpl.nasa.gov/soagroup/meetings.html) (<https://dartslab.jpl.nasa.gov/soagroup/meetings.html>)
- [Model/assemblies style guide](https://dartslab.jpl.nasa.gov/technotes/Modeling/DshellModeling.pdf) (<https://dartslab.jpl.nasa.gov/technotes/Modeling/DshellModeling.pdf>)
- [Ndarts Doxygen docs](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/Ndarts/html/) (<https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/Ndarts/html/>)
- [Team talks](https://dlabportal.jpl.nasa.gov/resources/darts-lab-talks/) (<https://dlabportal.jpl.nasa.gov/resources/darts-lab-talks/>)
- [2019, 2020 course slides](https://dlabportal.jpl.nasa.gov/documentation/coursetutorial-links/) (<https://dlabportal.jpl.nasa.gov/documentation/coursetutorial-links/>)
- [Sphinx docs](https://dartslab.jpl.nasa.gov/dlabdocs/) (<https://dartslab.jpl.nasa.gov/dlabdocs/>)
- [Jupyter notebooks](https://dlabnotebooks.jpl.nasa.gov/hub/login) (<https://dlabnotebooks.jpl.nasa.gov/hub/login>)
- [DARTS Lab QA site](https://dartslab.jpl.nasa.gov/qa/) (<https://dartslab.jpl.nasa.gov/qa/>)

## 6.2. Design

### 6.2.1. Architecture overview

#### 6.2.1.1. Dependencies

Frames layer, spec nodes, SOA classes, spatial inertia related content needed by this module.

NdartsConstraints, NdartsContact, NdartsFlex, Dshell++, NdartsContactModels etc modules that depend on this module.

#### 6.2.1.2. Structural overview

An overview of the various classes and how they fit together

The Ndarts implementation of DARTS (???) can model a wide range of multibody mechanisms. DARTS/Dshell simulations generally involve at least a single rigid body (for the a spacecraft, for example) or potentially a complex, multiply connected multibody system.

For more detailed background information, please see:

- *Robot and Multibody Dynamics, Analysis and Algorithms*. Abhi Jain. Springer. 2011.
- Darts Manual 1997.

#### 6.2.1.3. DARTS Multibody Modeling Concepts

##### • **Tree Topology Systems.**

Tree-topology systems are multibody systems with no closed-chains.

In many cases, the multibody physics of the system can be modeled as a "Tree Topology System". Ndarts can handle closed-chain systems, but often a Tree topology system is adequate.

"Closed-chain" means that a set of bodies are connected to each other in a loop, or "closed chain".

##### **In Tree Topology Systems:**

- there is one and only one path from one body to another
- each body has only a single *parent*
- each body can have multiple *child* bodies.

- **Flexible Body.**

A **Flexible Body** has *deformation* degrees-of-freedom (dofs) in addition to rigid-body *articulation dofs*.

- **Spatial Vectors.**

These are 6-dimensional vectors that simplify many dynamics related expressions.

- A *spatial velocity* vector for a coordinate frame is the 6-dimensional vector formed by the concatenation of the angular velocity and linear velocity vectors for the frame.
- The *spatial force* vector associated with a frame is the 6-dimensional vector formed by the concatenation of the moment and force vectors associated with the frame.
- The SOA vector/matrix math library<index.html> implements spatial vector math (as well as other vector/matrix math used in Darts).

- **Nodes.**

Set of discrete points in the body:

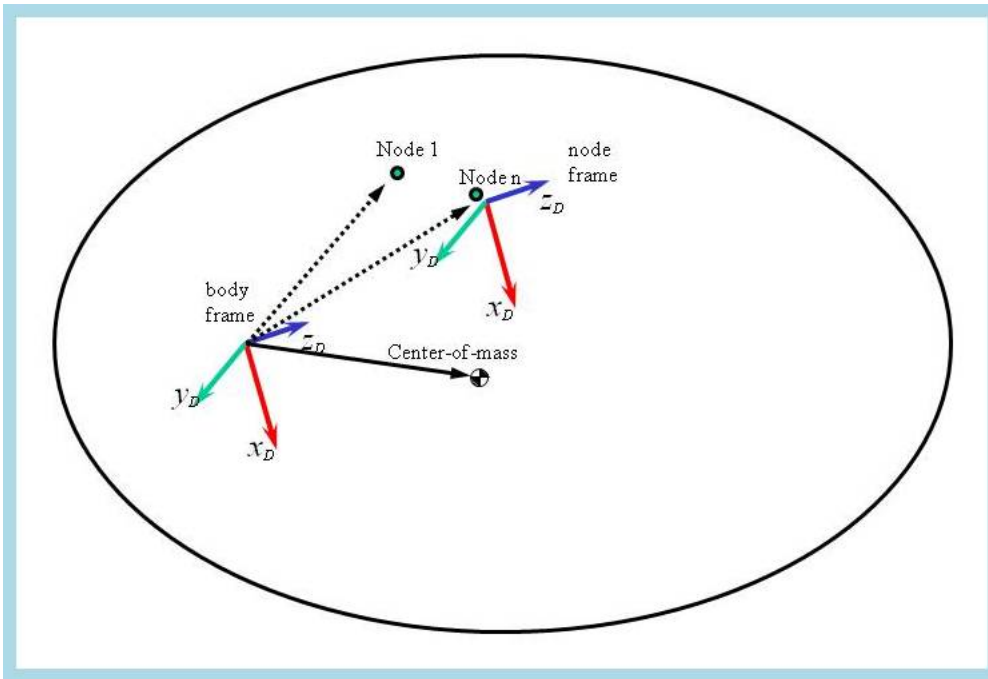


Figure 33. Body Frame and Nodes

- **Nodes:**

- can be defined by the user
- are associated with *hinges* interconnecting bodies
- can arise from modeling flexible modes: the structural model of a flexible body involves a spatial discretization of the body into a collection of nodal elements

- **Node Frames**

- each node has its own reference frame.
  - for a rigid body, all the node frames are aligned with the body frame.
  - for a flexible body, the deformation of the body causes relative motion between the node and the body frames.

- **Actuator Nodes**

Point of application for applying external forces and moments on the body that can arise from actuators, environmental effects or other sources.

- **Sensor Nodes**

Point at which body information is obtained e.g. body attitude, rates

- **Hinges/Joints.** The kinematic constraints on the relative motion between adjacent bodies are modeled by hinges coupling the bodies.

- hinges can be translational, ball, full-6dof, slider; etc.
- each hinge has an associated pair of *hinge nodes*
  - *inboard hinge node* (Onode) located on the inboard body
  - *outboard hinge node* (Pnode) located on the outboard body.

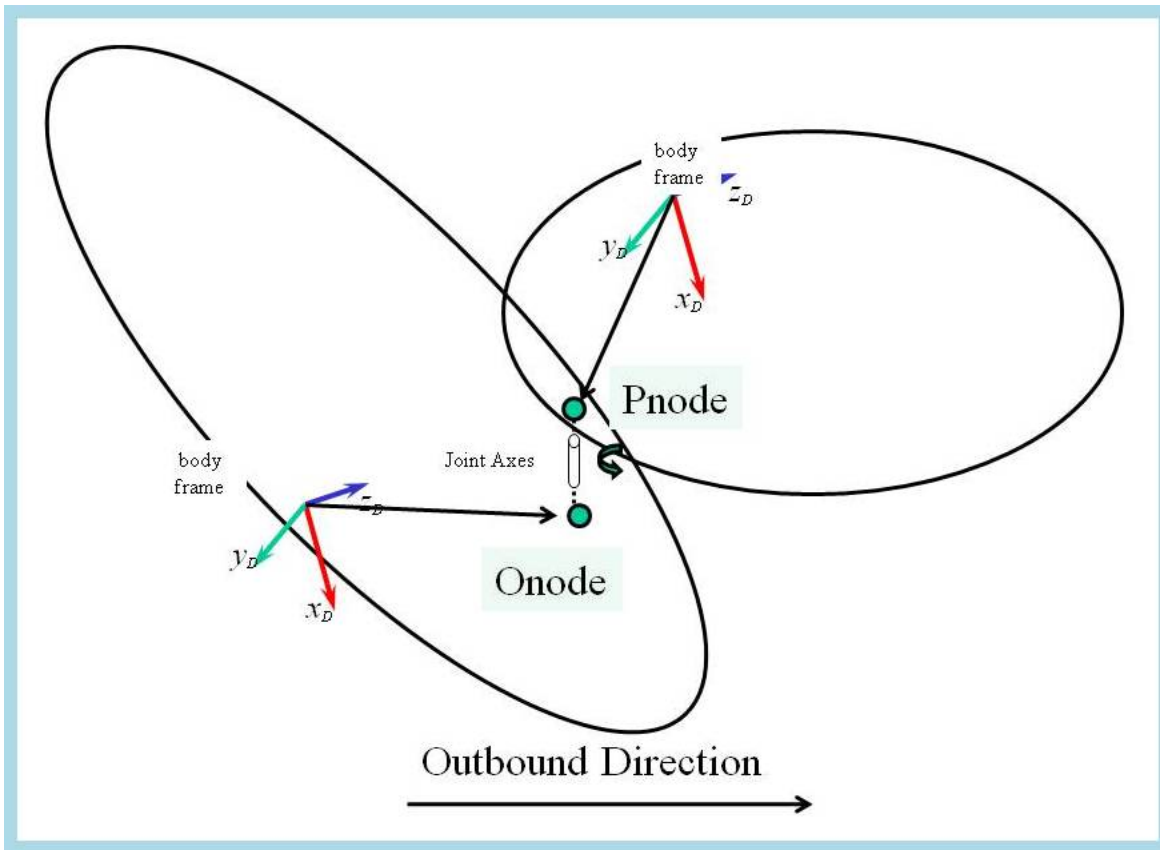


Figure 34. Multibody Hinges and Nodes

- the *generalized coordinates* for the hinge parameterizes the relative configuration of the hinge frames
- the *generalized velocity* for hinge parameterizes the relative spatial velocity between the frames.
- when the generalized coordinates of a hinge are the zero element, the *Onode* and *Pnode* frames coincide
- Historical note: the "O" and "P" nodes are in alphabetical order going from the base towards the leaf bodies. The "O" and "P" terminology goes back to the early days of Darts and uses the same nomenclature as the old multibody simulation software DISCOS. DISCOS (Dynamic Interaction Simulation Of Controls and Structure) was developed by H.P. Firsich at Goddard Space Flight Center. For further information about DISCOS, search for DISCOS in <http://naca.larc.nasa.gov/search.jsp>.

• **Zero Configuration.** Multibody configuration where all the generalized coordinates are the zero element.

- used to define all the kinematic and dynamics model information for the system in the model file e.g. vectors from the body frames to the joint node frame.

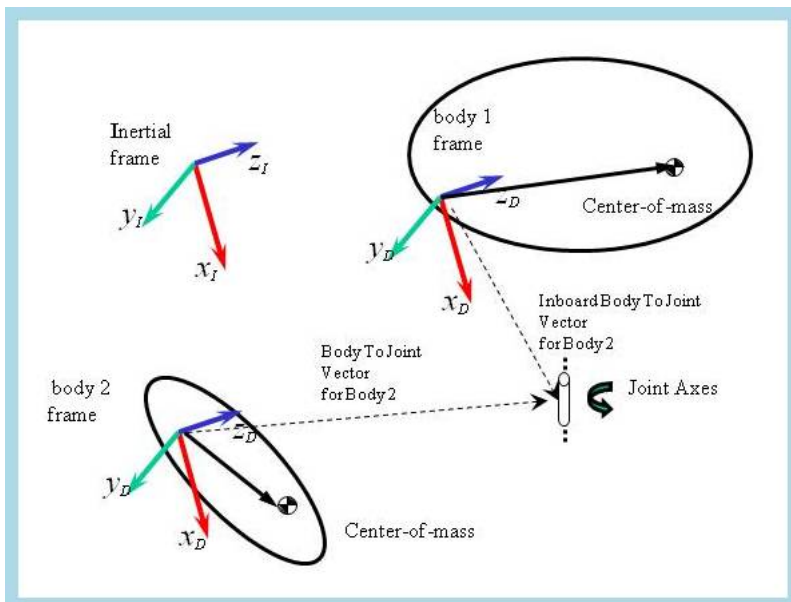


Figure 35. Multibody Zero Configuration

- In the zero configuration, the following terms are used to describe the geometry of the joint connecting a body to its parent body. Any of these terms may be zero.

**inbToJoint**

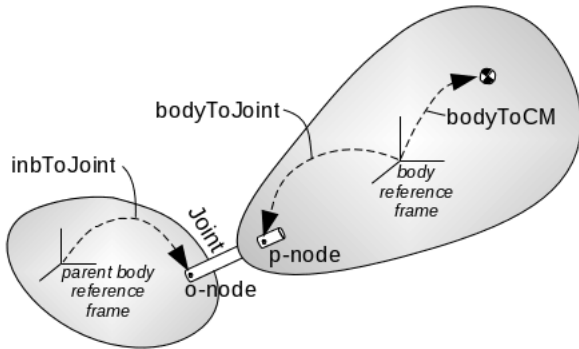
The vector from the parent body reference frame to the joint, expressed in the parent body frame.  
 Note that the rotation be be set with the **inbToJointQuat** and the full transform can be set with **inbToJointTransform**.

**bodyToJoint**

The vector from the child body reference frame to the joint, expressed in the child body frame.  
 Note that the rotation be be set with the **bodyToJointQuat** and the full transform can be set with **bodyToJointTransform**.

**bodyToCM**

The vector from the body reference frame to the center of mass for the body (in the body's reference frame).



o Note that all reference coordinate systems are parallel in the "Zero" configuration (unless **inbToJointQuat** or **bodyToJointQuat** are not the identity matrix).

• **System Generalized Coordinates and Velocities**

- o The *state vector* for a multibody system consists of a combination of:
  - the generalized coordinates vector, denoted **Q** consisting of
    - hinge coordinates of hinges connecting bodies to each other
    - modal coordinates related to the flexibility of a body
  - the corresponding generalized velocities vector, denoted **U**
- o The time derivative  $dQ/dt$  of **Q**, denoted **Qdot**, is obtained by a simple kinematic transformation of the generalized velocities vector **U**. For a joint with full rotational freedom, **Q** is modeled as a quaternion.
- o The combination of all the hinge generalized forces is designated as **Tjoint**
- o The combination of all the external forces acting on the multibody nodes is denoted as **Fext**
- o Darts computes  $dU/dt$  denoted as **Udot** which is then used by the Dshell integrator to compute **U**.

• **Kinematic Maps**

- o mapping from **U** to **Qdot** is usually a trivial identity map
- o exception: attitude dof generalized coordinates are the 4 elements of the attitude quaternion, while the 3 elements of the angular velocity vector are used for the generalized velocity coordinates. In this case, the mapping from the generalized velocities to the generalized coordinate rates is a non-linear, configuration dependent kinematic map.

• **Darts Prescribed and Non-Prescribed Hinges**

- o The Multibody hinge generalized coordinates state can belong to one of two categories:

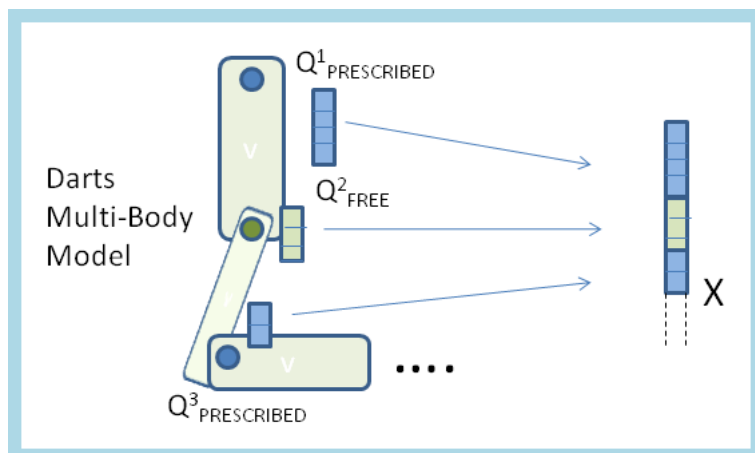


Figure 36. Multibody Prescribed and Nonprescribed States

- **Non-Prescribed Hinges.** The coordinates ( $Q_{\text{FREE}}$ ) associated with these hinges evolve in response to user or model specified **generalized forces** at the hinges connecting the bodies as well as input forces (and torques) on the multi-body system.
  - Represents the *Forward Dynamics* of the multi-body system.
- **Prescribed Hinges.** The coordinates ( $Q_{\text{PRESCRIBED}}$ ) associated with these hinges evolve in response to user or model specified **kinematic profiles** at the hinges connecting the bodies as well as the input forces (and torques) on the bodies in the multi-body system.
  - Represents the *Inverse Dynamics* of the multi-body system.
  - Prescribed motion hinges are often used to freeze articulated dofs, or when there are tight control loops governing the motion of the hinge dofs.

There is a related concept called "masking". For more detail about prescribing and masking, please see [prescribing](#).

• **Forward and Inverse Dynamics.**

- *Forward Dynamics* is solving for output accelerations ( $dv/dt$ ) given input forces ( $f$ )
- *Inverse Dynamics* is solving for output forces ( $f$ ) given input accelerations ( $a$ ) by using Newtons Law suitably generalized for articulated/flexible body dynamics.

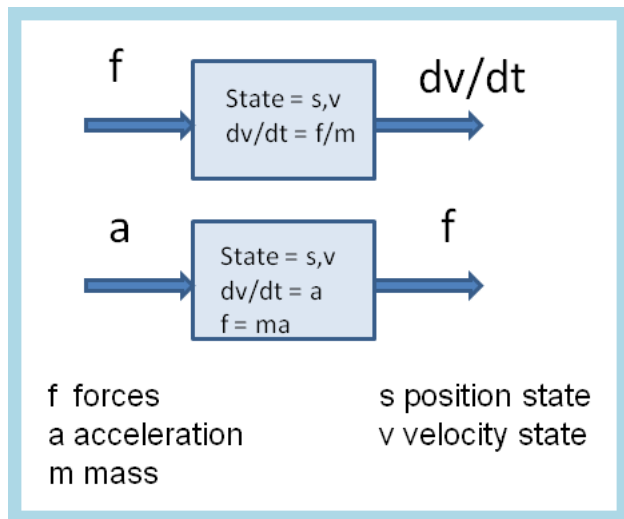


Figure 37. Forward and Inverse Dynamics

- In Darts *Forward Dynamics* involves computing the unknown hinge accelerations **Udot** from the generalized forces **Fext** and generalized hinge forces **Tjoint**
- In Darts *Inverse Dynamics* involves computing the generalized hinge force **Tjoint** from a *prescribed motion* of the generalized hinge **Udot** and the external generalized forces **Fext**

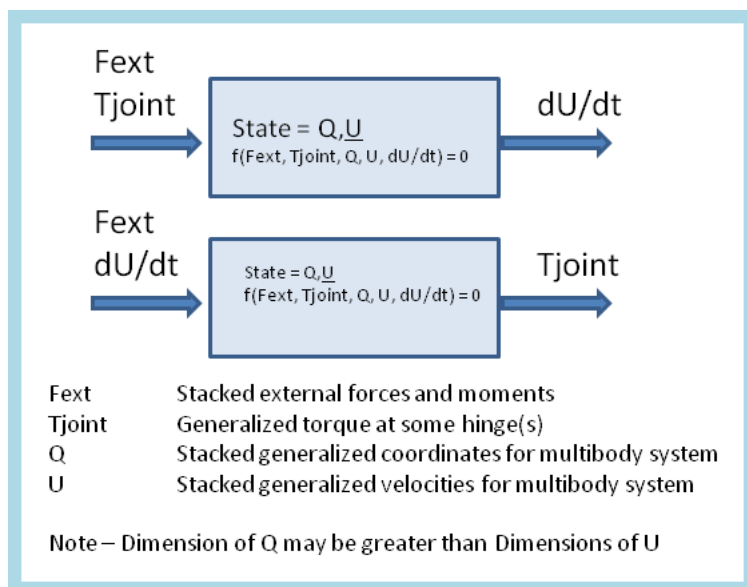


Figure 38. Darts Dynamics Engine

- Darts solves the general mixed forward/inverse dynamics problem i.e. where the hinges are a mix (respectively) of Non-Prescribed/Prescribed

6.2.1.4. Overview of DARTS Framework Objects

- The `DartsMbody` object

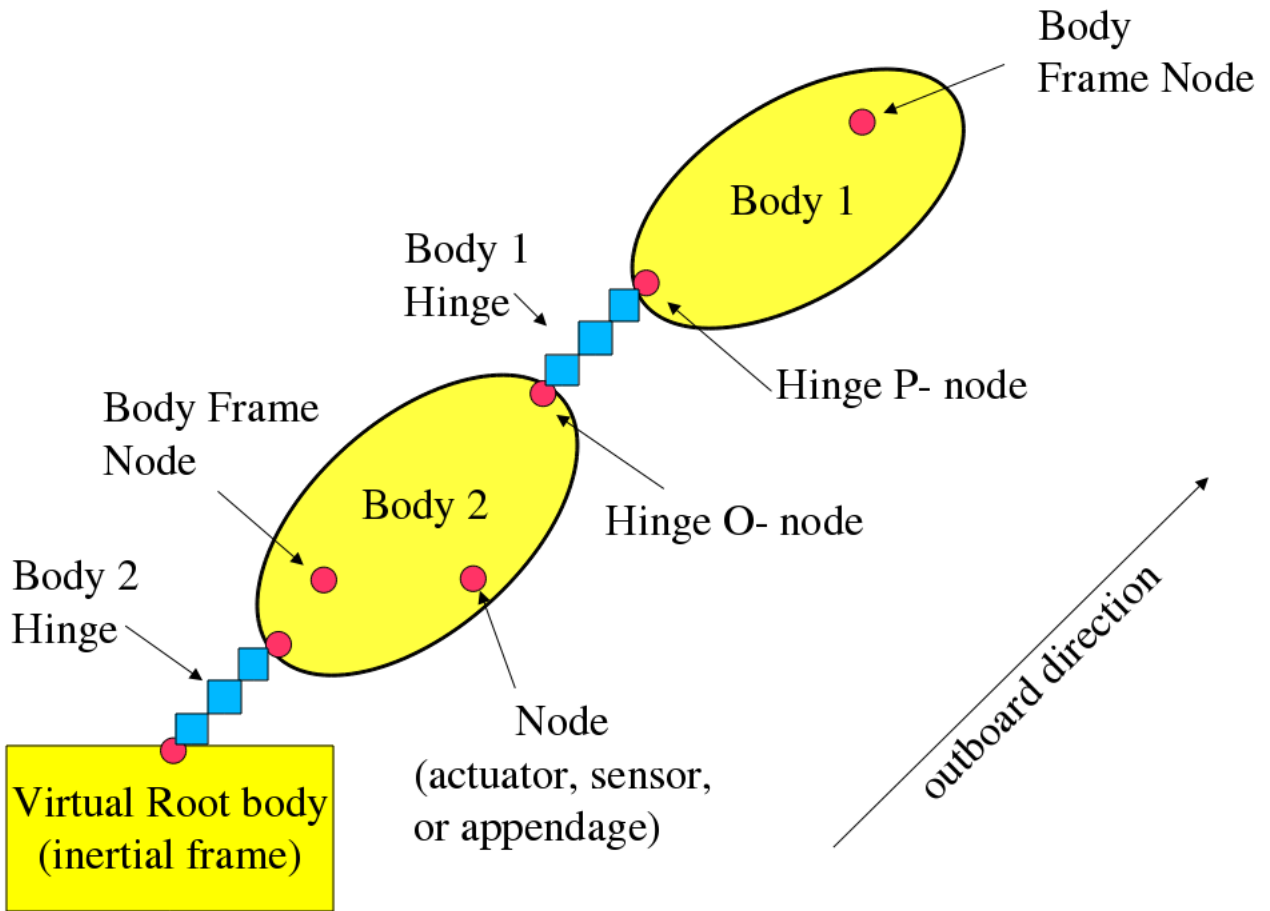


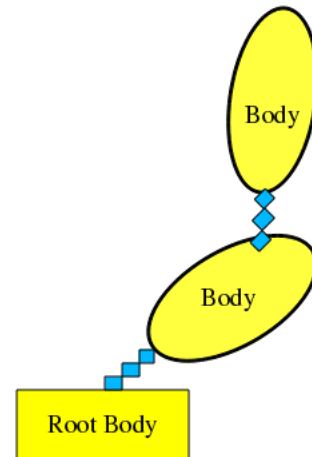
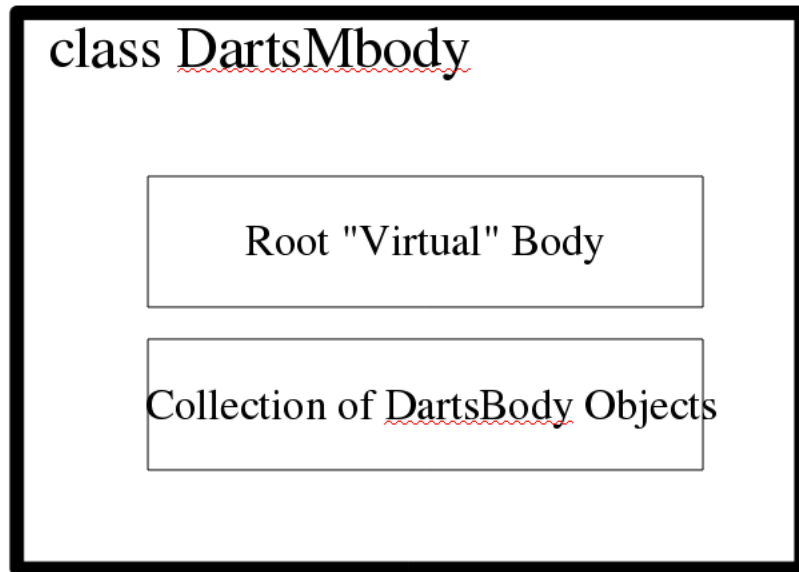
Figure 39. A Darts Mbody ("multibody") object consists of DartsBody objects.

Continued:

## DartsMbody Object

Manages a collection of DartsBody objects

The "root" is the head of the body list.



- 
- The DartsBody object



# DartsBody Object

Manages a linked list of DartsNode objects.

## class DartsBody

### Hinge

(connection between body and its parent body)

### Hinge Node

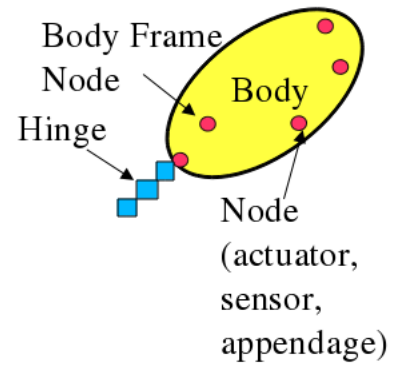
(node where the Hinge is attached)

### Body Frame Node

(node containing body frame properties)

### LinkedList of Nodes

(attachment pts for appendages, actuators, sensors)

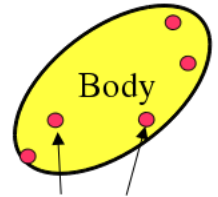
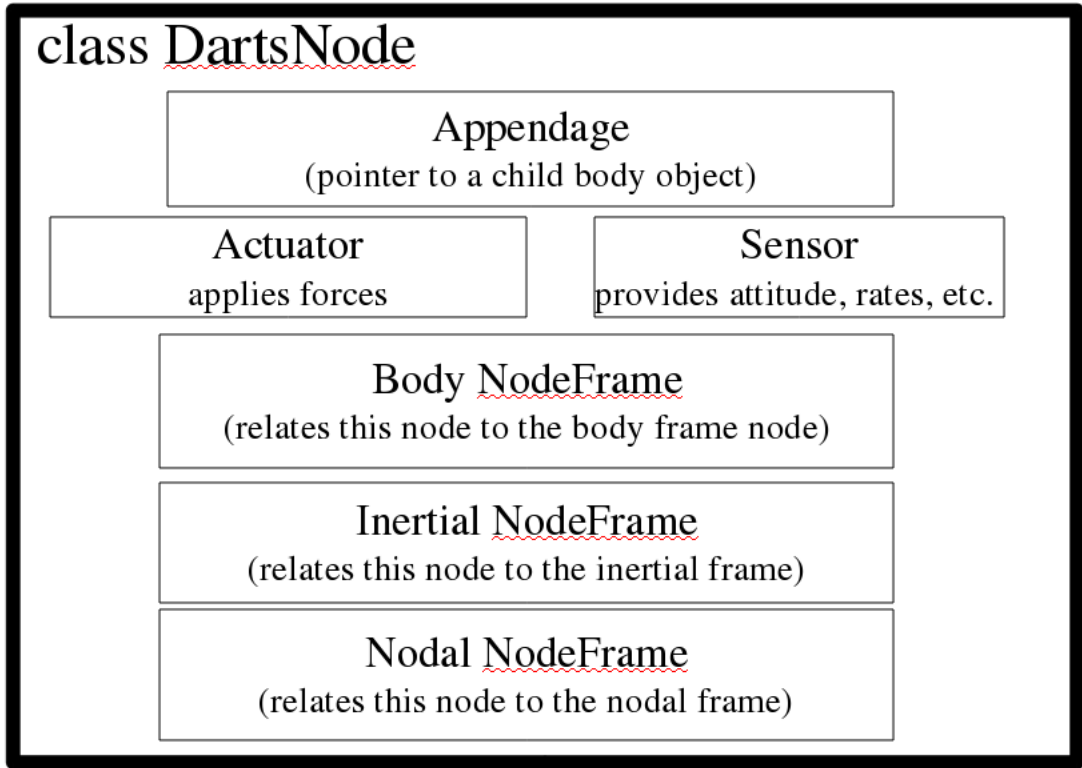


- The DartsNode object

# DartsNode Object

Represents a point on a body.

Holds the state properties of a point on a body.



Nodes  
(actuator,  
sensor,  
appendage)

- The DartsHinge object

## DartsHinge Object

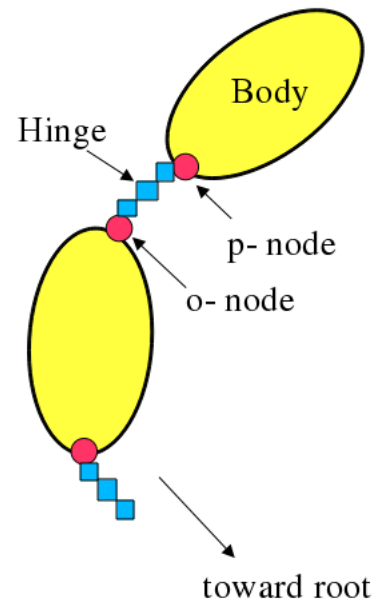
Models large angle relative motion between two connected bodies.  
Container for DartsSubhinge objects.

### class DartsHinge

Inboard Node (*o- node*)  
(pointer to attachment node on inboard body)

Hinge Node (*p- node*)  
(pointer to attachment node on hinge' s body)

Linked List of DartsSubhinge objects



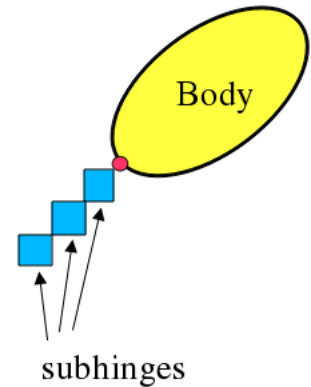
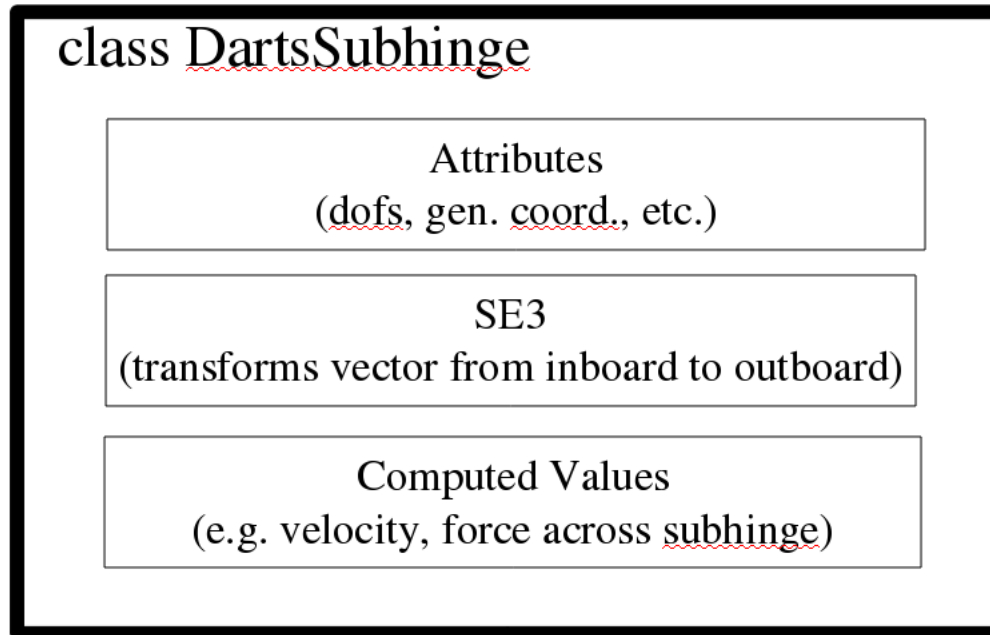
- The DartsSubhingeBase object

# DartsSubhinge Object

Basic hinge element.

Methods for computing the velocity/force across the subhinge are declared virtual.

New hinge types are introduced by subclassing the DartsSubhinge class.



## 6.2.1.5. Introduction

In Darts, hinges (or joints) are used to attach nodes to one another and provide degrees of freedom between attached nodes.

The Darts Hinge class is derived from the ChainedFrame2Frame class. Hinges are responsible for connecting two bodies together and have varying degrees of freedom depending on the type of hinge used.

Because nodes are implemented as frame objects, it is intuitive to implement the hinges that connect the nodes as Frame2Frame objects.

### 6.2.1.5.1. Hinge Types (Joint Types)

There are two main categories of hinges (or joints). Hinges are the actual objects that connect nodes together, whereas Subhinges are subparts from which hinges are built. All of the main hinge types can be broken down into "pin" and "slider" subhinge types, however there are several other three degree of freedom subhinge types for more complex hinges.

Here are a list of the basic hinge types with brief descriptions:

- PIN - a one axis rotational/revolute hinge (1 DOF)
- UJOINT - a universal (or Cardan) joint (2 DOF). See [https://en.wikipedia.org/wiki/Universal\\_joint](https://en.wikipedia.org/wiki/Universal_joint)
- GIMBAL - 1 gimbal with 3 rotational axes such as is typically used to mount a gyroscope (3 DOF). <https://en.wikipedia.org/wiki/Gimbal>
- BALL - a rotational joint with 3 degrees of freedom (3 DOF). The rotation is represented by a quaternion.
- SLIDER - a translational joint with one axis of translational freedom (1 DOF). Does not support any rotation. Sometimes called a prismatic joint.
- PLANAR - a translational joint with two axes of translational freedom (2 DOF). Does not support any rotation.
- TRANSLATIONAL - a translational joint with three axes of translational freedom (3 DOF). Does not support any rotation.
- COMPOSITE\_TRANSLATIONAL - a translational joint with three translational degrees of freedom. However, the three axes of translation are assumed to be the x-y-z axes (3 DOF). Does not support any rotation. In some situations a joint with both full translational and rotational degrees of freedom can be represented by a COMPOSITE\_TRANSLATIONAL joint followed by a BALL joint.
- FULL6DOF - a joint with full translational and rotational freedom (6 DOF). Joint rotations are represented by quaternions, but joint rates are represented by vectors of 3 values. In FULL6DOF joints, the attitude and velocity is represented in the child body frame.
- FULL6DOF\_INERTIAL - a joint with full translational and rotational freedom (6 DOF). Like in FULL6DOF joints, joint rotations are represented by quaternions, but joint rates are represented by 3-vectors. In FULL6DOF\_INERTIAL joints, the attitude and velocity is represented in the parent body frame (which is often the inertial frame, hence the name).

Other types joints can be made up of combinations of these types of joints. For instance a cylindrical joint, which as 1 translational DOF and 1 rotational DOF could be constructed by combining a SLIDER and a PIN joint.

Ndarts Prescribing and Masking Joints

### 6.2.2. Ndarts Prescribing and Masking Joints

In Ndarts, prescribing and masking a joint are completely independent concepts:

- **Prescribing** a joint controls how the Ndarts dynamics algorithms compute the torques and accelerations for the joint.
- **Masking** a joint hides (masks) selected hinge states for the joint from the Dshell integrator (e.g., for overriding). Usually Dshell models (and not the integrator) manage the values in these masked states. Masking controls whether joint position/velocity states come from the integrator (which is the normal case) or are set directly via a “deus ex machina” mechanism by the model (or user).



#### Note

Note that masking is related to the integrator and is not an Ndarts concept at all, but is covered together with prescribing because of their similarities.

For details about when to prescribe a joint and when to mask a joint, please see [Ndarts\\_Prescribing\\_Masking\\_Choice](#).

Prescribing Joints in Ndarts

#### 6.2.2.1. Prescribing Joints in Ndarts

In Ndarts, prescribing a joint controls how the dynamics engine computes the joint torques and accelerations.

##### 6.2.2.1.1. Unprescribed Joints

In normal operation when a joint is not prescribed, typically models will inject forces into the joint and Ndarts will compute the corresponding joint accelerations as shown in this figure:



Figure 40. Normal operation of joint (NOT prescribed)

In summary, **if a joint is not prescribed:**

- Joint forces/torques determine joint acceleration (using Ndarts)
- Joint velocity/position come from integration of joint acceleration (Udot)

##### 6.2.2.1.2. Prescribed Joints

If a joint is prescribed, some model (or the user) sets the joint acceleration and Ndarts will compute the necessary joint torque to achieve the prescribed acceleration as shown in this figure:

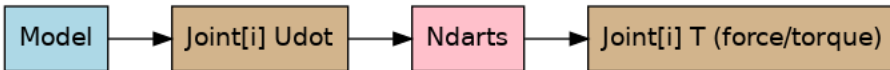


Figure 41. Effect of prescribing a joint

In summary, **if a joint is prescribed:**

- Joint acceleration determines joint forces/torques (using Ndarts)
- Joint velocity/position come from integration of joint acceleration (Udot)

##### 6.2.2.1.3. Setting the Joint Kinematics for a Prescribed Joint

The intent of prescribing a joint is for the models (or the user) to specify the joint acceleration and let the Ndarts dynamics compute the necessary joint torques. There is also an expectation that the integrator will integrate the given joint acceleration to compute joint velocities and joint positions. Note that the integration of the joint coordinates also depends on initial joint velocity and acceleration (which are usually set during initialization). There are several options for dealing with prescribed joints:

- **Specify joint acceleration at run-time.** Based on the initial joint velocity and position, the integrator will behave normally and produce smooth and consistent dynamics for the joint motion. The models or user can set the joint acceleration of a prescribed joint at any time during integration, including in the middle of integration steps using `preDeriv()` calls (in the same way that models can set joint torques at any time).

The code to set the joint acceleration looks like this in the C++ model code (note that `setGenAccel()` is a function of the C++ Model class, not the subhinge):

```
setGenAccel(hinge().subhinge(0), accel);
```

where the joint's velocity and position are initialized elsewhere.



#### Warning

Never set the joint acceleration directly through the joint subhinge API at run-time in models. Always use the model code call shown above since it checks the correctness of the call.

- **Specify the initial joint velocity and set the joint acceleration to zero.** In order to produce a prescribed motion that is constant velocity, the joint acceleration should be set to zero and the joint velocity should be initialized to the desired value. This might be useful for a model that sets up a constant-velocity ejection of one body from another. For example this code might be called in a model's init code:

```
setGenCoord(hinge().subhinge(0), initial_joint_position);
setGenVel(hinge().subhinge(0), desired_joint_velocity);
setGenAccel(hinge().subhinge(0), zero);
```



**Warning**

It is important to note that the `setGenVel()` function cannot be called in the middle of integration steps since it modifies the integrator state (the integrator must be reset). So updating the joint velocity cannot be done in `preDeriv()` calls. It may be called in the `startIntegrationStep()` or `endIntegrationStep()` functions.

- **Specify the initial joint position and set the joint velocity and acceleration to zero.** This might be useful to fix the position of a body with respect to its parent and determine the torques necessary to accomplish this. For example this code might be called in a model's initialization function:

```
setGenCoord(hinge().subhinge(0), initial_joint_position);
setGenVel(hinge().subhinge(0), zero);
setGenAccel(hinge().subhinge(0), zero);
```

6.2.2.1.4. Other Notes about Prescribing a Joint

- Prescribing a joint applies only to subhinges. Although a joint may be composed of several subhinges, prescribing occurs only at the subhinge level. There are some convenience functions in the body parameter code that allows setting a 'prescribed' for a composite joint to a single value (e.g., *True*), but the underlying code applies this same setting to all of the composite joint's subhinges.
- Setting the joint acceleration via `setGenAccel()` function calls in the model code can be done at any time with little computational impact. It is as efficient as setting the joint torque (for non-prescribed joints).
- Changing whether a joint is prescribed has very little overhead in Ndots since the size of the state vector is not changed and no unlocking/locking is necessary.

6.2.2.2. Masking Joints

In some cases, we want the joint to behave in a specific kinematic fashion irrespective of the integrator. In these cases, we can 'mask' the joint from the integrator, essentially overriding the integrator for these joint states as necessary to get the desired behavior.

There are two forms of *masking*:

- Joint masking in U (joint velocity)
- Joint masking in Q (joint position)

It is important to understand that masking is essentially a method of overriding the integrator and is not part of the dynamics model at all. This is why masking cannot be set while initializing the Darts body parameters.

Note that there is nothing exclusive to Ndots about the this concept. Masking also applies to Darts++.

6.2.2.2.1. Masking a Joint in Velocity (U)

When a joint is masked in joint velocity (U), some model in the simulation will override the joint velocity (U) state during normal execution. Here is what happens:

- The integrator ignores the joint acceleration (Udot)
- The joint acceleration (Udot) is as computed by Ndots.

Note that the overridden U and the Udot from Ndots are not generally consistent. The joint position comes from the regular integration of joint velocity Qdot.

This figure illustrates the basic concept:

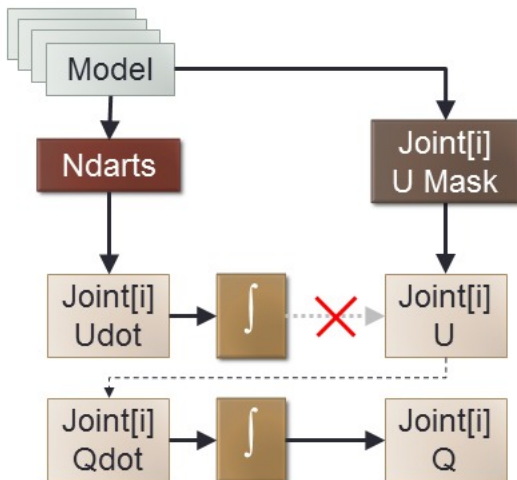


Figure 42. The effect of masking joint velocity (U)

Note that the model overrides the joint velocity. Therefore the joint acceleration (Udot) and joint velocity (U) will not generally be consistent. But the joint velocity specified by the masking process is integrated normally, therefore the joint velocity and position will be consistent with each other.

The function calls to set masking data for masking in joint velocity (U) looks something like this in the C++ model code:

```
coordMaskData(hinge().subhinge(0), joint_velocity, NULL);
```

The NULL argument above can be replaced with the desired joint acceleration (Udot) if it is known.

Note that coordMaskData() function calls are not part of the Darts hinge API and must be called via the C++ model API as shown.

#### 6.2.2.2.2. Masking a joint in Position (Q)

When a joint is masked in joint position (Q), some model in the simulation will override the joint position (Q) during normal execution. Here is what happens:

- The integrator ignores the joint velocity (Qdot)
- The joint acceleration is as computed by the Ndarts.

Note that the overridden Q and the Qdot are not necessarily consistent unless the model has used consistent values.

This figure illustrates the basic concept:

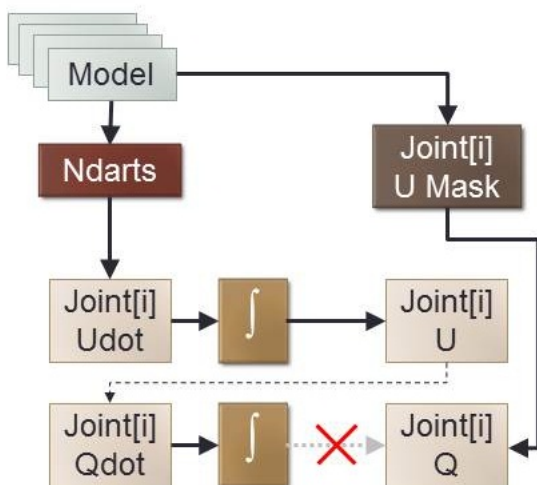


Figure 43. The effect of masking joint position (Q)

Note that the model overrides joint position. Therefore the joint velocity (Qdot), and the joint position (Q) will not generally be consistent and compatible.

The function calls to set masking data for masking in joint position (Q) looks something like this in the C++ model code:

```
coordMaskData(hinge().subhinge(0), joint_position, NULL);
```

The NULL argument above can be replaced with the desired joint acceleration (Udot) if it is known.

#### 6.2.2.2.3. Masking a joint in Position and Velocity (Q and U)

When a joint is masked in joint position and the joint velocity (Q and U), some model in the simulation will override the joint position (Q) and the joint velocity (U) during normal execution. Here is what happens:

- The integrator ignores the joint acceleration (Udot) and velocity (Qdot)
- The joint acceleration is as computed by the Ndarts.

Note that U and the Udot from Ndarts are not consistent. Similarly, the Q and the Qdot are not necessarily consistent unless the model has used consistent values.

This figure illustrates the basic concept:

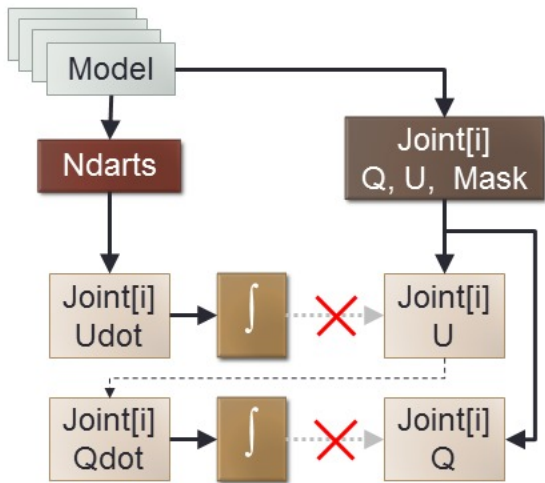


Figure 44. The effect of masking joint position and velocity ( $Q$  and  $U$ )

Note that the model overrides both the the joint velocity and position. Therefore the joint acceleration ( $Udot$ ), the joint velocity ( $U$ ), and the joint position ( $Q$ ) will not generally be consistent and compatible.

The function calls to set masking data for masking in joint position ( $Q$ ) and joint velocity ( $U$ ) looks something like this in the C++ model code:

```
coordMaskData(hinge().subhinge(0), joint_position, joint_velocity, NULL);
```

The NULL argument above can be replaced with the desired joint acceleration ( $Udot$ ) if it is known.

#### 6.2.2.2.4. The Masking Interface

Since masking is tied to integration, the API for changing joint masking is also tied to the integrator. There is a masking interface object that can be obtained from the simulation and masking in  $Q$  can be set up for a subhinge (in python) like this:

```
from Dshell.DartsCommon import DartsBody, DshellStateMaskingIF
sim.dynCoordMaskingInterface().coordMaskType(subhinge, DshellStateMaskingIF.MASK_Q)
```

where `sim` is the simulation object.

In C++ model code, comparable code should be put into the model's `setup0` function and might look like this:

```
coordMaskType(hinge().subhinge(0), DartsIF::MASK_Q);
```

#### 6.2.2.2.5. Other Notes about Masking a Joint

- All masking is done at the subhinge level.
- Changing the coordinate data for a masked joint is an efficient operation and can be done by models during normal execution without significant run-time penalty. This is normally done using a models 'coordMaskData()' function.
- Note that changing the coordinate mask data should only be done at integration sub-step boundaries and not in `preDeriv0` calls. The reason is that `coordMaskData0` calls essentially change the integrator state without the knowledge of the integrator. Typically, the `coordMaskData0` calls would be done in the model's `startIoStep0` or `updateFlowOuts0` functions.
- Changes in the masking state are actually requests that are accumulated and implemented right before the next integration sub-step. So calls to the `coordMaskType0` function can be done at any time, but will only take effect at the next integration sub-step.
- Changing the masking state of a joint involves more overhead and should not be done during normal integration since it requires that the integrator be reset and the state size changed after it is done.
- Currently the setting up the masking states of joints is done outside the model (e.g., in the assembly). But it makes sense to move this into the model code since the model is the only place where the masked joints can be updated.

#### 6.2.2.3. When to Prescribe a Joint and When To Mask a Joint

In a certain sense, both prescribing and masking a joint produce similar effects: they make the joint move like we want it to. When should you prescribe a joint and when should you mask it?

If all the following conditions apply, use 'prescribing':

- Smooth dynamics are desired
- Consistent joint kinematics (position, velocity, acceleration)
- Consistent dynamics and kinematics
- Only joint accelerations change during integration

Otherwise, use masking.



Here are some example scenarios:

- **Body Separation** - At some point a child body is detached from its parent body and a desired constant separation velocity is desired. Use **prescribing**. At the instant of separation, disconnect the bodies, set the desired separation velocity (via `setGenVel()` in model code) and set the desired joint acceleration to zero.
- **Trim control** - In many cases it is necessary to specify the trim angle for a body during flight to match some desired trim history (or via some state-dependent way). Since the only concern is the body attitude and not the necessary torques to produce it, there is no need for the dynamics and kinematics for the body attitude to be consistent. Therefore **masking** is appropriate.
- **Acceleration Specified** - If the desired joint acceleration is given and there are no other limitations, then **prescribing** should be used.

#### 6.2.2.4. Creating shapes for your Ndots bodies

It is straight-forward to add graphics to your simulations by added PartGeometry data in the 'geometry' definition of your body parameters.

When you define body parameters using the BodyParam class, you can add in a series of geometrical shapes by specifying a 'geometry' section:

```
from DshellCommon.params.BodyParam import BodyParam
...
'SC' : BodyParam(
    mass = 23.0,
    ...
    geometry = {
        'body' : {
            'shape' : 'cylinder',
            'radius' : 1.0,
            'height' : 3.0,
            'scale' : (1, 1, 1),
            'translation' : (0, 0, 0),
            'quaternion' : (0, 0, 0, 1),
            'diffusivity' : (0.6, 0.1, 0.1)
        },
        'body2' : {
            'shape' : 'cube',
            'length' : 0.2,
            'width' : 0.15,
            'height' : 0.15,
            'diffusivity' : [0.35, 0.30, 0.50],
            'scale' : (1, 1, 1),
            'translation' : (0, 0, 0),
            'quaternion' : (0, 0, 0, 1)
        },
        'body3' : {
            'file' : 'foo.mesh',
            'scale' : (1, 1, 1),
            'translation' : (0, 0, 0),
            'quaternion' : (0, 0, 0, 1)
        }
    }
)
```

where some parts have been omitted for brevity. Notice that multiple geometrical parts can be given. The name of the graphical object is its key (eg, `body` or `body2` here). Each graphics object is a fixed distance and rotation from the body reference frame (as specified in the `translation` and `quaternion` parameters).

#### Example 1. Danger

If you set one of the below parameters and you don't set the associated parameters, the program segfaults for some reason. For example, this happens when `cylinder` is set, but `height` and `radius` is not set.

The types of common shapes that can be specified (including their required parameters) are:

- **sphere**
  - radius
- **cylinder**
  - radius (x and z)
  - height (y)
- **cone**
  - radius (x and z)
  - height (y)
- **cube**
  - length (x)
  - width (y)
  - height (z)
- **file**

To specify a mesh file instead of a primitive shape, use the `file` parameter instead of the `shape` parameter (as shown for `body3` above). The `file` parameter takes a mesh filename as its value. When using `file`, you may not be able to modify the color of the object as that is usually embedded in the mesh file. See `Ndarts_ShapeMeshFileFormats` below for more details about the supported file formats.

All of these graphics objects accept the following optional parameters:

- **scale** - a tuple/list of 3 values describing how to scale the:: object in the x, y, and z directions (body reference frame).
- **translation** - a tuple/list of 3 values (dx, dy, dz) of the:: offset between the body reference frame and the center of the graphics object.
- **quaternion** - a tuple/list of 4 values of the quaternion for the:: rotation between the body reference frame and the graphics object.
- **diffusivity** - a tuple/list of three values (0 to 1.0) for the:: Red, Green, and Blue color of the object. Note that this does not apply for meshes (which must provide their own coloring).
- **specularity** - a tuple/list of three values (0 to 1.0) for the:: Red, Green, and Blue specularity (shininess) of the object. Note that this does not apply for meshes (which must provide their own coloring).
- **emissivity** - a tuple/list of three values (0 to 1.0) for the:: Red, Green, and Blue emissive color of the object. Note that this does not apply for meshes (which must provide their own coloring).
- **transparency** - a single value (0 to 1.0) describing the:: transparency of the graphics object (1.0 is completely transparent, 0 is not transparent at all).
- **visibility** - a single value (0 or 1) indicating whether this:: graphics object should be visible initially.

#### 6.2.2.4.1. Mesh File Formats

The `file` parameters can be an OGRE `.mesh` file or a Wavefront `.obj` file. If you want other types, use a Python file to point to them. For example `example.py`:

```
{'wavefront': 'foo.obj',  
'ogre': 'foo.mesh'}
```

Then you would set `file` to `example.py`.

Part geometries support a few other 3D graphics file formats by using the `assimp` library to load the 3D graphics data from mesh files. (See [http://www.assimp.org/main\\_features\\_formats.html](http://www.assimp.org/main_features_formats.html)). The currently supported files are:

- **OGRE**, use:
  - `'ogre': '<filename>.mesh'`
- **Collada**, use:
  - `'collada': '<filename>.dae'` *or*
  - `'collada': '<filename>.xml'`
- **Wavefront Object**, use:
  - `'wavefront': '<filename>.obj'`
- **3ds Max 3DS**, use:
  - `'3ds': '<filename>.3ds'`

If you have other file formats, the `assimp` tool can be used to convert to one of these supported formats. See <http://www.assimp.org> for more information.

#### 6.2.2.5. Manipulating a graphical object at runtime

You may manipulate graphical objects at run time. First get its handle. Suppose `sc` is the body object that these graphics objects are attached to.

```
from Math.SOA_Py import SOAVector3  
...  
cyl = sc.getPartGeometry('body')  
cyl.scale(SOAVector3(0.5, 0.5, 0.5))
```

PYTHON

This rescales the size of the graphics object to be half what it normally is (with `scale=(1,1,1)`).

Most of the other properties outlined above may be modified (the names may be a bit different; browse a live object to find what you need).

You may also active the axes for a graphical object:

```
cyl.showAxes(True)
```

PYTHON

#### 6.2.2.6. Other documentation

For Doxygen documentation, see: `PartGeometry`.

### 6.2.3. Multibody Modeling with Ndarts - Design

The Ndarts library provides classes and algorithms for the kinematics and dynamics of rigid multibody systems for use within the Dshell simulation framework. It is a successor to the Darts++ module.

#### 6.2.3.1. Background

The objective of this new multibody kinematics/dynamics implementation is motivated by several issues with the current "Darts++" implementation, as well as a need and desire to add several new advanced capabilities. The limitations in the current capabilities includes:

- Inverse kinematics, Constraints
  - A need to support new wheel geometries and contact in rover sims (broader wheels, toroidal shape, etc.)
  - The IK implementation with IKGraph is complex (eg. uses Euler coordinates). The NRSolve code is dense and hard to use in its current form.
  - Hard to debug cause when inverse kinematics failed - and so hard to fix issues. Lots of struggles with Chariot wishbone kinematics. Have been tossing around ad-hoc and unsatisfactory work around solutions - weights, staged IK, etc.
  - Need to be able to expand configuration kinematics to allow new constraints on motion (eg. Athlete cases)
  - Ability to handle constraints between multiple vehicles (eg. coordinated motion, docking)
  - Lack of general way to handle constrained/closed-chain dynamics
  - Questions about proper handling of hinge/constraint duality to simplify addition of new constraint types.
  - Unable to do IK on subgraphs within multibody.
  - Hard to bring in contact and collision detection physics.
  - Challenging to "generalize" constraint embedding to handle arbitrary "gearing" and "loop" type constraints (limited to four bar linkage currently)
- Functionality Limitations
  - Accumulation of narrow/ad-hoc algorithmic implementations that are hard to generalize (eg. special handling of full 6dof hinges, topology changes, handling constraints)
  - Hard to add new hinge types (eg. locked, alternate 6dof) hinges - lots of switches and if/thens in code.
  - More complex than necessary multibody creation process - split between Python and C++.
  - Not possible to make arbitrary frame to frame relative transformation, velocity, acceleration queries within multibody system (needed more generally but especially for constraints)
  - Hard to bring in PyCraft like ideas into regular dynamics functionality
- Architectural and Design Integrity
  - Increased entropy in the Darts++ software implementation from changes/hacks that have been made over the years.
  - While the test coverage has improved, the coverage is uneven, making it harder to add and test new changes.
  - While code "works", it is hard to add in new features without endangering existing capabilities.
  - User level methods and internal code Documentation quality is poor.
  - Method naming conventions are uneven.
  - Body and node methods have parallel methods which are named and/or do something differently.
  - Darts++ code split across DartsBase module for backwards compatibility with Darts

The goal is to rethink, redesign, and streamline the current implementation to have a more agile architecture that will address the above, enabling some of the more advanced following capabilities:

- Advanced mbody algorithms (PyCraft ideas)
  - Allow the implementation of new algorithms (eg. innovations, velocity coordinates transformations, and inverse)
  - Support recursions on sub-trees for embedded constraints and augmented bodies, onboard models, ODCA/parallel implementation
  - Support recursions involving different types of propagation operators (eg. rigid bodies, flex bodies)
  - Support multiple operations/recursions on a fragment, and have means to store and keep access to data products so they can be reused in subsequent recursions (eg. correction recursions in augmented constraints implementation, momentum matching, ODCA decomposition and correction, ATBI recursions)
  - Support recursions for state initialization for momentum matching after impact; or for CM drift nulling for MD simulations
  - Support Pycraft algorithms for sensitivities, Coriolis, etc.
- Constraint/collision/contact handling
  - Support configuration changes in the multibody system, i.e. constraints coming and going (eg. for humanoid application, docking, walking)
  - Support addition of body geometry and surface contact constraints
  - Support use in interactive computations for inverse kinematics; closed-chain state projection
- Miscellany
  - Support "internal model control (IMC)" type algorithms for computations within onboard control software.
  - Support body reordering and reversal of bodies (for optimization of closure cuts)
  - Support using alternative integration variables such as for diagonalized dynamics. May need to generalize Qdot to U and converse functions (already doing this for CK?)
  - Support being able to switch between different frame of integration (eg. inter-planetary trajectory, to EDL trajectory, to rover trajectory)
  - Support use of different frame (body, inertial, internally referenced) for equations of motion.

### 6.2.3.2. Ndarts Design Details

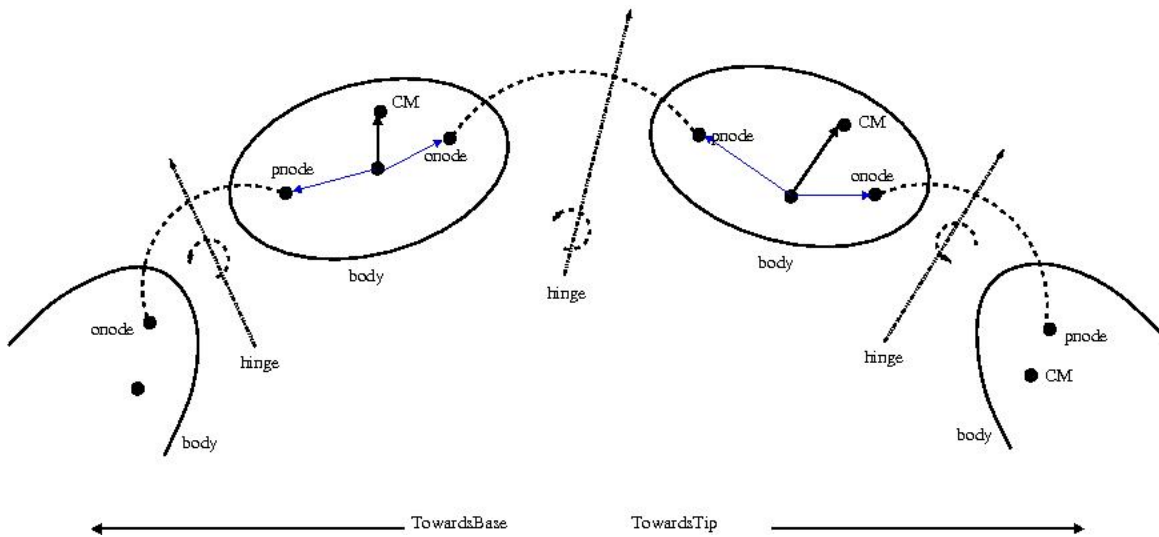


Figure 45. Links and hinges in a multibody system

#### 6.2.3.2.1. Key Requirements

- Code structuring
  - Streamline higher level algorithms by moving out lower level computations into lower level classes so can bring in new algorithms.
  - Keep user level API changes to a minimum.
  - Give low priority to allowing Dshell to work with Darts++ and Ndarts at the same time to avoid constraining design space.
- Kinematics
  - Use the SOAFrames "Frames" class from the "SOA" module to simplify the kinematics computations.
  - Allow automatic queries for relative transformations, velocities, and accelerations across arbitrary points of interest.
  - Avoid unnecessary kinematics computations and overhead.
- Hinges
  - Add support for locked hinge types.
  - Simplify addition of new hinge types.
  - Support run-time user defined hinge types.
- Constraints
  - Support constraint embedding.
  - Support handling of closure constraints.
  - Support handling of surface contact constraints.
- Generalize inverse kinematics to handle much broader class of requirements.
- Support working with system subgraphics.

#### 6.2.3.2.2. Obsolete Modules

- IKGraph
- DartsBase
- TerrainSurface
- SurfaceContact

#### 6.2.3.2.3. API and Functionality Changes

- SOA
  - Changed and renamed all SOAHomTran phi methods
  - Added support for 0 size matrices and vectors
  - Now Frame and Frame2Frame classes are derived from DartsBaseObject
  - Reworked Frame2Frame cache management to allow transform/velocity and acceleration level cache staleness settings
  - Fixed up the matrix SVD and orthogonal complement methods
- Removed and renamed other node, body, hinge methods.
- Moved DartsBaseObject class code to DshellEnv
- Class refactoring
  - Stopped using most base classes from DartsBase
  - Added hinge classes.

- Derive subhinge and higne classes from Frame2Frame.
- New 6dof hinge.
- Derive Node class from Frame.
- DartsBody is now derived from DartsNode class
  - Hence no longer have parallel set of methods for position/velocity of nodes and frames
- DartsHinge only requires frame pair - not node pair to allow use with constraints
- Kinematics
  - evalKinematics, evalActuatorKinematics, evalSensorKinematics methods etc. have been removed.
  - Now do lazy evaluation with caching for transforms, velocities, and accelerations within Frame classes.
  - Extensive reliance on Frames classes.
  - Consolidate DartsBody and DartsNode methods to get position, velocities, etc. In the process some of the methods have been renamed.
  - Changing prescribed flag of a subhinge should only be done when system is unlocked (open to discussion) since constraint related buffer sizes are effected.
  - Added notion of minGenCoord to hinges.
- Added new DShape class for body geometry
- Cosntraints
  - Complete reworking of constraint handling and inverse kinematics.
  - Obsoleted use of IKGraph. Changed IK scheme to more robust rechnique.
  - Added FramePair constraint class to handle any hinge based constraint.
  - Added support for closure and surface shape to surface shape contact constraints.
  - Now surface to Dem constraints is handled same way.
  - Activating/deactivating constraints can only be done when system is unlocked.
  - Inverse kinematics now leaves prescribed motion hinges unchanged.
- Jacobians and Gradients
  - Now can get Jacobian between any frame pair
  - Transform gradient methods to complement Jacobian ones.
  - Now Jacobian methods take extra arguments for skipping prescribed subhinges, and for being maps from generalized velocities of Qdot.
  - Concept of min gen coords for gradients.
- Stricter use of locks
- Now need to unlock to change prescribed status as well as active status of constraints.
- Global vector offsets are tracked for subhinges and constraint hinges.
- Added DartsSubGraph class for a subset of connected bodies.

#### 6.2.3.2.4. Class Hierarchy

- Derive Frame and Frame2Frame from DartsBaseObject class
- Derive DartsNode from Frame class
  - DartsHingePnode (new)
  - DartsHingeOnode (new)
  - DartsActuatorNode (new)
  - DartsSensorNode (new)
  - etc.
- Derive DartsBody from DartsNode class

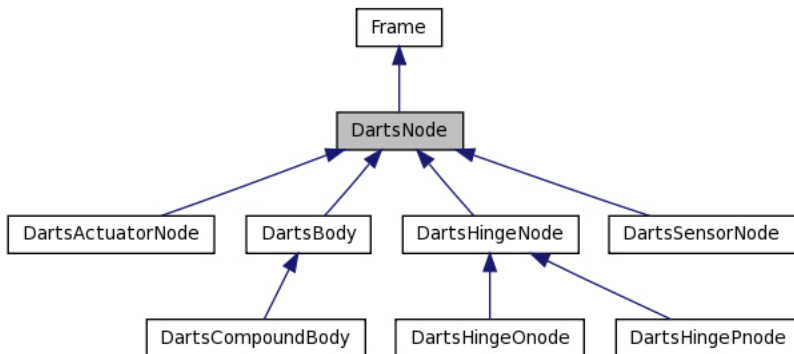


Figure 46. DartsNode classes

- Derive DartsSubhinge from Frame2FrameEdge class
  - DartsLockedSubhinge (new)
  - DartsPinSubhinge
  - DartsLinearSubhinge
  - DartsSphericalSubhinge
  - DartsLinear3Subhinge
  - etc.
- Derive DartsHinge from Frame2FrameChain class
  - DartsPinHinge (new)
  - DartsSliderHinge (new)
  - DartsBallHinge (new)
  - DartsTranslationalHinge (new)
  - DartsFull6DofHinge (new)
  - DartsCustomHinge (new)

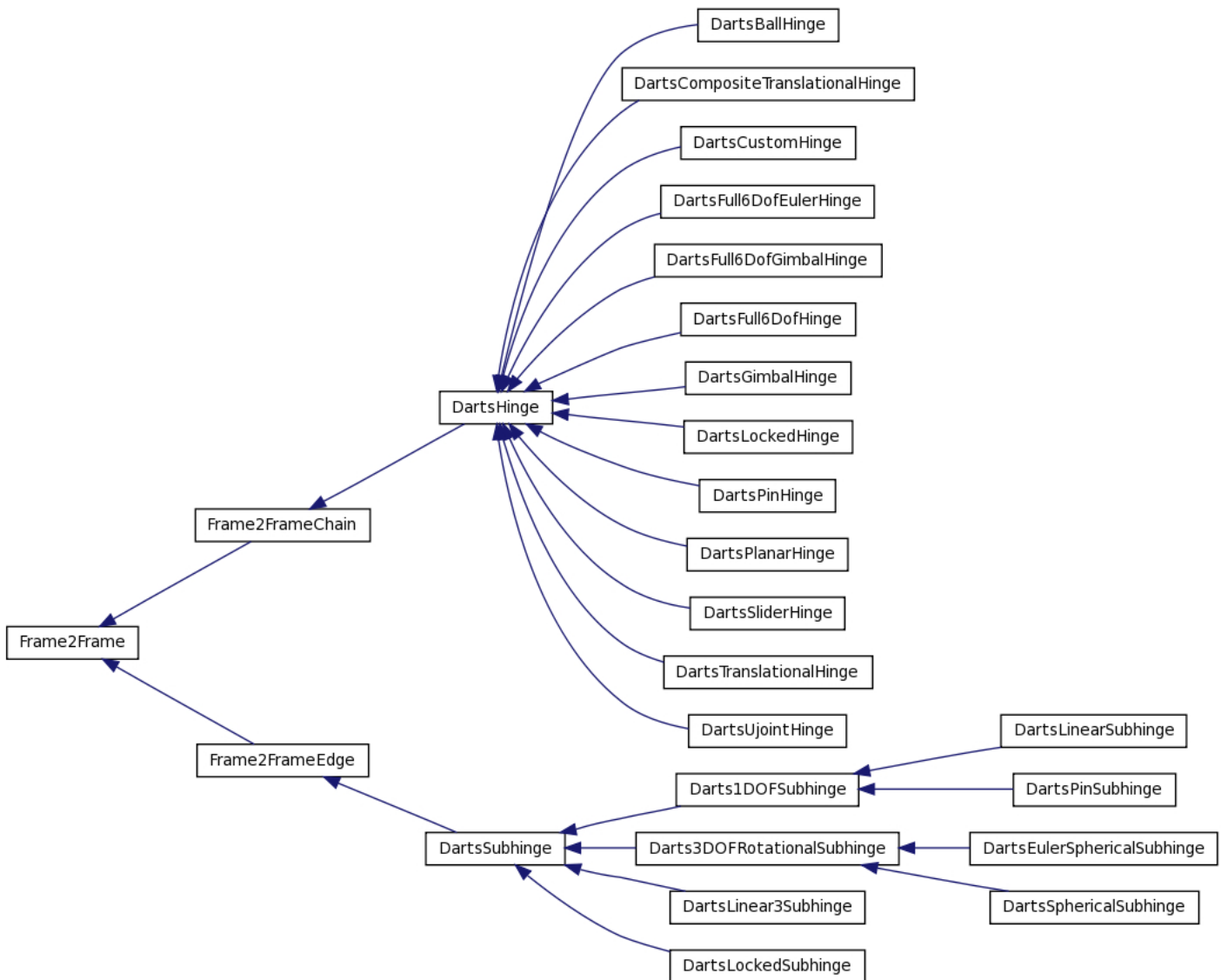


Figure 47. Hinge and subhinge classes

#### 6.2.3.2.5. Body Creation Process

- The root body's frame is connected to the global root frame
- The steps for the creation of a body include
  - Create the body and body frame. Temporarily make the body frame the child of the root frame
  - Create a pnode for the body and (temporarily) make the pnode frame a child of the body frame. Delete the body frame to pnode frame edge and make the body frame a child of the pnode frame. The pnode frame at this point is dangling and not attached to the frames tree.

- Create a 6dof hinge connecting the root body to the new body.
  - Detach the pnode frame from the body frame if it is attached to the tree (so it can be used as a pframe for the sub-hinges).
  - Detach the body frame from its current parent frame if the current parent is not the pnode frame since it needs to be a child of the pnode frame.
  - Create an onode for the body in the root body. Its frame is a child of the root body's frame.
  - Create a sequence of sub-hinges that go from the onode and end at the pnode. At this point, the pnode frame is connected back to the frames tree.
  - Connect the body frame as a child of the pnode frame. Now the body frame is connected to the frames tree.
- Now create the proper joint type by deleting the existing hinge and creating one specified by the joint type and connected to correct inboard body.
  - Delete the existing hinge by deleting all the subhinges and the onode. The subhinge deletion will delete all the frame edges and intermediate oframes. It will leave the pnode frame unattached to the frame tree.
  - Create a new hinge of the correct type between the new parent and the body.

### 6.2.3.3. Body Shape Classes

Each DShape shape represents a parameterized geometry object. Can add a number of geometry objects to a DartsBody instance. Each shape can have its own SOAHomTran offset transform. Usage of DShape classes include:

- Contact kinematics and constraints
- partGraphics
- DBullet interface
- TopoDem terrain vehicle placement, configuration kinematics.
- Imposing task space motion constraints (manipulator, vehicle)
- General mesh surface (eg. rocks, VRML parts) contact.

Non-penetrating contact between shapes requires that the tangent planes at the points of contact be parallel. The dimensionality of a DShape instance is defined by the number of parameters needed to specify the contact point on the surface. The following shape classes are currently defined:

- DShapePoint : 0-dimensional point.
- DShape1D : 1-dimensional surfaces
  - DShapeStraightLine : Linear straight line.
  - DShape1DCircle : Thin circular disc.
  - DShape1DEllipse : Thin ellipse
- DShape2D : 2-dimensional surfaces
  - DShape2DPlane : 2D plane
  - DShape2DSphere : Spherical surface
  - DShape2DEllipsoid : A general ellipsoid
  - DShape2DTorus : A torus
  - DShapeDem : TopoDem surface
  - DShapeMesh : General mesh surface (BD)
- DShape3D : 3-dimensional surfaces with overhangs (TBD)

DShape classes have a DShape::partGraphics Python method to return a part graphics dictionary appropriate for the shape type.

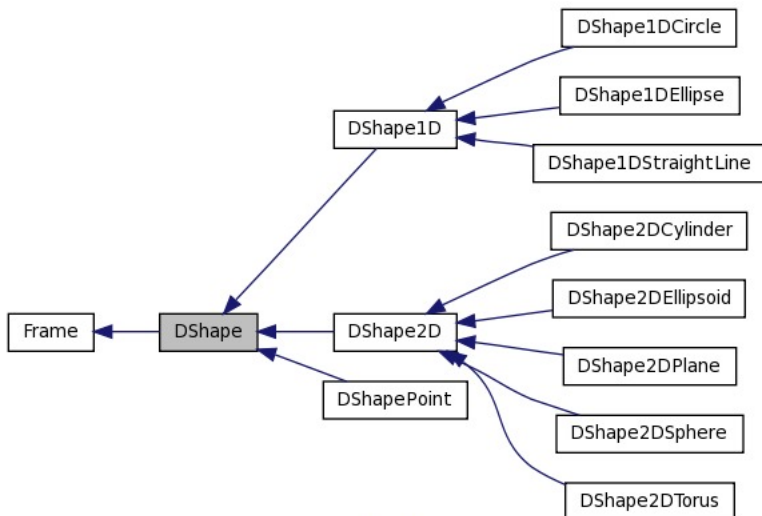


Figure 48. DShape body geometry classes

### 6.2.3.4. Constraints

Constraint classes are used to enforce kinematics and dynamics constraints on the articulation and motion of the multibody system. The current family of constraint classes is as follows:

- DartsFramePairConstraint : Base class for frame to frame constraints.
- DartsClosureConstraint : Limits motion between a pair of frame defined by a DartsHinge type.
  - Single Node to Frame constraint between a multibody node and an arbitrary Frame (moving or fixed).
  - Node pair constraint between a pair of multibody nodes.
- DartsContactConstraint : Constraint between a multibody node and a DShape surface.
- Vehicle configuration kinematics for system level constraints on vehicle/manipulator placement and motion
  - Place rover
  - KDriver constraint
  - Driving Athlete with level hex
  - Driving Athlete to come to dock
  - Driving 2 Athletes in docked configuration
- Constraint embedding
  - General loop constraints
    - Four bar linkage/wishbone constraint
  - GenCoordsConstraint - algebraic constraint between arbitrary gen coord values (TBD)
    - For local constraints (gearing, differential), use constraint embedding
    - More generally need to handle this
    - Treat prescribed motion as a special case of this constraint on a single hinge
    - Examples
      - Prescribed motion constraint (or joint locking)
      - ATRV driven wheels constraint
      - Differential constraint
      - Gearing constraint

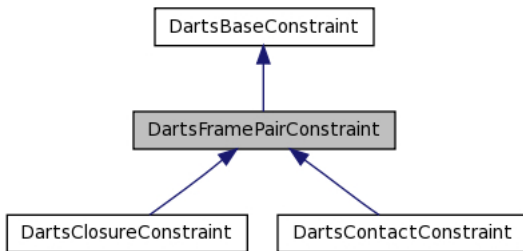


Figure 49. Constraint classes

#### 6.2.3.4.1. Node to Fixed frame Constraint

SingleNode constraint - one node is on a multibody system and the other Frame is not on a multibody (but could be a moving frame). The constraint is expressed as a hinge specifying the allowable motion in the constraint. Enforces constraint relations such as:

$${}^I T_{ec}(\theta) * {}^{ee} T_c(\theta) * T_{offsets} = {}^I T_{tgt}$$

- This constraint takes a node, frame, and hinge as arguments.
- It is simpler than the NodePair constraint in that the Jacobian of only one node needs to be computed.
- Examples
  - Articulate antenna to be pointing towards a moving s/x or the sun
  - Arm inverse kinematics
  - Move arm so the end effector meets a distance/attitude constraint to a desired task frame.

#### 6.2.3.4.2. Node to Node Constraint (dual arm)

NodePair constraints - where both nodes are on multibody systems and there is a hinge specifying the constraint between them. Enforces constraint relations such as:

$${}^I T_{ec}(\theta) * {}^{ee} T_c(\theta) * T_{offsets} = {}^I T_{tgt}(\theta)$$

- This constraint takes a node pair and a hinge as arguments
- Examples
  - Place/move rovers in formation offset by a fixed transform
  - Move arm so its end-effector is a certain distance from a wheel



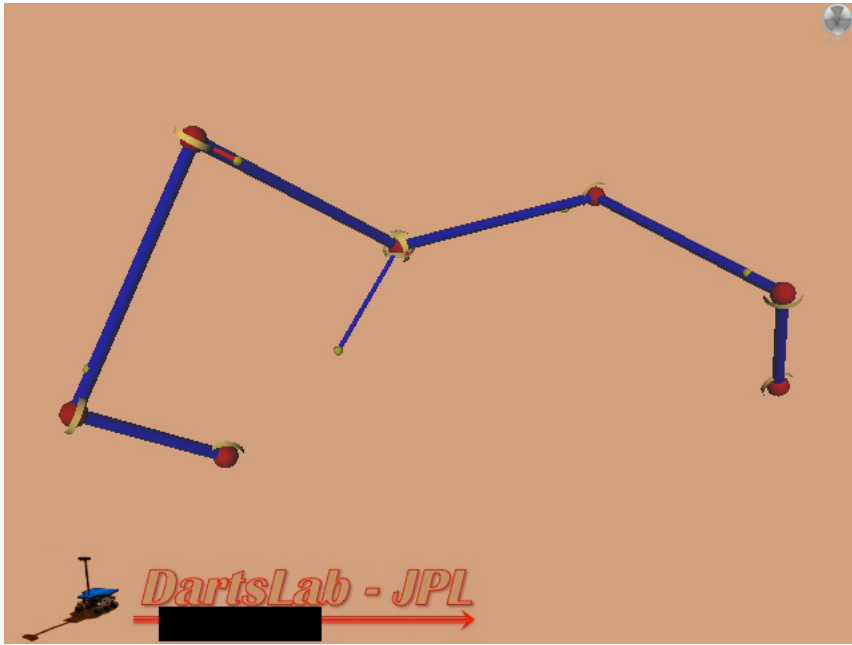


Figure 50. Example of a node to node constraint on dual-arm end-effectors

#### 6.2.3.4.3. Contact constraints

Constraints between a DShape (0, 1, or 2D) and a DShape2D 2D surface

- For a 0d surface, the constraint is that the distance between the surfaces is zero.
- For a 1d surface, the constraint is that the distance between the surfaces be zero, and at a point of contact, the 1d tangent be orthogonal to the 2D surface normal.
- For a 2d surface, the constraint is that the distance between the surfaces be zero, and at the point of contact, the tangent planes for both surfaces be parallel.

The following describes examples of different surface to surface contact constraint

##### 6.2.3.4.3.1. Point 0d contact

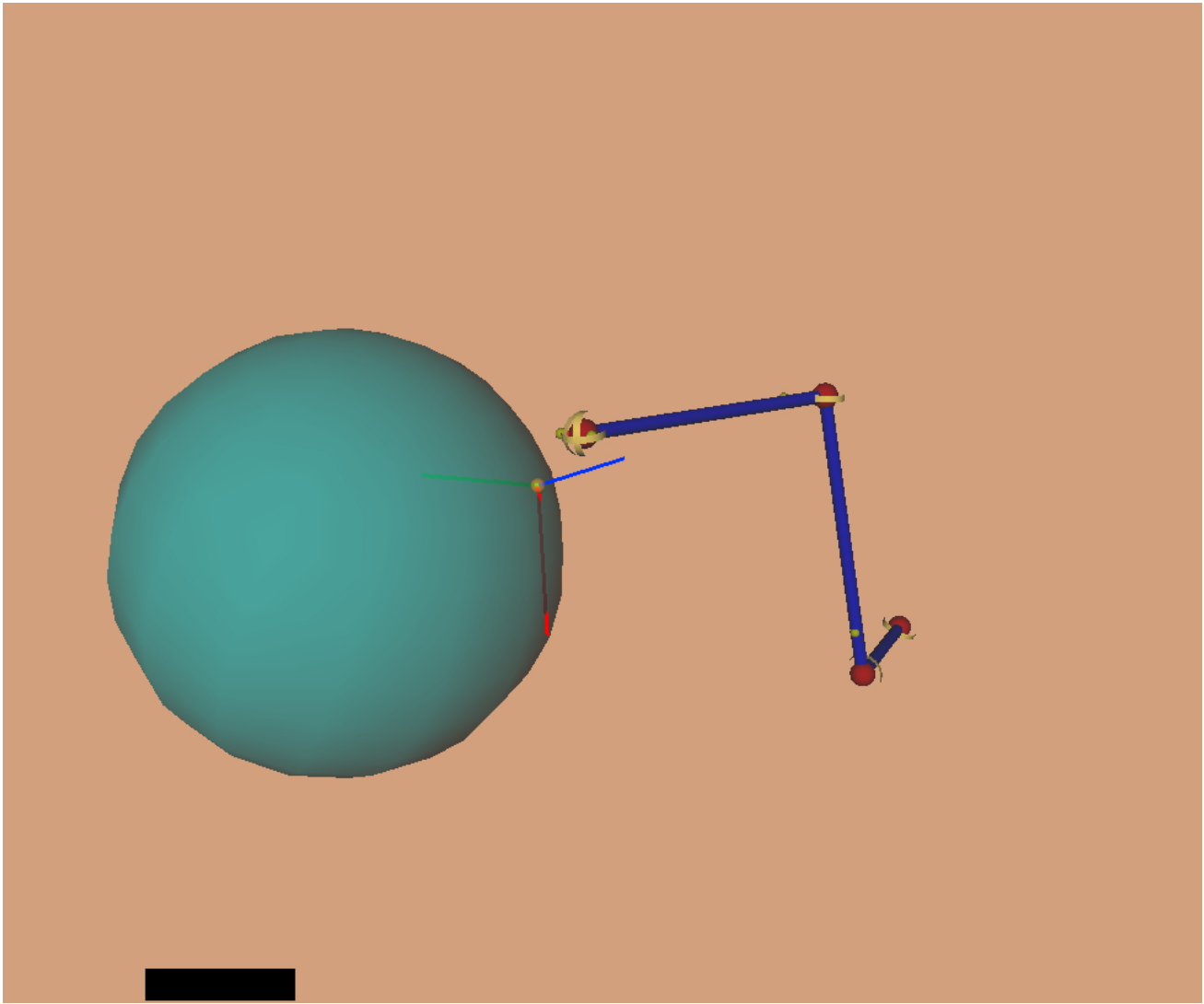


Figure 51. Example of a point contact constraint between the end-effector and a spherical surface

6.2.3.4.3.2. Circle 1d contact

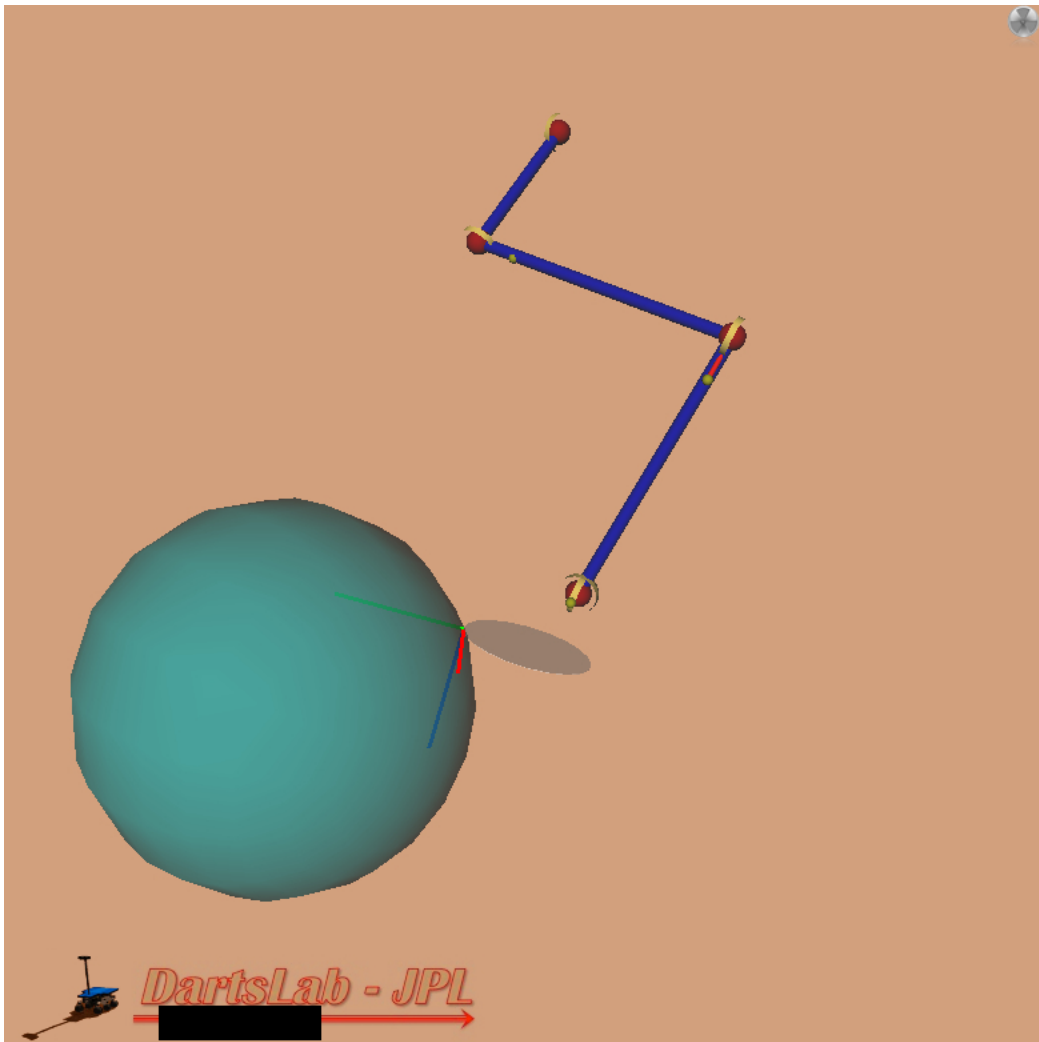


Figure 52. Example of a 1D circle contact constraint between the end-effector and a spherical surface

6.2.3.4.3.3. Ellipse 1d contact

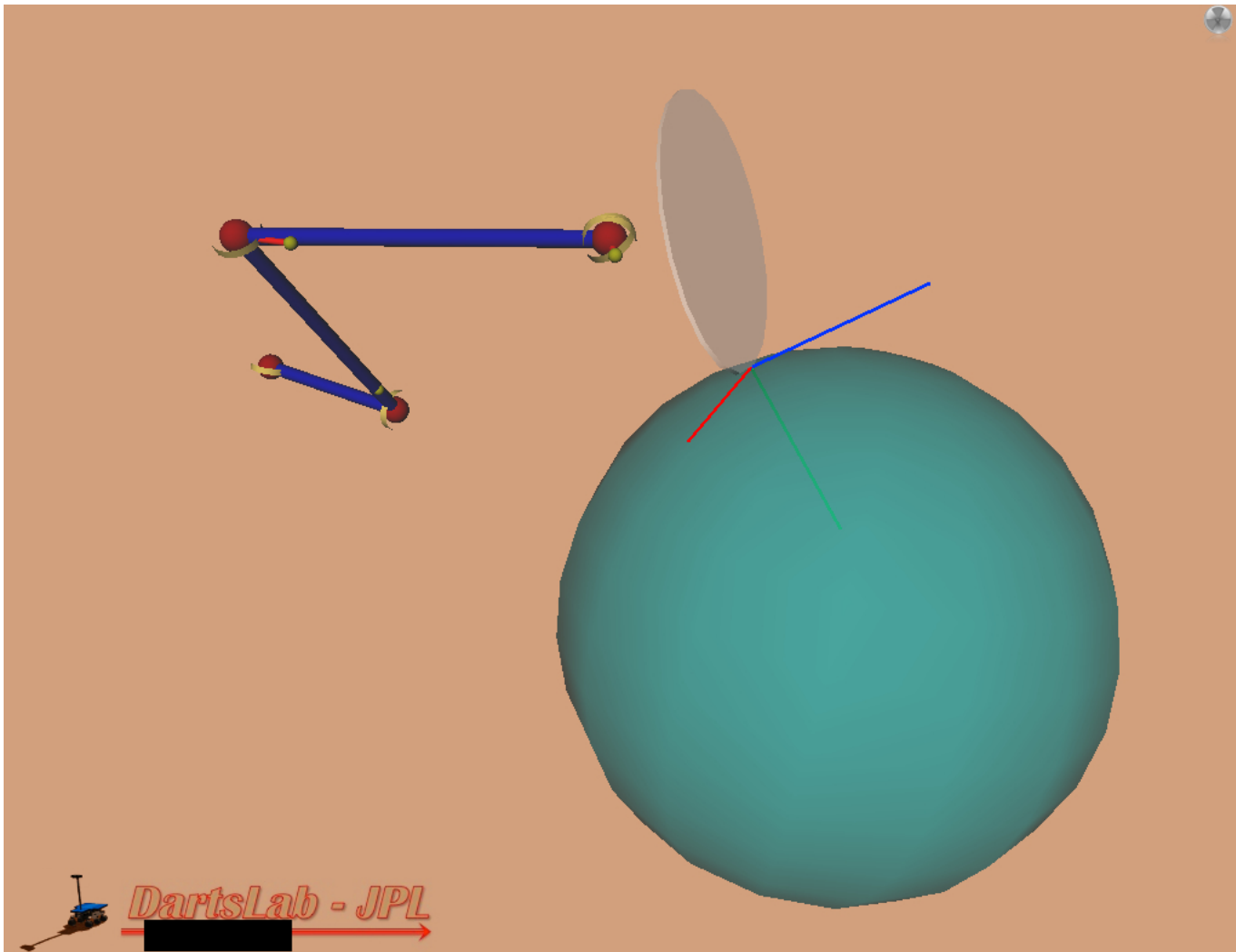


Figure 53. Example of a 1D ellipse contact constraint between the end-effector and a spherical surface

6.2.3.4.3.4. Planar 2d contact

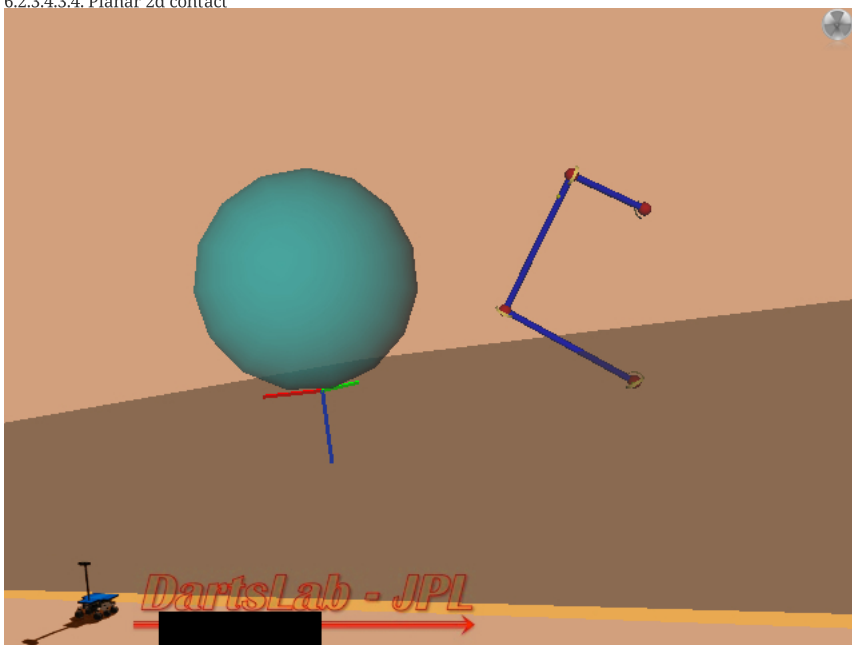


Figure 54. Example of a 2D plane contact constraint between the end-effector and a spherical surface.

6.2.3.4.3.5. Spherical 2d contact

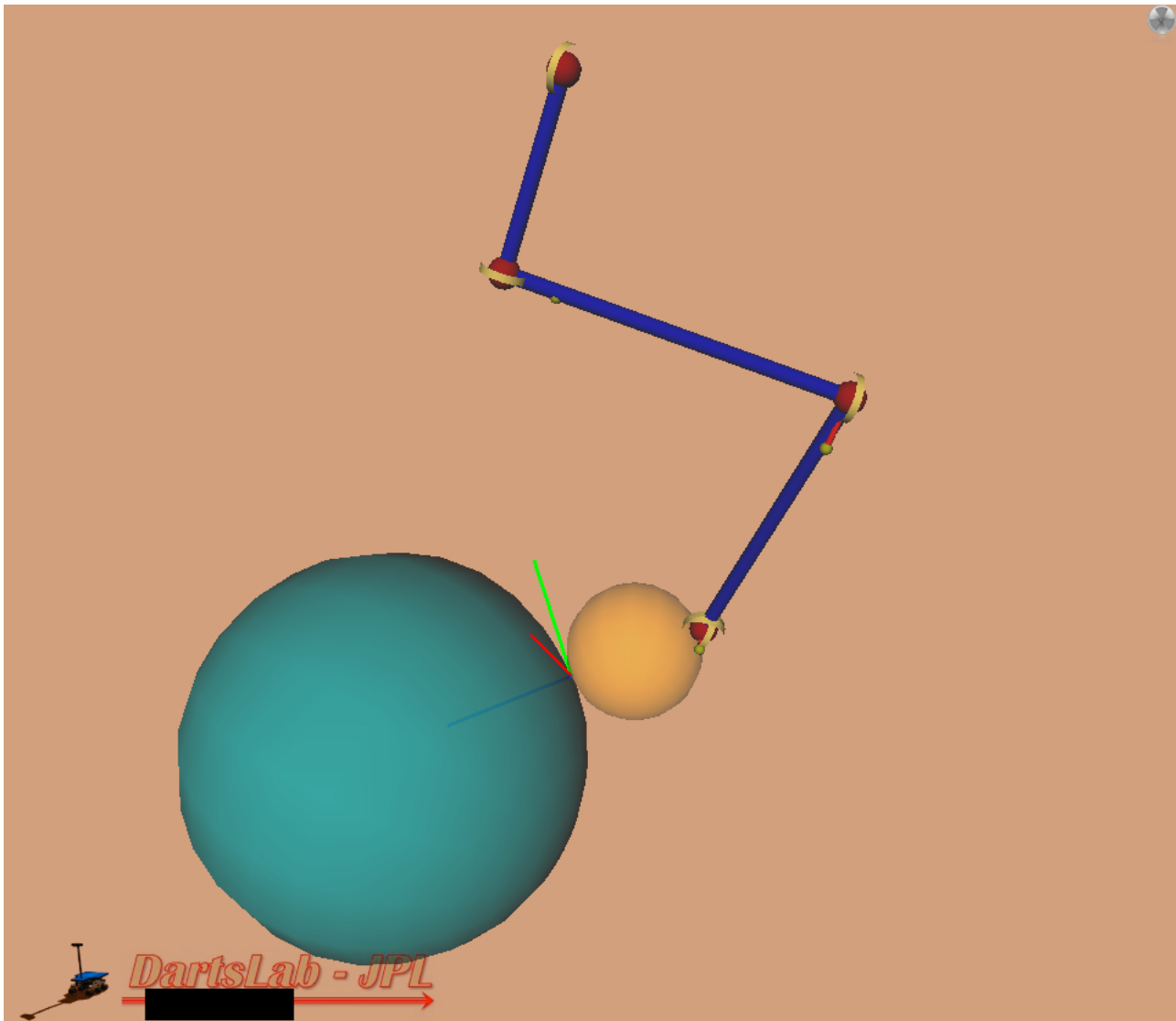


Figure 55. Example of a 2D spherical contact constraint between the end-effector and a spherical surface.

6.2.3.4.3.6. Cylindrical 2d contact

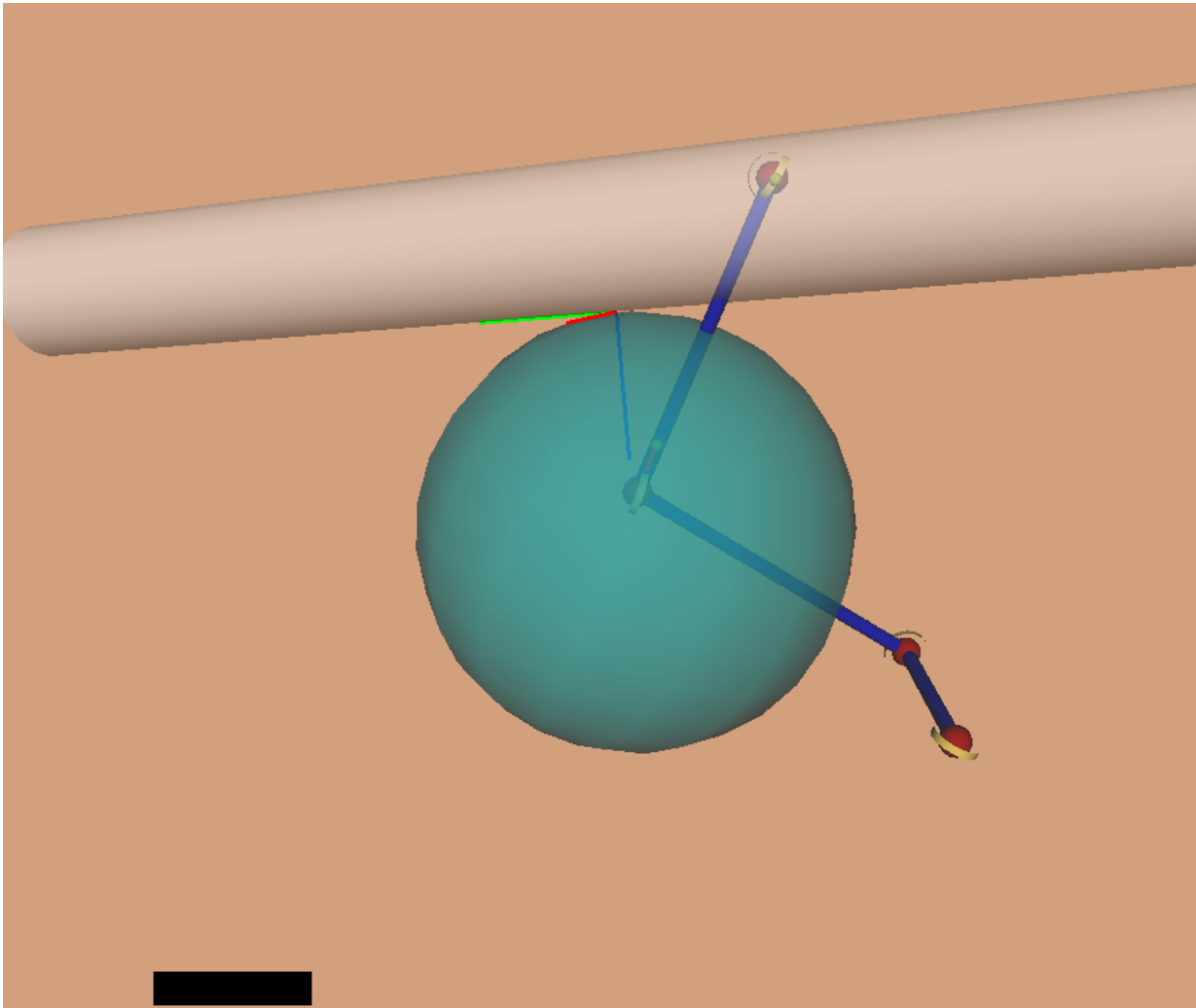


Figure 56. Example of a 2D cylindrical contact constraint between the end-effector and a spherical surface.

6.2.3.4.3.7. Ellipsoidal 2d contact

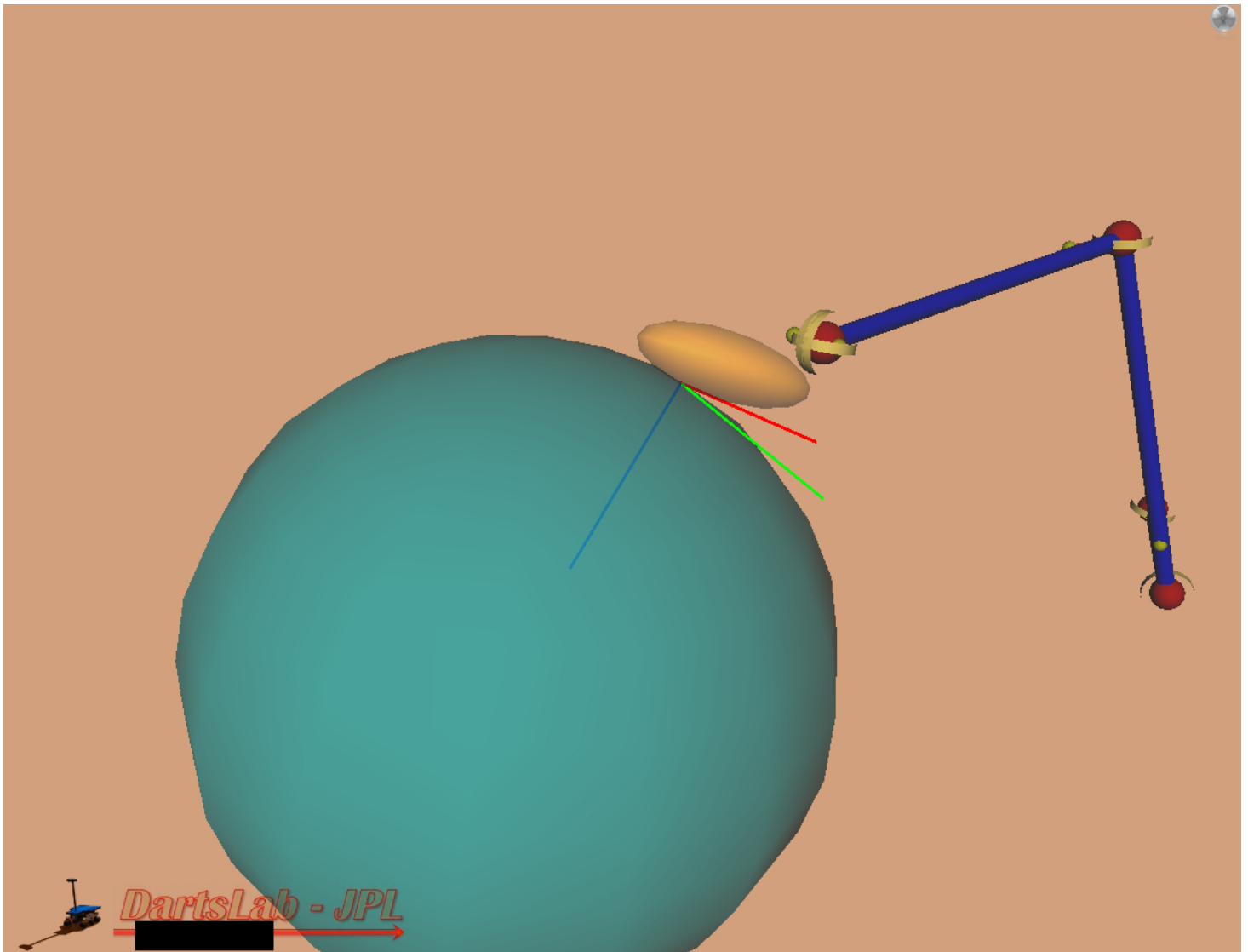


Figure 57. Example of a 2D ellipsoidal contact constraint between the end-effector and a spherical surface.

6.2.3.4.3.8. Toroidal 2d contact

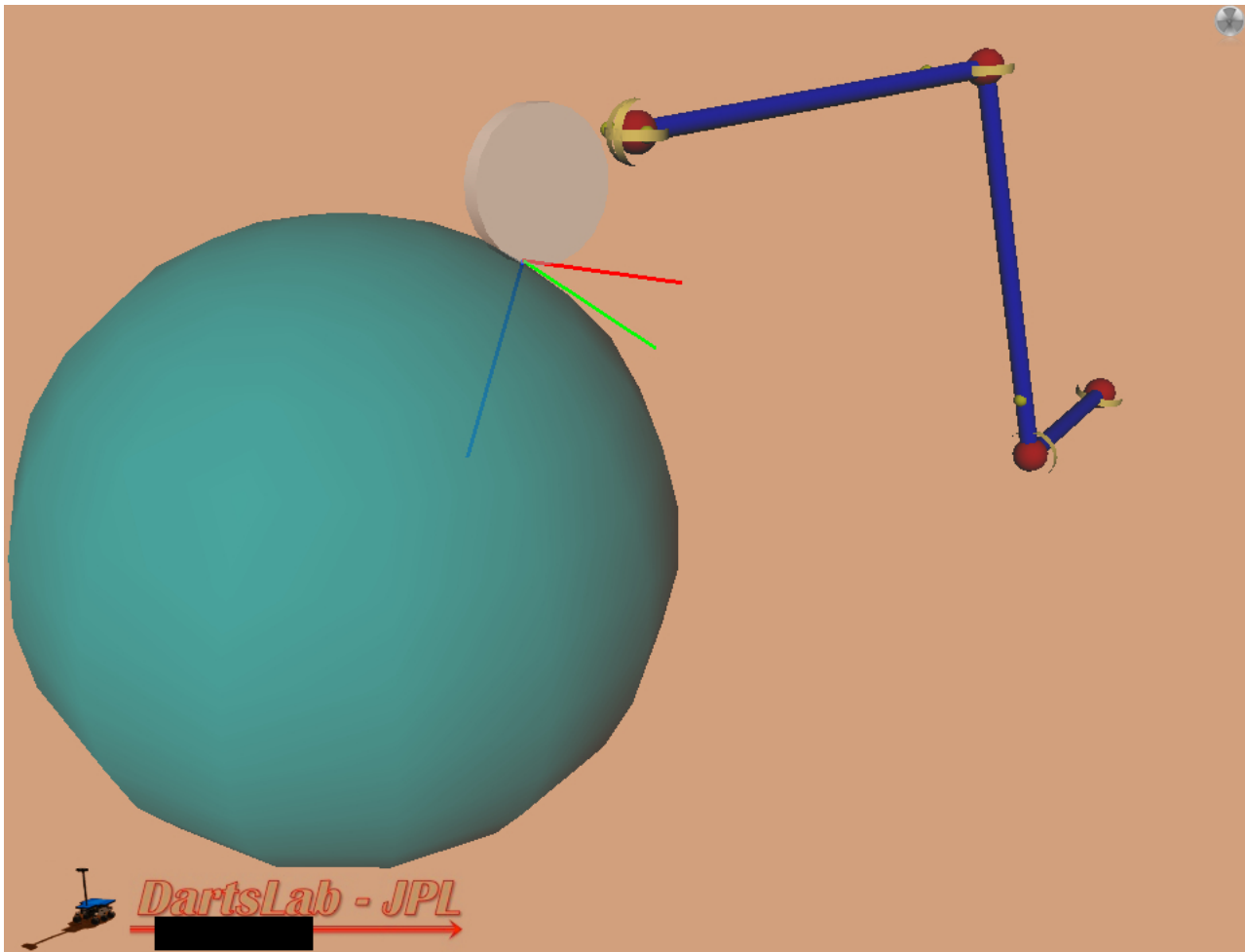


Figure 58. Example of a 2D torus contact constraint between the end-effector and a spherical surface (incorrect torus graphics - needs to be fixed).

6.2.3.4.3.9. TopoDem 2D contact



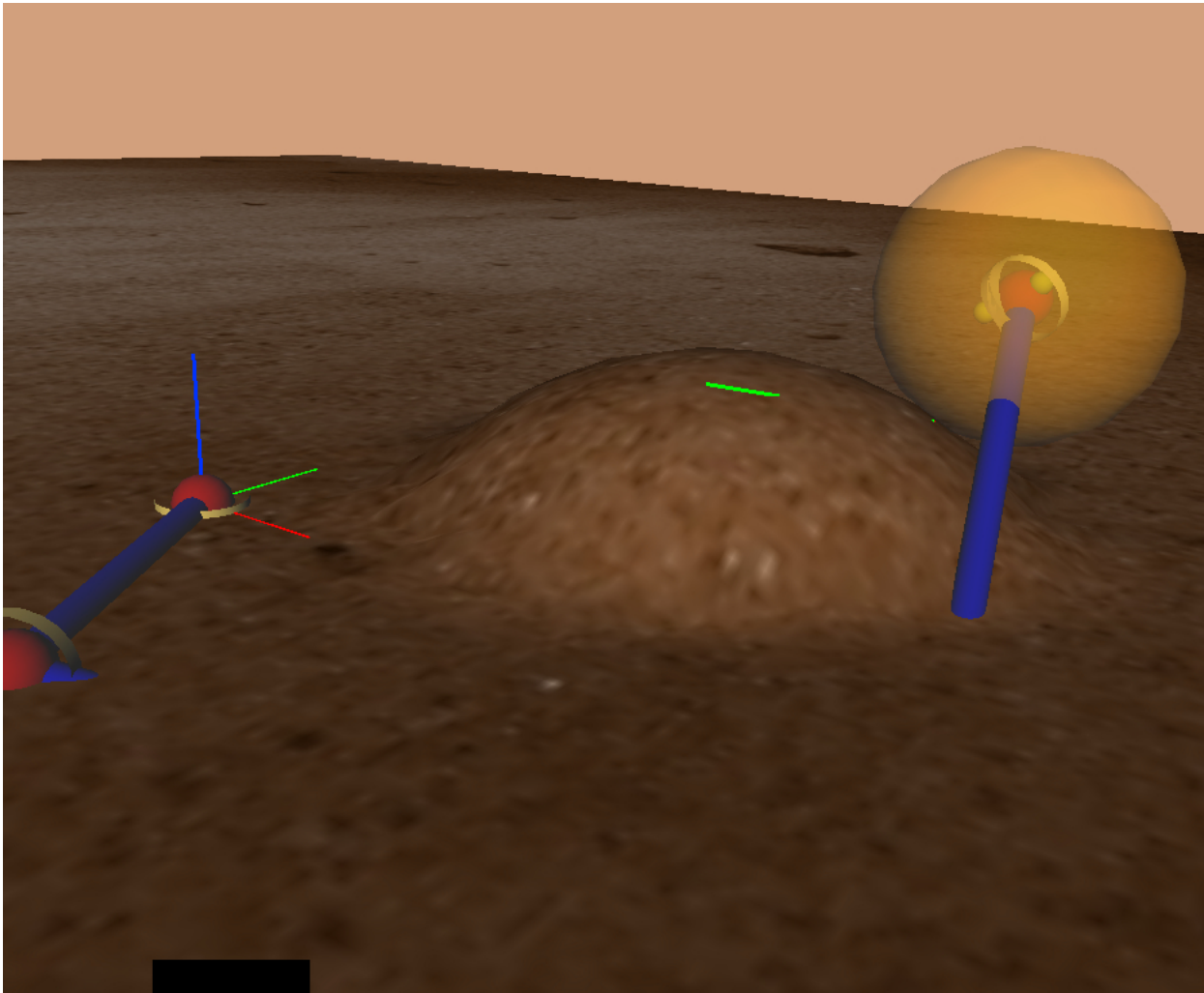


Figure 59. Example of a topodem contact constraint between the end-effector and a spherical surface.]

#### 6.2.3.5. Constraint Embedding

Constraint embedding

- Four bar linkage/wishbone constraint
- Gearing constraint
- Differential constraint

##### 6.2.3.5.1. Four Bar Linkages

TBD

##### 6.2.3.5.2. Geared Pin Joints

TBD

##### 6.2.3.5.3. Joint-Level Constraints

Algebraic constraint between arbitrary gen coord values. Example usage:

- For local constraints (gearing, differential), use constraint embedding
- More generally need to handle this
- Treat prescribed motion as a special case of this constraint on a single hinge
- Examples:
  - Prescribed motion constraint (or joint locking)
  - ATRV locked wheels constraint
  - Differential constraint
  - Gearing constraint

##### 6.2.3.5.4. Vehicle Configuration Kinematics

Example usage of vehicle configuration kinematics:

- Place rover

- KDriver constraint
- Driving Athlete with level hex
- Driving Athlete to come to dock
- Driving 2 Athletes in docked configuration

#### 6.2.4. Algorithms available

##### 6.2.4.1. Forward kinematics, frame transformations

##### 6.2.4.2. Inverse kinematics

##### 6.2.4.3. Jacobians

##### 6.2.4.4. Inverse dynamics

##### 6.2.4.5. ATBI Forward dynamics

##### 6.2.4.6. Hybrid dynamics - prescribed motion

##### 6.2.4.7. CRB algorithms

##### 6.2.4.8. DCA algorithm

##### 6.2.4.9. Interbody forces

#### 6.2.5. Additional concepts

##### 6.2.5.1. Subgraph level computations

##### 6.2.5.2. Use of data caches and dependencies

##### 6.2.5.3. Introspection

### 6.3. Usage

#### 6.3.1. Creating a multibody model

##### 6.3.1.1. Via a model file

##### 6.3.1.2. Procedurally

##### 6.3.1.3. Serial, tree systems

#### 6.3.2. Using the model

##### 6.3.2.1. Computing mass properties

##### 6.3.2.2. Simulation (forward dynamics)

##### 6.3.2.3. Control (inverse dynamics)

##### 6.3.2.4. Computing state (with constraints)

#### 6.3.3. Changing model configuration

##### 6.3.3.1. Attaching/detaching

##### 6.3.3.2. Adding/removing bodies

##### 6.3.3.3. Changing root body

#### 6.3.4. Troubleshooting and FAQ

Common usage questions, and troubleshooting tips

### 6.4. Software

An overview of the classes and functions and how they relate to each other.

#### 6.4.1. DartsMbody

#### 6.4.2. DartsBody

#### 6.4.3. DartsHinge

#### 6.4.4. DartsSubinge



**TBD:** Flesh out this subhinges section

[Link to SOA module.](#) [Reference to SOA.](#)

[Link to SOAVector section and SOAHomTran section.](#) Reference examples [SOAVector](#) and [SOAQuaternion](#).

##### 6.4.4.1. Resetting the coordinates chart

The `updateCoordOffset()` subhinge method can be used to change the offset coordinate value for the coordinates chart in order to recenter/reset it.

- Its purpose is to do whatever is needed, eg. change the chart for Rodrigues params, change Euler angles method when near singularity, and request a soft reset for the integrator. In fact this method would be needed for any 3-coordinate version of a 3-dof subhinge since this representation inherently has a singularity problem. This method should only be called when not in the midst of an integration step.
- Should there be a corresponding method at the hinge level, since a subhinge may not know when to update the offset. The specific example for this is the gimbal hinge whose individual subhinges do not have the singularity problem, but the overall hinge does have the problem.
- The solution would be for the dyn solver to loop through the `updateCoordOffset()` calls for the **hinges** in its subgraph. The default implementation of this method would be to call the corresponding `updateCoordOffset` method for each of the subhinges. The only problem with this approach is that the dyn solver leaves it to the `subhingeData` instance to handle the unmarshalling of the state vector, and the `subhingeData` only knows about subhinges, and not hinges.
- Actual implementation of this modifies the offset and input coordinates, sets them in hinge. The return value is a bool indicating a change has happened. The integrator soft reset is not done here, but is the responsibility of the calling program.

The `SUBHINGE_RODRIGUES_SPHERICAL` subhinge type uses the minimal `SOARodriguesParam` coordinate values for its generalized coordinates. However these coordinates have singularity at multiples of  $2\pi$ . Hence its coordinates chart needs to be recentered in the vicinity of a singularity. For this case the chart offset is defined by a `SOAQuaternion` value and the generalized coordinate Rodrigues param defines the offset to the actual attitude value.

#### 6.4.5. DartsNode

#### 6.4.6. DartsSubGraph

##### 6.4.6.1. Internal details



TBD: update and clean up this section

The `DartsSubGraph` class caches structural data in several container members as described below. Some of these are kept around for performance reasons as algorithms traverse the subgraph topology. We refer to subgraphs that create compound boundaries as `constraint embedding (CE)` subgraphs since their bodies are not a subset of the parent graph, while regular subgraphs whose bodies are a subset of those of the parent are referred to as `non-CE` subgraphs.

- `bool _has_parent_bodies`: `true` for non-CE graphs and `false` for CE graphs.
- `DartsSubGraph* _graph`: the parent/owner subgraph for the subgraph
- `DartsSubGraphList _child_subgraphs`: list of all subgraphs of this subgraph
- `DartsBodyBaseList _owned_bodies`: list of bodies owned by this subgraph - only these bodies will be destroyed by the subgraph destructor. Physical bodies are owned by the multibody subgraph. Compound bodies are owned by the subgraph creating them.
- `DartsBody* _virtualRoot`: the virtual root parent body for the subgraph. Note that a compound body cannot be a virtual root for a subgraph. This member is updated when locking by the `_updateSubGraphBodies()` which is called by the constructor and `add/removeBody` methods as well as when locking by the `_lockObject()` method.
- `DartsBodyBaseList ordered_bodies`: this contains a list of all `rigid` and `compound` bodies owned by the subgraph. **Is order important?** This list does not include the virtual root body.
- For a CE-subgraph, the compound body's constructor calls `updateAggregatedBodies(agg_sg, cb_bd)` to update this list so that all the aggregated bodies are removed, and replaced by the single compound body. This list is updated when locking by the `_updateSubGraphBodies()` which is called by the constructor and `add/removeBody` methods as well as when locking by the `_lockObject()` method.
- `DartsBodyBaseList _base_bodies`: list of base bodies for the subgraph, i.e. the bodies in the subgraph that are the immediate children of the virtual root body. Note that in general that the virtual root body may have additional children bodies that do not belong to this subgraph. This list is updated when locking by the `_updateSubGraphBodies()` which is called by the constructor and `add/removeBody` methods as well as when locking by the `_lockObject()` method.
- `DartsChildBodiesMap _child_bodies_map`: map defining the children bodies for a body for this subgraph. This map is initialized as a copy of the parent subgraph's own `_child_bodies_map` (**why** - perhaps so CE subgraphs have a starting point since `lockObject` does not set this map for them), it is possible initially for this map to have additional key entries for bodies that may not belong to this subgraph. The `_topologicalChildBodies(bd)` method uses this map to return the result for non-multibody subgraphs. The `dumpChildBodiesMap()` method can be used to dump the contents of this map.
- For a non-CE subgraph, this map contain key entries for each subgraph body with the value being the child bodies (per the parent subgraph) filtered down to those that belong to the subgraph. The subgraph's `_lockObject` method calls `_setupChildBodiesMap()` to populate this for non-CE subgraphs.
- For an aggregation subgraph, this map has entries for each of its bodies, and the value for each body is the list of bodies in the parent subgraph that are immediate children but are also not within the aggregation subgraph. The compound body's constructor calls `updateAggregatedBodies(agg_sg, cb_bd)` which in turn calls `updateAggregatedChildBodiesMap(agg_sg)` to update this map for the aggregation subgraph with key entries for each aggregated body, and value being the non-aggregated children for the body.
- For a CE-subgraph, there are entries related to compound bodies. For each compound body, there is a key entry for its virtual root - or possibly instead the subgraph's compound body that contains the virtual root - with the compound body and other child bodies in its values list. Also there is an entry for each compound body as a key with values being its immediate children. The compound body's constructor calls `updateAggregatedBodies(agg_sg, cb_bd)` to update this map with the new info for the compound body. `undoAggregateBodies(agg_sg, cb_bd)` does the reverse updates when a compound body is being deleted.
- `DartsParentBodiesMap _parent_bodies_map`: map defining the parent body for a body for this subgraph. This map is initialized by the corresponding map for the parent subgraph. If there is no entry for a body, then the parent body for the body is its physical parent body. This map is used by the `topologicalInbBody(bd)` method. The `dumpParentBodiesMap()` method can be used to dump the contents of this map.
- This map is empty for non-CE and aggregation subgraphs
- For a CE-subgraph, there are entries related to compound bodies. For each compound body, there is a key entry with value being the virtual root - or possibly instead the subgraph's compound body that contains the virtual root. Also, for each child body of a compound body, there is a key value for the child with the compound body as the value. The compound body's constructor calls `updateAggregatedBodies(agg_sg, cb_bd)` to update this map with the new info for the compound body. `undoAggregateBodies(agg_sg, cb_bd)` does the reverse updates when a compound body is being deleted.

- `_descendant_bodies` : map defining the list of outboard descendant bodies for each body owned by the subgraph. This is updated by the `_lockObject()` method. his list is updated by `_computeTerminalBodies()` which in turn calls `_updateDescendantBodies()` when locking via the `_lockObject()` method.
- `std::map<size_t, const DartsBodyBase*> _body_anc_map` : map to keep track of the ancestor body for each pair of bodies (used for OSC computations). This is updated by the `_lockObject()` method.
- `DartsChildBodiesMap _terminal_bodies_map` : map defining the downstream terminal bodies for the base bodies for this subgraph. There are only entries for base bodies, i.e. one for each sub-tree in the system. This map is updated by `_computeTerminalBodies()` when locking by the `_lockObject()` method.
- `DartsBodyBaseList _terminal_bodies` : list of terminal bodies (i.e. bodies with no children) for the subgraph. This list is updated by `_computeTerminalBodies()` when locking by the `_lockObject()` method.

#### 6.4.7. DartsTreeDynamicsSolver



TBD: Flesh out this dyn solver section

##### 6.4.7.1. Solving the dynamics equations

This class calls `updateCoordOffsets()` for each subhinge before calling `integrate` (or the first thing in the `integrate` method). This allows the subhinges to change charts etc. to fix up the coords so that we do not have any ill behavior.

#### 6.4.8. CM frame

#### 6.4.9. General Functions

##### 6.4.9.1. visualizeSpatialForces

The `visualizeSpatialForces` function is used to visualize the external spatial forces of the bodies in the simulation. It takes the following parameters:

- `sg` (*DartsSubGraph*) - SubGraph whose bodies' spatial forces will be displayed.
- `scene` (*DartsFacadeScene*) - The scene the forces will be drawn in.
- `sgForExternalSpatialForces` (*DartsSubGraph, optional*) - The SubGraph used when calling `body.externalSpatialForces`. If None, then the SubGraph supplied via the first argument will be used. For example, these two SubGraphs might be different if compound bodies are used. In that case, `sg` may be `sim.mbody()` and `sgForExternalSpatialForces` may be `sim.dynSolver().sg()`. (Default = None)
- `enable` (*bool, optional*) - Used to enable/disable visualization of the spatial forces. (Default = True)
- `scale` (*float, optional*) - Used to scale the external force vectors. (Default = 0.1)

### 6.5. Raw documentation



TBD: Need scrubbing before integration.

##### 6.5.1. Ndarts: Switch `DartsBody.setBodyParams()` to use `createPartGeometries()` instead of `load()` for geometries



TBD: Needs scrubbing. Notes brought over from [issue](https://dartsgitlab-internal.jpl.nasa.gov/darts-lab/jpl-internal/all-access/development/dynamics/ndart/-/issues/5#note_8613) (https://dartsgitlab-internal.jpl.nasa.gov/darts-lab/jpl-internal/all-access/development/dynamics/ndart/-/issues/5#note\_8613).

This approach tries to minimize the amount of code changes. The main change to the existing part geometry classes is to

- extend the `load()` method to handle the case where `AssimpImporter` is used to first generate a `\DMesh::MeshGroup` with a list of `DMesh::Mesh` instances (instead of just a single one previously),
- and to provide an API for manipulating the material properties of the individual `DMesh::Mesh`.

Everything else can stay as is, including

- the current handling of primitive geometry shapes and their methods
- the `load` method for specialized file types
- the CS level `dmesh()` method for creating a geometry

Proposed changes

- extend the CS part geometry/scene object to have a CS level `MeshGroup` instead of single mesh. This is to hold the `MeshGroup` coming from `assimp` obj importers. This will impact `DspaceScene` and `DOptixScene` which are currently designed to create a single 'C' level `dsGraphicsObject` and `DOptixRenderable` instances respectively for the single mesh - they will need to be extended to loop through and create these instances for each mesh in the mesh group. `BulletScene` and `DMeshScene` already support lists of 'C' level meshes for each part geometry - but again would need to be extended to loop through and do so for each mesh group mesh.
- Extend `AssimpImporter` to return a `MeshGroup` instance instead of list of meshes
- each CS part geometry will have a list of `DMesh::MeshGroup` instance for the meshes created by the `AssimpImporter`
  - Add API for retrieving individual such `DMesh::Mesh` instances to overwrite its material properties. `DMesh::MeshBuild` already provides an API for retrieving its individual `Mesh` instances which can be used for this.
- For collision world CS, add collision filters between `Mesh` instances for the same scene object.
- Extend the `SceneObject::visibility`, `SceneObject::scale` etc methods to apply this to all component meshes.

- For singleton meshes (eg. primitive shapes, dmesh shapes) we will continue to have the current support. We should have only one instance of the `MeshGroup` pointer be non-null.
- When deleting a scene object, all the `C` meshes corresponding to the `MeshGroup` should be deleted.

## 6.6. Sphinx documentation

### 6.6.1. DARTS Multibody Modelings Basics

The Ndart's implementation of DARTS (???) can model a wide range of multibody mechanisms. DARTS/Dshell simulations generally involve at least a single rigid body (for the a spacecraft, for exampe) or potentially a complex, multiply connected multibody system.

For more detailed background information, please see:

- *Robot and Multibody Dynamics, Analysis and Algorithms*. Abhi Jain. Springer. 2011.
- Darts Manual 1997.

#### 6.6.1.1. DARTS Multibody Modeling Concepts

- **Tree Topology Systems.**

Tree-topology systems are multibody systems with no closed-chains.

In many cases, the multibody physics of the system can be modeled as a "Tree Topology System". Ndart's can handle closed-chain systems, but often a Tree topology system is adequate.

"Closed-chain" means that a set of bodies are connected to each other in a loop, or "closed chain".

In **Tree Topology Systems**:

- there is one and only one path from one body to another
- each body has only a single *parent*
- each body can have multiple *child* bodies.

- **Flexible Body.**

A **Flexible Body** has *deformation* degrees-of-freedom (dofs) in addition to rigid-body *articulation* dofs.

- **Spatial Vectors.**

These are 6-dimensional vectors that simplify many dynamics related expressions.

- A *spatial velocity* vector for a coordinate frame is the 6-dimensional vector formed by the concatenation of the angular velocity and linear velocity vectors for the frame.
- The *spatial force* vector associated with a frame is the 6-dimensional vector formed by the concatenation of the moment and force vectors associated with the frame.
- The `SOA` vector/matrix [library<index.html>](#) implements spatial vector math (as well as other vector/matrix math used in Darts).

- **Nodes.**

Set of discrete points in the body:

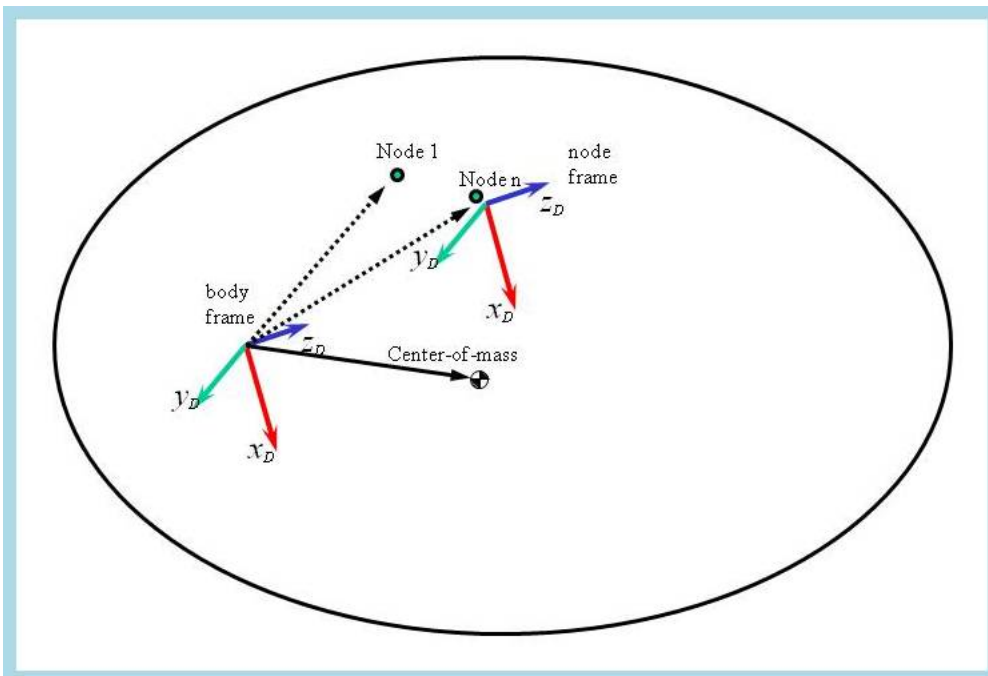


Figure 60. Body Frame and Nodes

- **Nodes:**

- can be defined by the user
- are associated with *hinges* interconnecting bodies
- can arise from modeling flexible modes: the structural model of a flexible body involves a spatial discretization of the body into a collection of nodal elements
- **Node Frames**
  - each node has its own reference frame.
    - for a rigid body, all the node frames are aligned with the body frame.
    - for a flexible body, the deformation of the body causes relative motion between the node and the body frames.
- **Actuator Nodes**

Point of application for applying external forces and moments on the body that can arise from actuators, environmental effects or other sources.
- **Sensor Nodes**

Point at which body information is obtained e.g. body attitude, rates

- **Hinges/Joints.** The kinematic constraints on the relative motion between adjacent bodies are modeled by hinges coupling the bodies.
  - hinges can be translational, ball, full-6dof, slider, etc.
  - each hinge has an associated pair of *hinge nodes*
    - *inboard hinge node* (Onode) located on the inboard body
    - *outboard hinge node* (Pnode) located on the outboard body.

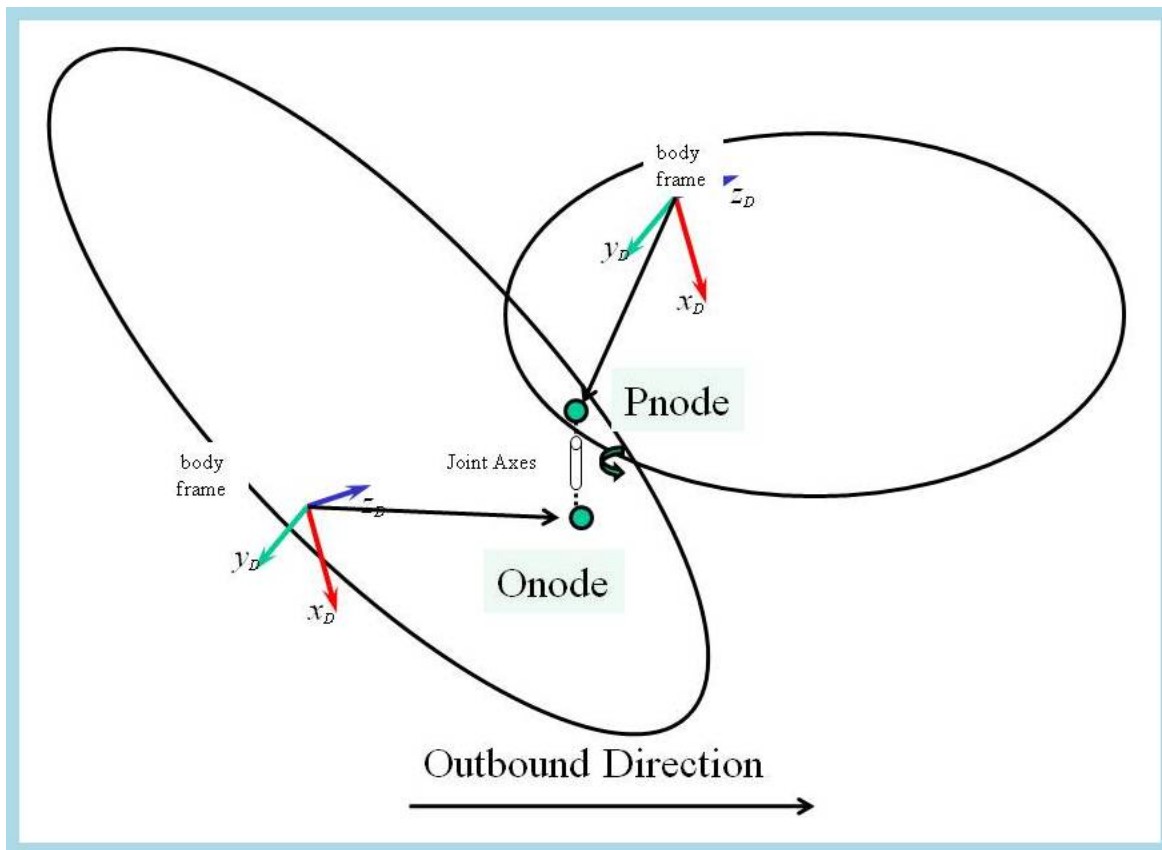


Figure 61. Multibody Hinges and Nodes

- the *generalized coordinates* for the hinge parameterizes the relative configuration of the hinge frames
  - the *generalized velocity* for hinge parameterizes the relative spatial velocity between the frames.
  - when the generalized coordinates of a hinge are the zero element, the *Onode* and *Pnode* frames coincide
  - Historical note: the the "O" and "P" nodes are in alphabetical order going from the base towards the leaf bodies. The "O" and "P" terminology goes back to the early days of Darts and uses the same nomenclature as the old multibody simulation software DISCOS. DISCOS (Dynamic Interaction Simulation Of Controls and Structure) was developed by H.P. Firsch at Goddard Space Flight Center. For further information about DISCOS, search for DISCOS in <http://naca.larc.nasa.gov/search.jsp>.
- 
- **Zero Configuration.** Multibody configuration where all the generalized coordinates are the zero element.
    - used to define all the kinematic and dynamics model information for the system in the model file e.g. vectors from the body frames to the joint node frame.

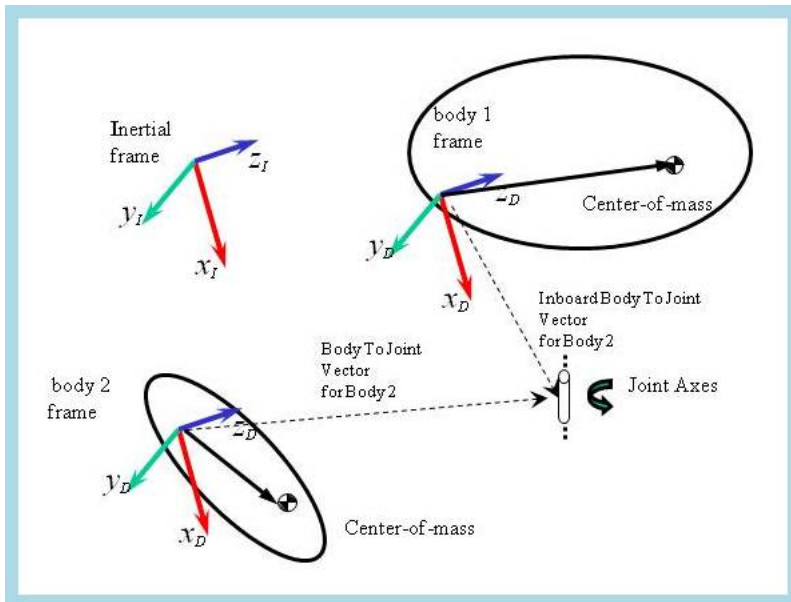


Figure 62. Multibody Zero Configuration

- In the zero configuration, the following terms are used to describe the geometry of the joint connecting a body to its parent body. Any of these terms may be zero.

**inbToJoint**

The vector from the parent body reference frame to the joint, expressed in the parent body frame.

Note that the rotation be set with the **inbToJointQuat** and the full transform can be set with **inbToJointTransform**.

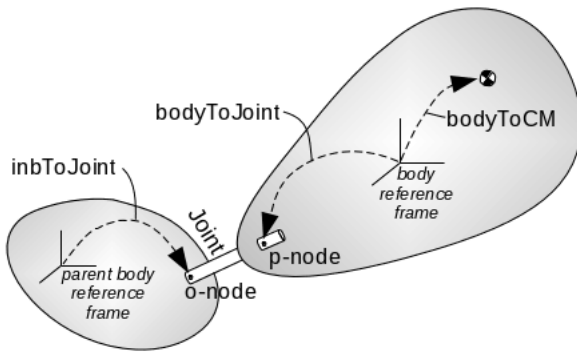
**bodyToJoint**

The vector from the child body reference frame to the joint, expressed in the child body frame.

Note that the rotation be set with the **bodyToJointQuat** and the full transform can be set with **bodyToJointTransform**.

**bodyToCM**

The vector from the body reference frame to the center of mass for the body (in the body's reference frame).



- Note that all reference coordinate systems are parallel in the "Zero" configuration (unless **intToJointQuat** or **bodyToJointQuat** are not the identity matrix).

• **System Generalized Coordinates and Velocities**

- The *state vector* for a multibody system consists of a combination of:

- the generalized coordinates vector; denoted **Q** consisting of
  - hinge coordinates of hinges connecting bodies to each other
  - modal coordinates related to the flexibility of a body
- the corresponding generalized velocities vector; denoted **U**

- The time derivative  $dQ/dt$  of **Q**, denoted **Qdot**, is obtained by a simple kinematic transformation of the generalized velocities vector **U**. For a joint with full rotational freedom, **Q** is modeled as a quaternion.

- The combination of all the hinge generalized forces is designated as **Tjoint**

- The combination of all the external forces acting on the multibody nodes is denoted as **Fext**

- Darts computes  $dU/dt$  denoted as **Udot** which is then used by the Dshell integrator to compute **U**.

- **Kinematic Maps**

- mapping from  $U$  to  $\mathbf{Qdot}$  is usually a trivial identity map
- exception: attitude dof generalized coordinates are the 4 elements of the attitude quaternion, while the 3 elements of the angular velocity vector are used for the generalized velocity coordinates. In this case, the mapping from the generalized velocities to the generalized coordinate rates is a non-linear, configuration dependent kinematic map.

- **Darts Prescribed and Non-Prescribed Hinges**

- The Multibody hinge generalized coordinates state can belong to one of two categories:

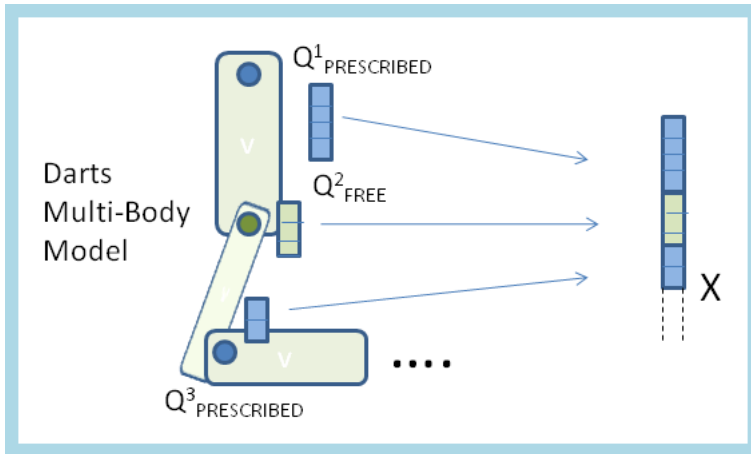


Figure 63. Multibody Prescribed and Nonprescribed States

- **Non-Prescribed Hinges.** The coordinates ( $Q_{FREE}$ ) associated with these hinges evolve in response to user or model specified **generalized forces** at the hinges connecting the bodies as well as input forces (and torques) on the multi-body system.
  - Represents the *Forward Dynamics* of the multi-body system.
- **Prescribed Hinges.** The coordinates ( $Q_{PRESCRIBED}$ ) associated with these hinges evolve in response to user or model specified **kinematic profiles** at the hinges connecting the bodies as well as the input forces (and torques) on the bodies in the multi-body system.
  - Represents the *Inverse Dynamics* of the multi-body system.
  - Prescribed motion hinges are often used to freeze articulated dofs, or when there are tight control loops governing the motion of the hinge dofs.

There is a related concept called "masking". For more detail about prescribing and masking, please see [prescribing](#).

- **Forward and Inverse Dynamics.**

- *Forward Dynamics* is solving for output accelerations ( $dv/dt$ ) given input forces ( $f$ )
- *Inverse Dynamics* is solving for output forces ( $f$ ) given input accelerations ( $a$ ) by using Newtons Law suitably generalized for articulated/flexible body dynamics.

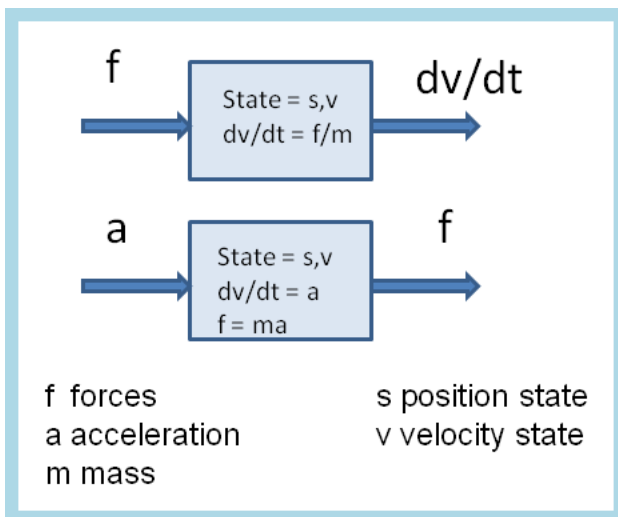


Figure 64. Forward and Inverse Dynamics

- In Darts *Forward Dynamics* involves computing the unknown hinge accelerations  $\mathbf{Udot}$  from the generalized forces  $F_{ext}$  and generalized hinge forces  $\mathbf{T}_{joint}$
- In Darts *Inverse Dynamics* involves computing the generalized hinge force  $\mathbf{T}_{joint}$  from a *prescribed motion* of the generalized hinge  $\mathbf{Udot}$  and the external generalized forces  $F_{ext}$



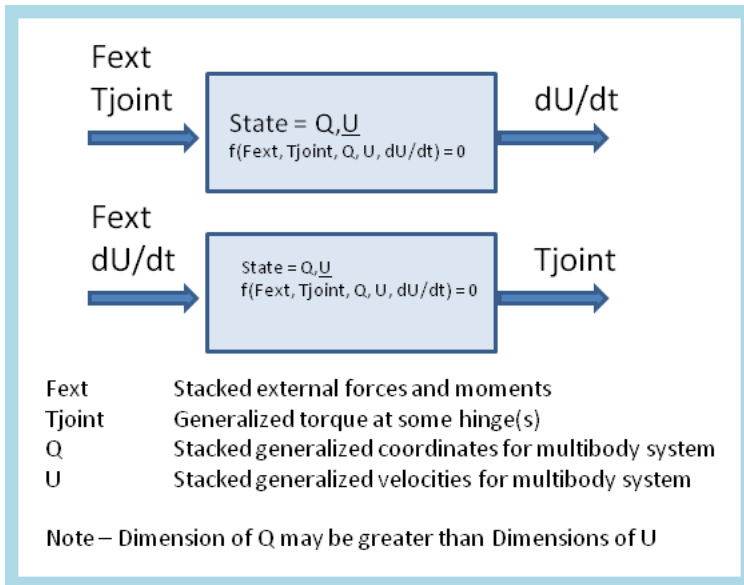


Figure 65. Darts Dynamics Engine

- o Darts solves the general mixed forward/inverse dynamics problem i.e. where the hinges are a mix (respectively) of Non-Prescribed/Prescribed

6.6.1.2. Overview of DARTS Framework Objects

- The DartsMbody object

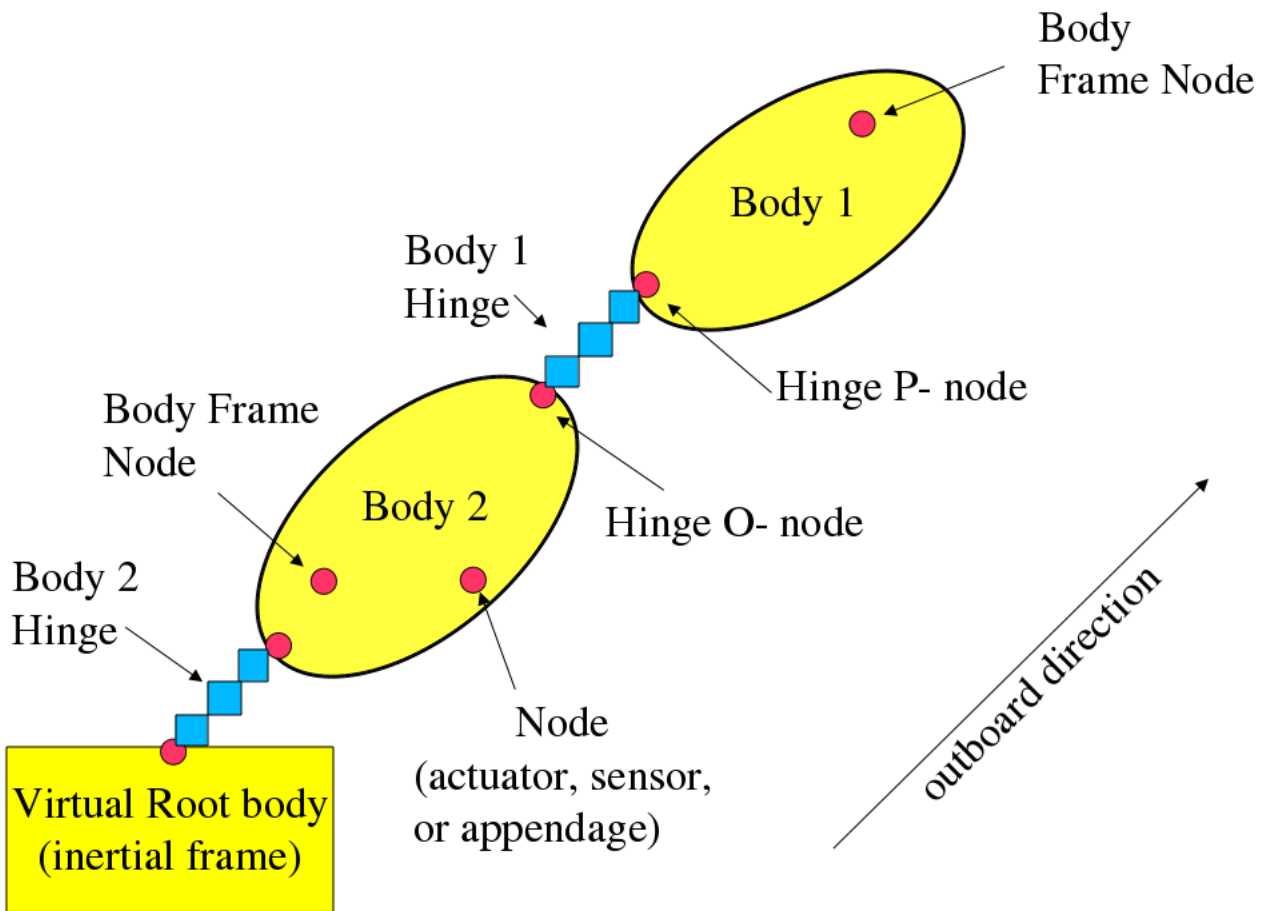


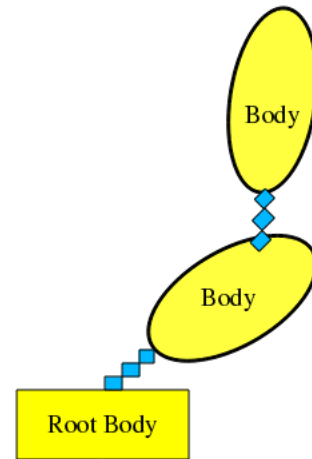
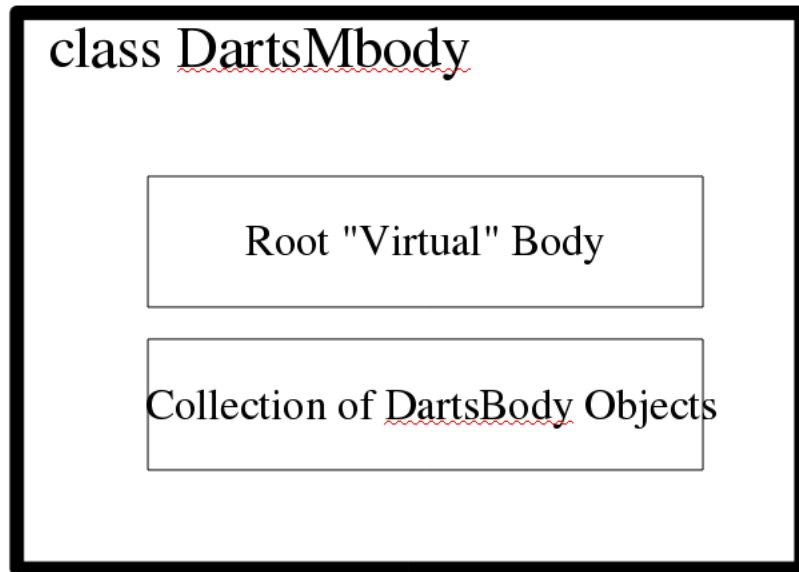
Figure 66. A Darts Mbody ("multibody") object consists of DartsBody objects.

Continued:

## DartsMbody Object

Manages a collection of DartsBody objects

The "root" is the head of the body list.



- 
- The DartsBody object

# DartsBody Object

Manages a linked list of DartsNode objects.

## class DartsBody

### Hinge

(connection between body and its parent body)

### Hinge Node

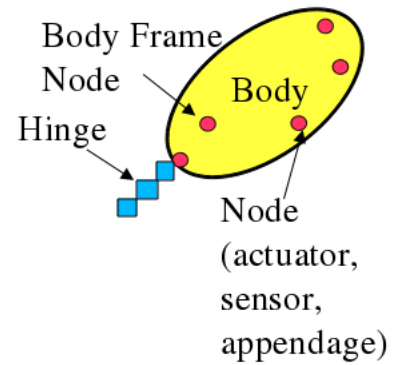
(node where the Hinge is attached)

### Body Frame Node

(node containing body frame properties)

### LinkedList of Nodes

(attachment pts for appendages, actuators, sensors)

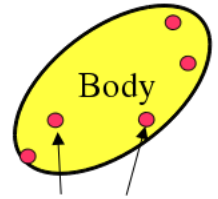
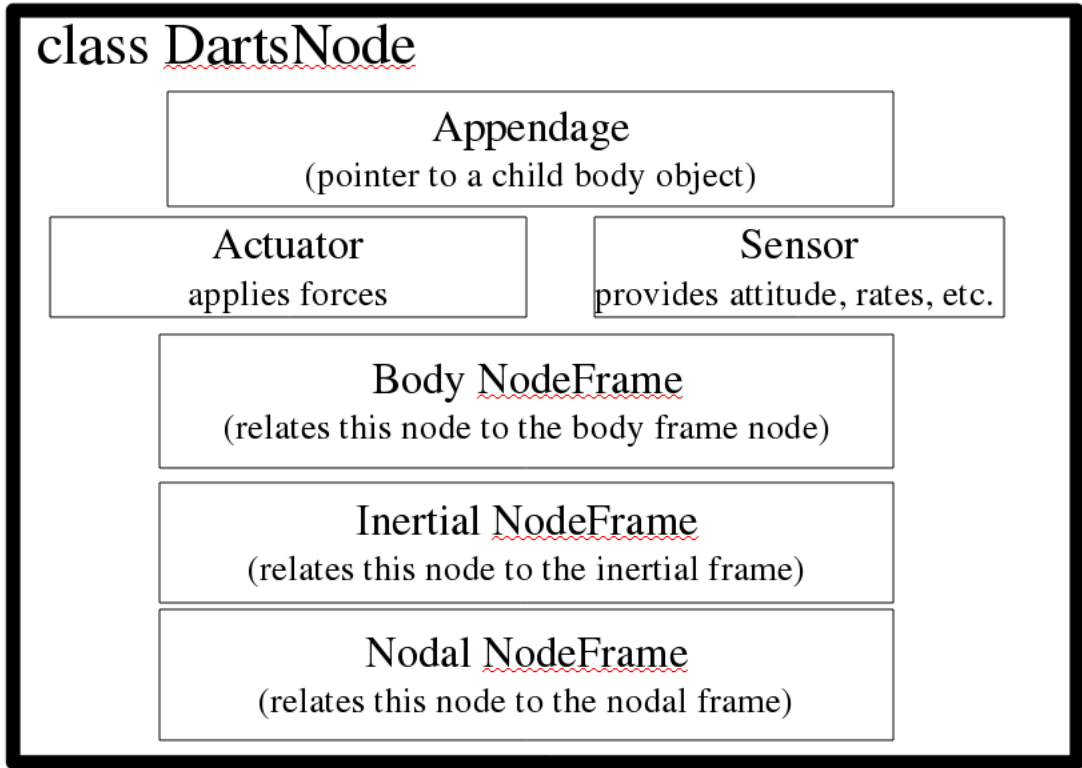


- The DartsNode object

# DartsNode Object

Represents a point on a body.

Holds the state properties of a point on a body.



Nodes  
(actuator,  
sensor,  
appendage)

- The DartsHinge object

## DartsHinge Object

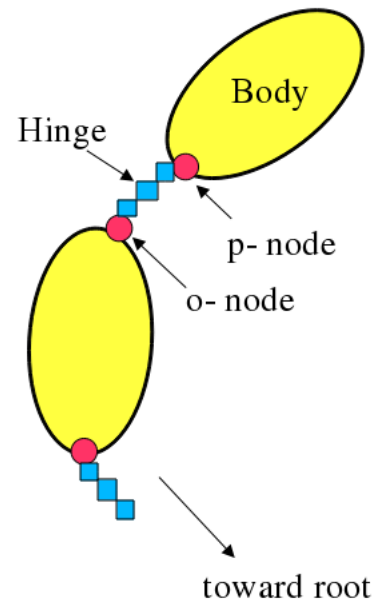
Models large angle relative motion between two connected bodies.  
Container for DartsSubhinge objects.

### class DartsHinge

Inboard Node (*o- node*)  
(pointer to attachment node on inboard body)

Hinge Node (*p- node*)  
(pointer to attachment node on hinge' s body)

Linked List of DartsSubhinge objects



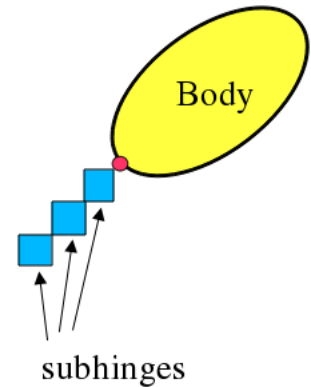
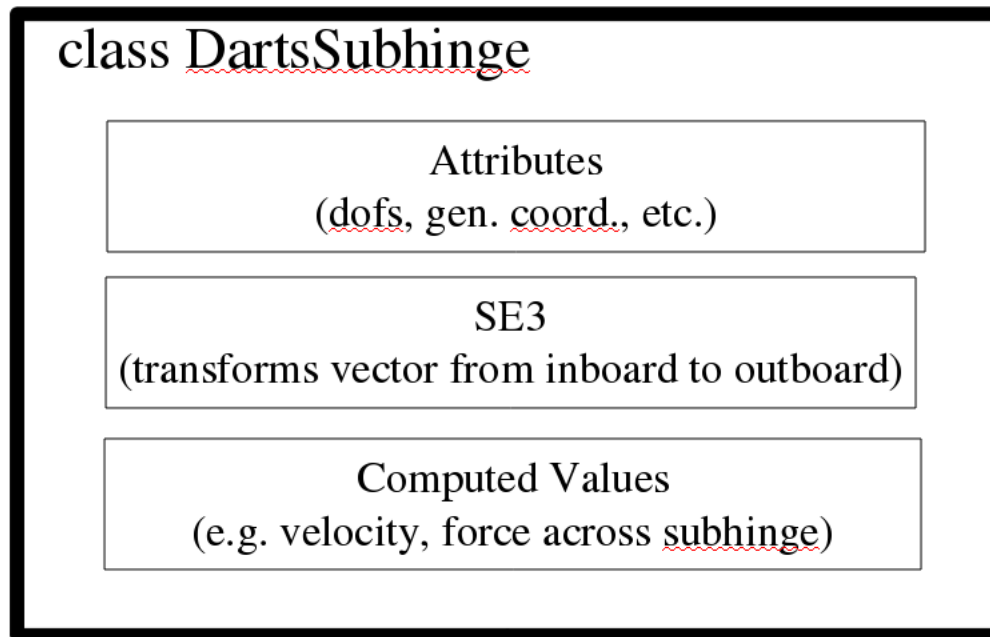
- The DartsSubhingeBase object

# DartsSubhinge Object

Basic hinge element.

Methods for computing the velocity/force across the subhinge are declared virtual.

New hinge types are introduced by subclassing the DartsSubhinge class.



## 6.6.2. Ndarts Hinges

### 6.6.2.1. Introduction

In Darts, hinges (or joints) are used to attach nodes to one another and provide degrees of freedom between attached nodes.

The Darts Hinge class is derived from the ChainedFrame2Frame class. Hinges are responsible for connecting two bodies together and have varying degrees of freedom depending on the type of hinge used.

Because nodes are implemented as frame objects, it is intuitive to implement the hinges that connect the nodes as Frame2Frame objects.

#### 6.6.2.1.1. Hinge Types (Joint Types)

There are two main categories of hinges (or joints). Hinges are the actual objects that connect nodes together; whereas Subhinges are subparts from which hinges are built. All of the main hinge types can be broken down into "pin" and "slider" subhinge types, however there are several other three degree of freedom subhinge types for more complex hinges.

Here are a list of the basic hinge types with brief descriptions:

- PIN - a one axis rotational/revolute hinge (1 DOF)
- UJOINT - a universal (or Cardan) joint (2 DOF). See [https://en.wikipedia.org/wiki/Universal\\_joint](https://en.wikipedia.org/wiki/Universal_joint)
- GIMBAL - 1 gimbal with 3 rotational axes such as is typically used to mount a gyroscope (3 DOF). <https://en.wikipedia.org/wiki/Gimbal>
- BALL - a rotational joint with 3 degrees of freedom (3 DOF). The rotation is represented by a quaternion.
- SLIDER - a translational joint with one axis of translational freedom (1 DOF). Does not support any rotation. Sometimes called a prismatic joint.
- PLANAR - a translational joint with two axes of translational freedom (2 DOF). Does not support any rotation.
- TRANSLATIONAL - a translational joint with three axes of translational freedom (3 DOF). Does not support any rotation.
- COMPOSITE\_TRANSLATIONAL - a translational joint with three translational degrees of freedom. However, the three axes of translation are assumed to be the x-y-z axes (3 DOF). Does not support any rotation. In some situations a joint with both full translational and rotational degrees of freedom can be represented by a COMPOSITE\_TRANSLATIONAL joint followed by a BALL joint.
- FULL6DOF - a joint with full translational and rotational freedom (6 DOF). Joint rotations are represented by quaternions, but joint rates are represented by vectors of 3 values. In FULL6DOF joints, the attitude and velocity is represented in the child body frame.

- **FULL6DOF\_INERTIAL** - a joint with full translational and rotational freedom (6 DOF). Like in FULL6DOF joints, joint rotations are represented by quaternions, but joint rates are represented by 3-vectors. In FULL6DOF\_INERTIAL joints, the attitude and velocity is represented in the parent body frame (which is often the inertial frame, hence the name).

Other types joints can be made up of combinations of these types of joints. For instance a cylindrical joint, which as 1 translational DOF and 1 rotational DOF could be constructed by combining a SLIDER and a PIN joint.

## Ndarts Prescribing and Masking Joints

### 6.6.3. Ndarts Prescribing and Masking Joints

In Ndarts, prescribing and masking a joint are completely independent concepts:

- **Prescribing** a joint controls how the Ndarts dynamics algorithms compute the torques and accelerations for the joint.
- **Masking** a joint hides (masks) selected hinge states for the joint from the Dshell integrator (e.g., for overriding). Usually Dshell models (and not the integrator) manage the values in these masked states. Masking controls whether joint position/velocity states come from the integrator (which is the normal case) or are a set directly via a “deus ex machina” mechanism by the model (or user).



#### Note

Note that masking is related to the integrator and is not an Ndarts concept at all, but is covered together with prescribing because of their similarities.

For details about when to prescribe a joint and when to mask a joint, please see [Ndarts\\_Prescribing\\_Masking\\_Choice](#).

## Prescribing Joints in Ndarts

### 6.6.3.1. Prescribing Joints in Ndarts

In Ndarts, prescribing a joint controls how the dynamics engine computes the joint torques and accelerations.

#### 6.6.3.1.1. Unprescribed Joints

In normal operation when a joint is not prescribed, typically models will inject forces into the joint and Ndarts will compute the corresponding joint accelerations as shown in this figure:

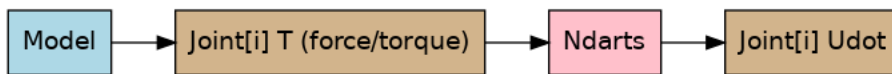


Figure 67. Normal operation of joint (NOT prescribed)

In summary, **if a joint is not prescribed:**

- Joint forces/torques determine joint acceleration (using Ndarts)
- Joint velocity/position come from integration of joint acceleration (Udot)

#### 6.6.3.1.2. Prescribed Joints

If a joint is prescribed, some model (or the user) sets the joint acceleration and Ndarts will compute the necessary joint torque to achieve the prescribed acceleration as shown in this figure:

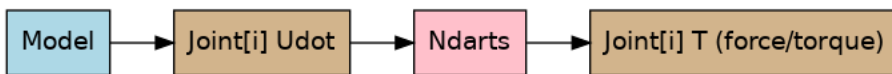


Figure 68. Effect of prescribing a joint

In summary, **if a joint is prescribed:**

- Joint acceleration determines joint forces/torques (using Ndarts)
- Joint velocity/position come from integration of joint acceleration (Udot)

#### 6.6.3.1.3. Setting the Joint Kinematics for a Prescribed Joint

The intent of prescribing a joint is for the models (or the user) to specify the joint acceleration and let the Ndarts dynamics compute the necessary joint torques. There is also an expectation that the integrator will integrate the given joint acceleration to compute joint velocities and joint positions. Note that the integration of the joint coordinates also depends on initial joint velocity and acceleration (which are usually set during initialization). There are several options for dealing with prescribed joints:

- **Specify joint acceleration at run-time.** Based on the initial joint velocity and position, the integrator will behave normally and produce smooth and consistent dynamics for the joint motion. The models or user can set the joint acceleration of a prescribed joint at any time during integration, including in the middle of integration steps using `preDeriv()` calls (in the same way that models can set joint torques at any time).

The code to set the joint acceleration looks like this in the C++ model code (note that `setGenAccel()` is a function of the C++ Model class, not the subhinge):

```
setGenAccel(hinge().subhinge(0), accel);
```

where the joint's velocity and position are initialized elsewhere.



#### Warning

Never set the joint acceleration directly through the joint subhinge API at run-time in models. Always use the model code call shown above since it checks the correctness of the call.

- **Specify the initial joint velocity and set the joint acceleration to zero.** In order to produce a prescribed motion that is constant velocity, the joint acceleration should be set to zero and the joint velocity should be initialized to the desired value. This might be useful for a model that sets up a constant-velocity ejection of one body from another. For example this code might be called in a model's init code:

```
setGenCoord(hinge().subhinge(0), initial_joint_position);
setGenVel(hinge().subhinge(0), desired_joint_velocity);
setGenAccel(hinge().subhinge(0), zero);
```



**Warning**

It is important to note that the `setGenVel()` function cannot be called in the middle of integration steps since it modifies the integrator state (the integrator must be reset). So updating the joint velocity cannot be done in `preDeriv()` calls. It may be called in the `startIntegrationStep()` or `endIntegrationStep()` functions.

- **Specify the initial joint position and set the joint velocity and acceleration to zero.** This might be useful to fix the position of a body with respect to its parent and determine the torques necessary to accomplish this. For example this code might be called in a model's initialization function:

```
setGenCoord(hinge().subhinge(0), initial_joint_position);
setGenVel(hinge().subhinge(0), zero);
setGenAccel(hinge().subhinge(0), zero);
```

6.6.3.1.4. Other Notes about Prescribing a Joint

- Prescribing a joint applies only to subhinges. Although a joint may be composed of several subhinges, prescribing occurs only at the subhinge level. There are some convenience functions in the body parameter code that allows setting a 'prescribed' for a composite joint to a single value (e.g., *True*), but the underlying code applies this same setting to all of the composite joint's subhinges.
- Setting the joint acceleration via `setGenAccel()` function calls in the model code can be done at any time with little computational impact. It is as efficient as setting the joint torque (for non-prescribed joints).
- Changing whether a joint is prescribed has very little overhead in *Ndarts* since the size of the state vector is not changed and no unlocking/locking is necessary.

6.6.3.2. Masking Joints

In some cases, we want the joint to behave in a specific kinematic fashion irrespective of the integrator. In these cases, we can 'mask' the joint from the integrator, essentially overriding the integrator for these joint states as necessary to get the desired behavior:

There are two forms of *masking*:

- Joint masking in U (joint velocity)
- Joint masking in Q (joint position)

It is important to understand that masking is essentially a method of overriding the integrator and is not part of the dynamics model at all. This is why masking cannot be set while initializing the *Darts* body parameters.

Note that there is nothing exclusive to *Narts* about this concept. Masking also applies to *Darts++*.

6.6.3.2.1. Masking a Joint in Velocity (U)

When a joint is masked in joint velocity (U), some model in the simulation will override the joint velocity (U) state during normal execution. Here is what happens:

- The integrator ignores the joint acceleration (Udot)
- The joint acceleration (Udot) is as computed by *Ndarts*.

Note that the overridden U and the Udot from *Ndarts* are not generally consistent. The joint position comes from the regular integration of joint velocity Qdot.

This figure illustrates the basic concept:

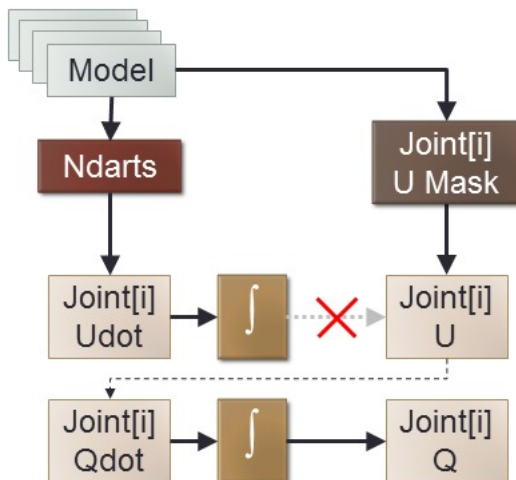


Figure 69. The effect of masking joint velocity (U)



Note that the model overrides the joint velocity. Therefore the joint acceleration (Udot) and joint velocity (U) will not generally be consistent. But the joint velocity specified by the masking process is integrated normally, therefore the joint velocity and position will be consistent with each other.

The function calls to set masking data for masking in joint velocity (U) looks something like this in the C++ model code:

```
coordMaskData(hinge().subhinge(0), joint_velocity, NULL);
```

The NULL argument above can be replaced with the desired joint acceleration (Udot) if it is known.

Note that coordMaskData() function calls are not part of the Darts hinge API and must be called via the C++ model API as shown.

### 6.6.3.2.2. Masking a Joint in Position (Q)

When a joint is masked in joint position (Q), some model in the simulation will override the joint position (Q) during normal execution. Here is what happens:

- The integrator ignores the joint velocity (Qdot)
- The joint acceleration is as computed by the Ndarts.

Note that the overridden Q and the Qdot are not necessarily consistent unless the model has used consistent values.

This figure illustrates the basic concept:

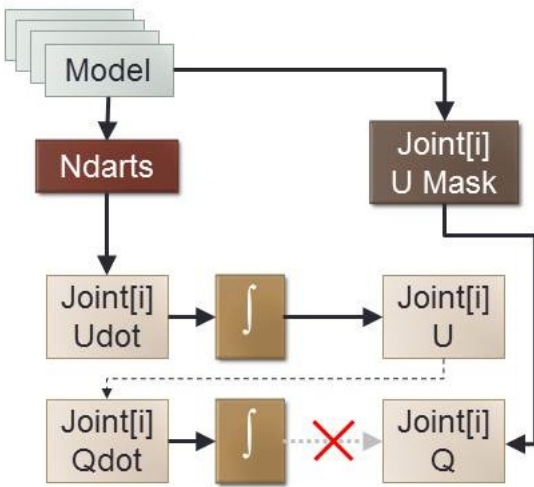


Figure 70. The effect of masking joint position (Q)

Note that the model overrides joint position. Therefore the joint velocity (Qdot), and the joint position (Q) will not generally be consistent and compatible.

The function calls to set masking data for masking in joint position (Q) looks something like this in the C++ model code:

```
coordMaskData(hinge().subhinge(0), joint_position, NULL);
```

The NULL argument above can be replaced with the desired joint acceleration (Udot) if it is known.

### 6.6.3.2.3. Masking a Joint in Position and Velocity (Q and U)

When a joint is masked in joint position and the joint velocity (Q and U), some model in the simulation will override the joint position (Q) and the joint velocity (U) during normal execution. Here is what happens:

- The integrator ignores the joint acceleration (Udot) and velocity (Qdot)
- The joint acceleration is as computed by the Ndarts.

Note that U and the Udot from Ndarts are not consistent. Similarly, the Q and the Qdot are not necessarily consistent unless the model has used consistent values.

This figure illustrates the basic concept:

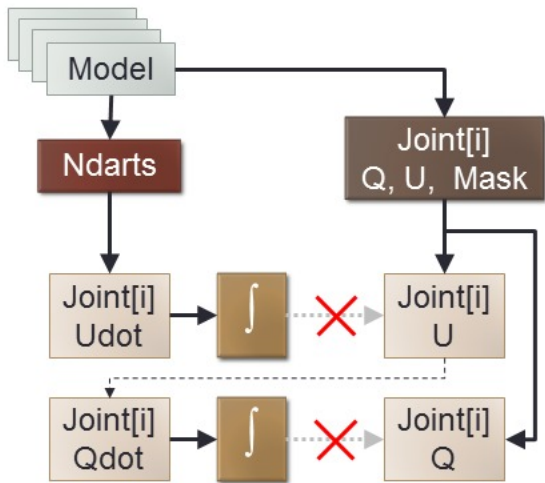


Figure 71. The effect of masking joint position and velocity ( $Q$  and  $U$ )

Note that the model overrides both the the joint velocity and position. Therefore the joint acceleration ( $Udot$ ), the joint velocity ( $U$ ), and the joint position ( $Q$ ) will not generally be consistent and compatible.

The function calls to set masking data for masking in joint position ( $Q$ ) and joint velocity ( $U$ ) looks something like this in the C++ model code:

```
coordMaskData(hinge().subhinge(0), joint_position, joint_velocity, NULL);
```

The NULL argument above can be replaced with the desired joint acceleration ( $Udot$ ) if it is known.

#### 6.6.3.2.4. The Masking Interface

Since masking is tied to integration, the API for changing joint masking is also tied to the integrator. There is a masking interface object that can be obtained from the simulation and masking in  $Q$  can be set up for a subhinge (in python) like this:

```
from Dshell.DartsCommon import DartsBody, DshellStateMaskingIF
sim.dynCoordMaskingInterface().coordMaskType(subhinge, DshellStateMaskingIF.MASK_Q)
```

where `sim` is the simulation object.

In C++ model code, comparable code should be put into the model's `setup0` function and might look like this:

```
coordMaskType(hinge().subhinge(0), DartsIF::MASK_Q);
```

#### 6.6.3.2.5. Other Notes about Masking a Joint

- All masking is done at the subhinge level.
- Changing the coordinate data for a masked joint is an efficient operation and can be done by models during normal execution without significant run-time penalty. This is normally done using a models 'coordMaskData()' function.
- Note that changing the coordinate mask data should only be done at integration sub-step boundaries and not in `preDeriv0` calls. The reason is that `coordMaskData0` calls essentially change the integrator state without the knowledge of the integrator. Typically, the `coordMaskData0` calls would be done in the model's `startIoStep0` or `updateFlowOuts0` functions.
- Changes in the masking state are actually requests that are accumulated and implemented right before the next integration sub-step. So calls to the `coordMaskType0` function can be done at any time, but will only take effect at the next integration sub-step.
- Changing the masking state of a joint involves more overhead and should not be done during normal integration since it requires that the integrator be reset and the state size changed after it is done.
- Currently the setting up the masking states of joints is done outside the model (e.g., in the assembly). But it makes sense to move this into the model code since the model is the only place where the masked joints can be updated.

#### 6.6.3.3. When to Prescribe a Joint and When To Mask a Joint

In a certain sense, both prescribing and masking a joint produce similar effects: they make the joint move like we want it to. When should you prescribe a joint and when should you mask it?

If all the following conditions apply, use 'prescribing':

- Smooth dynamics are desired
- Consistent joint kinematics (position, velocity, acceleration)
- Consistent dynamics and kinematics
- Only joint accelerations change during integration

Otherwise, use masking.

Here are some example scenarios:

- **Body Separation** - At some point a child body is detached from its parent body and a desired constant separation velocity is desired. Use **prescribing**. At the instant of separation, disconnect the bodies, set the desired separation velocity (via `setGenVel()` in model code) and set the desired joint acceleration to zero.
- **Trim control** - In many cases it is necessary to specify the trim angle for a body during flight to match some desired trim history (or via some state-dependent way). Since the only concern is the body attitude and not the necessary torques to produce it, there is no need for the dynamics and kinematics for the body attitude to be consistent. Therefore **masking** is appropriate.
- **Acceleration Specified** - If the desired joint acceleration is given and there are no other limitations, then **prescribing** should be used.

## 6.6.4. Geometric Shapes for Bodies

### 6.6.4.1. Creating shapes for your Ndarts bodies

It is straight-forward to add graphics to your simulations by adding PartGeometry data in the 'geometry' definition of your body parameters.

When you define body parameters using the BodyParam class, you can add in a series of geometrical shapes by specifying a 'geometry' section:

```
from DshellCommon.params.BodyParam import BodyParam
...
'SC' : BodyParam(
    mass = 23.0,
    ...
    geometry = {
        'body' : {
            'shape' : 'cylinder',
            'radius' : 1.0,
            'height' : 3.0,
            'scale' : (1, 1, 1),
            'translation' : (0, 0, 0),
            'quaternion' : (0, 0, 0, 1),
            'diffusivity' : (0.6, 0.1, 0.1)
        },
        'body2' : {
            'shape' : 'cube',
            'length' : 0.2,
            'width' : 0.15,
            'height' : 0.15,
            'diffusivity' : [0.35, 0.30, 0.50],
            'scale' : (1, 1, 1),
            'translation' : (0, 0, 0),
            'quaternion' : (0, 0, 0, 1)
        },
        'body3' : {
            'file' : 'foo.mesh',
            'scale' : (1, 1, 1),
            'translation' : (0, 0, 0),
            'quaternion' : (0, 0, 0, 1)
        }
    }
)
```

where some parts have been omitted for brevity. Notice that multiple geometrical parts can be given. The name of the graphical object is its key (eg. `body` or `body2` here). Each graphics object is a fixed distance and rotation from the body reference frame (as specified in the `translation` and `quaternion` parameters).

#### Example 2. Danger

If you set one of the below parameters and you don't set the associated parameters, the program segfaults for some reason. For example, this happens when `cylinder` is set, but `height` and `radius` is not set.

The types of common shapes that can be specified (including their required parameters) are:

- **sphere**
  - radius
- **cylinder**
  - radius (x and z)
  - height (y)
- **cone**
  - radius (x and z)
  - height (y)
- **cube**
  - length (x)
  - width (y)
  - height (z)
- **file**

To specify a mesh file instead of a primitive shape, use the `file` parameter instead of the `shape` parameter (as shown for `body3` above). The `file` parameter takes a mesh filename as its value. When using `file`, you may not be able to modify the color of the object as that is usually embedded in the mesh file. See `Ndarts_ShapeMeshFileFormats` below for more details about the supported file formats.

All of these graphics objects accept the following optional parameters:

- **scale** - a tuple/list of 3 values describing how to scale the:: object in the x, y, and z directions (body reference frame).
- **translation** - a tuple/list of 3 values (dx, dy, dz) of the:: offset between the body reference frame and the center of the graphics object.
- **quaternion** - a tuple/list of 4 values of the quaternion for the:: rotation between the body reference frame and the graphics object.
- **diffusivity** - a tuple/list of three values (0 to 1.0) for the:: Red, Green, and Blue color of the object. Note that this does not apply for meshes (which must provide their own coloring).
- **specularity** - a tuple/list of three values (0 to 1.0) for the:: Red, Green, and Blue specularity (shininess) of the object. Note that this does not apply for meshes (which must provide their own coloring).
- **emissivity** - a tuple/list of three values (0 to 1.0) for the:: Red, Green, and Blue emissive color of the object. Note that this does not apply for meshes (which must provide their own coloring).
- **transparency** - a single value (0 to 1.0) describing the:: transparency of the graphics object (1.0 is completely transparent, 0 is not transparent at all).
- **visibility** - a single value (0 or 1) indicating whether this:: graphics object should be visible initially.

#### 6.6.4.1.1. Mesh File Formats

The `file` parameters can be an OGRE `.mesh` file or a Wavefront `.obj` file. If you want other types, use a Python file to point to them. For example `example.py`:

```
{'wavefront': 'foo.obj',  
'ogre': 'foo.mesh'}
```

Then you would set `file` to `example.py`.

Part geometries support a few other 3D graphics file formats by using the `assimp` library to load the 3D graphics data from mesh files. (See [http://www.assimp.org/main\\_features\\_formats.html](http://www.assimp.org/main_features_formats.html)). The currently supported files are:

- **OGRE**, use:
  - `'ogre': '<filename>.mesh'`
- **Collada**, use:
  - `'collada': '<filename>.dae'` *or*
  - `'collada': '<filename>.xml'`
- **Wavefront Object**, use:
  - `'wavefront': '<filename>.obj'`
- **3ds Max 3DS**, use:
  - `'3ds': '<filename>.3ds'`

If you have other file formats, the `assimp` tool can be used to convert to one of these supported formats. See <http://www.assimp.org> for more information.

#### 6.6.4.2. Manipulating a graphical object at runtime

You may manipulate graphical objects at run time. First get its handle. Suppose `sc` is the body object that these graphics objects are attached to.

```
from Math.SOA_Py import SOAVector3  
...  
cyl = sc.getPartGeometry('body')  
cyl.scale(SOAVector3(0.5, 0.5, 0.5))
```

PYTHON

This rescales the size of the graphics object to be half what it normally is (with `scale=(1,1,1)`).

Most of the other properties outlined above may be modified (the names may be a bit different; browse a live object to find what you need).

You may also active the axes for a graphical object:

```
cyl.showAxes(True)
```

PYTHON

#### 6.6.4.3. Other documentation

For Doxygen documentation, see: `PartGeometry`.

#### 6.6.5. Multibody Modeling with Ndarts - Design

The Ndarts library provides classes and algorithms for the kinematics and dynamics of rigid multibody systems for use within the Dshell simulation framework. It is a successor to the Darts++ module.

#### 6.6.5.1. Background

The objective of this new multibody kinematics/dynamics implementation is motivated by several issues with the current "Darts++" implementation, as well as a need and desire to add several new advanced capabilities. The limitations in the current capabilities includes:

- Inverse kinematics, Constraints
  - A need to support new wheel geometries and contact in rover sims (broader wheels, toroidal shape, etc.)
  - The IK implementation with IKGraph is complex (eg. uses Euler coordinates). The NRSolve code is dense and hard to use in its current form.
  - Hard to debug cause when inverse kinematics failed - and so hard to fix issues. Lots of struggles with Chariot wishbone kinematics. Have been tossing around ad-hoc and unsatisfactory work around solutions - weights, staged IK, etc.
  - Need to be able to expand configuration kinematics to allow new constraints on motion (eg. Athlete cases)
  - Ability to handle constraints between multiple vehicles (eg. coordinated motion, docking)
  - Lack of general way to handle constrained/closed-chain dynamics
  - Questions about proper handling of hinge/constraint duality to simplify addition of new constraint types.
  - Unable to do IK on subgraphs within multibody.
  - Hard to bring in contact and collision detection physics.
  - Challenging to "generalize" constraint embedding to handle arbitrary "gearing" and "loop" type constraints (limited to four bar linkage currently)
- Functionality Limitations
  - Accumulation of narrow/ad-hoc algorithmic implementations that are hard to generalize (eg. special handling of full 6dof hinges, topology changes, handling constraints)
  - Hard to add new hinge types (eg. locked, alternate 6dof) hinges - lots of switches and if/thens in code.
  - More complex than necessary multibody creation process - split between Python and C++.
  - Not possible to make arbitrary frame to frame relative transformation, velocity, acceleration queries within multibody system (needed more generally but especially for constraints)
  - Hard to bring in PyCraft like ideas into regular dynamics functionality
- Architectural and Design Integrity
  - Increased entropy in the Darts++ software implementation from changes/hacks that have been made over the years.
  - While the test coverage has improved, the coverage is uneven, making it harder to add and test new changes.
  - While code "works", it is hard to add in new features without endangering existing capabilities.
  - User level methods and internal code Documentation quality is poor.
  - Method naming conventions are uneven.
  - Body and node methods have parallel methods which are named and/or do something differently.
  - Darts++ code split across DartsBase module for backwards compatibility with Darts

The goal is to rethink, redesign, and streamline the current implementation to have a more agile architecture that will address the above, enabling some of the more advanced following capabilities:

- Advanced mbody algorithms (PyCraft ideas)
  - Allow the implementation of new algorithms (eg. innovations, velocity coordinates transformations, and inverse)
  - Support recursions on sub-trees for embedded constraints and augmented bodies, onboard models, ODCA/parallel implementation
  - Support recursions involving different types of propagation operators (eg. rigid bodies, flex bodies)
  - Support multiple operations/recursions on a fragment, and have means to store and keep access to data products so they can be reused in subsequent recursions (eg. correction recursions in augmented constraints implementation, momentum matching, ODCA decomposition and correction, ATBI recursions)
  - Support recursions for state initialization for momentum matching after impact; or for CM drift nulling for MD simulations
  - Support Pycraft algorithms for sensitivities, Coriolis, etc.
- Constraint/collision/contact handling
  - Support configuration changes in the multibody system, i.e. constraints coming and going (eg. for humanoid application, docking, walking)
  - Support addition of body geometry and surface contact constraints
  - Support use in interactive computations for inverse kinematics; closed-chain state projection
- Miscellany
  - Support "internal model control (IMC)" type algorithms for computations within onboard control software.
  - Support body reordering and reversal of bodies (for optimization of closure cuts)
  - Support using alternative integration variables such as for diagonalized dynamics. May need to generalize Qdot to U and converse functions (already doing this for CK?)
  - Support being able to switch between different frame of integration (eg. inter-planetary trajectory, to EDL trajectory, to rover trajectory)
  - Support use of different frame (body, inertial, internally referenced) for equations of motion.

#### 6.6.5.2. Ndarts Design Details

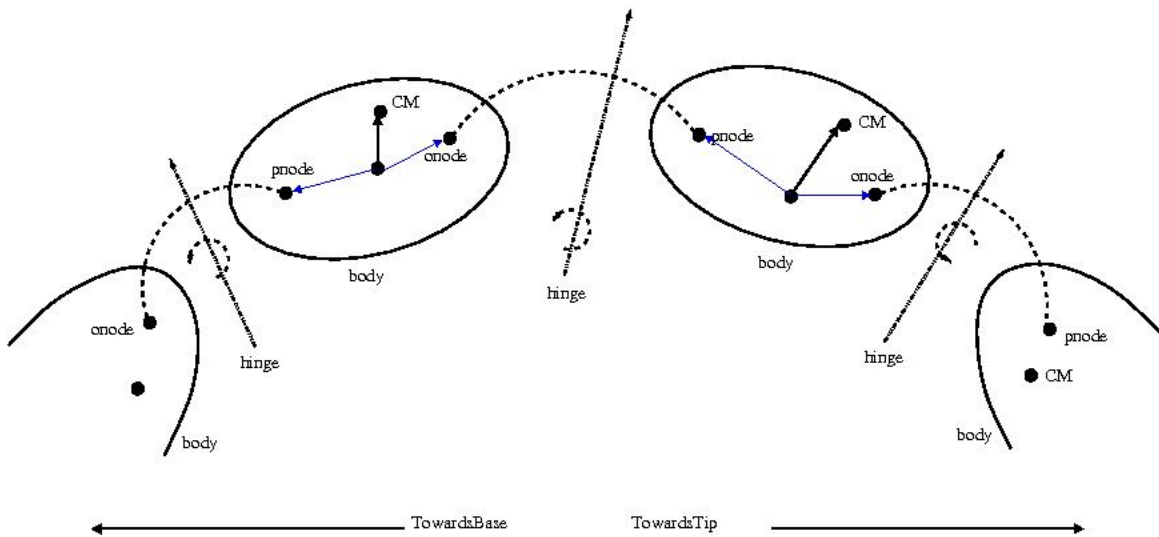


Figure 72. Links and hinges in a multibody system

#### 6.6.5.2.1. Key Requirements

- Code structuring
  - Streamline higher level algorithms by moving out lower level computations into lower level classes so can bring in new algorithms.
  - Keep user level API changes to a minimum.
  - Give low priority to allowing Dshell to work with Darts++ and Ndarts at the same time to avoid constraining design space.
- Kinematics
  - Use the SOAFrames "Frames" class from the "SOA" module to simplify the kinematics computations.
  - Allow automatic queries for relative transformations, velocities, and accelerations across arbitrary points of interest.
  - Avoid unnecessary kinematics computations and overhead.
- Hinges
  - Add support for locked hinge types.
  - Simplify addition of new hinge types.
  - Support run-time user defined hinge types.
- Constraints
  - Support constraint embedding.
  - Support handling of closure constraints.
  - Support handling of surface contact constraints.
- Generalize inverse kinematics to handle much broader class of requirements.
- Support working with system subgraphics.

#### 6.6.5.2.2. Obsolete Modules

- IKGraph
- DartsBase
- TerrainSurface
- SurfaceContact

#### 6.6.5.2.3. API and Functionality Changes

- SOA
  - Changed and renamed all SOAHomTran phi methods
  - Added support for 0 size matrices and vectors
  - Now Frame and Frame2Frame classes are derived from DartsBaseObject
  - Reworked Frame2Frame cache management to allow transform/velocity and acceleration level cache staleness settings
  - Fixed up the matrix SVD and orthogonal complement methods
- Removed and renamed other node, body, hinge methods.
- Moved DartsBaseObject class code to DshellEnv
- Class refactoring
  - Stopped using most base classes from DartsBase
  - Added hinge classes.

- Derive subhinge and higne classes from Frame2Frame.
- New 6dof hinge.
- Derive Node class from Frame.
- DartsBody is now derived from DartsNode class
  - Hence no longer have parallel set of methods for position/velocity of nodes and frames
- DartsHinge only requires frame pair - not node pair to allow use with constraints
- Kinematics
  - evalKinematics, evalActuatorKinematics, evalSensorKinematics methods etc. have been removed.
  - Now do lazy evaluation with caching for transforms, velocities, and accelerations within Frame classes.
  - Extensive reliance on Frames classes.
  - Consolidate DartsBody and DartsNode methods to get position, velocities, etc. In the process some of the methods have been renamed.
  - Changing prescribed flag of a subhinge should only be done when system is unlocked (open to discussion) since constraint related buffer sizes are effected.
  - Added notion of minGenCoord to hinges.
- Added new DShape class for body geometry
- Cosntraints
  - Complete reworking of constraint handling and inverse kinematics.
  - Obsoleted use of IKGraph. Changed IK scheme to more robust rechnique.
  - Added FramePair constraint class to handle any hinge based constraint.
  - Added support for closure and surface shape to surface shape contact constraints.
  - Now surface to Dem constraints is handled same way.
  - Activating/deactivating constraints can only be done when system is unlocked.
  - Inverse kinematics now leaves prescribed motion hinges unchanged.
- Jacobians and Gradients
  - Now can get Jacobian between any frame pair
  - Transform gradient methods to complement Jacobian ones.
  - Now Jacobian methods take extra arguments for skipping prescribed subhinges, and for being maps from generalized velocities of Qdot.
  - Concept of min gen coords for gradients.
- Stricter use of locks
- Now need to unlock to change prescribed status as well as active status of constraints.
- Global vector offsets are tracked for subhinges and constraint hinges.
- Added DartsSubGraph class for a subset of connected bodies.

#### 6.6.5.2.4. Class Hierarchy

- Derive Frame and Frame2Frame from DartsBaseObject class
- Derive DartsNode from Frame class
  - DartsHingePnode (new)
  - DartsHingeOnode (new)
  - DartsActuatorNode (new)
  - DartsSensorNode (new)
  - etc.
- Derive DartsBody from DartsNode class

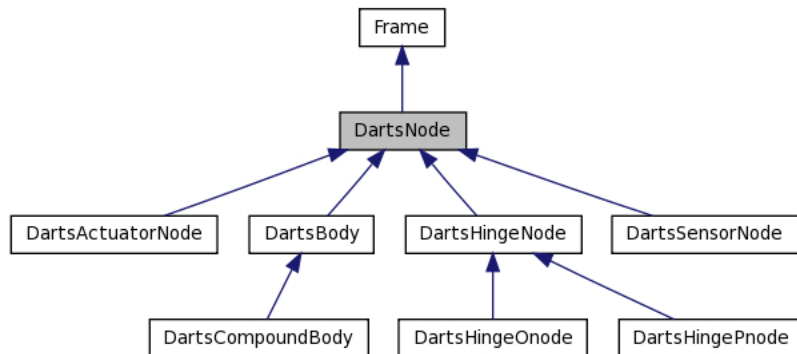


Figure 73. DartsNode classes

- Derive DartsSubhinge from Frame2FrameEdge class
  - DartsLockedSubhinge (new)
  - DartsPinSubhinge
  - DartsLinearSubhinge
  - DartsSphericalSubhinge
  - DartsLinear3Subhinge
  - etc.
- Derive DartsHinge from Frame2FrameChain class
  - DartsPinHinge (new)
  - DartsSliderHinge (new)
  - DartsBallHinge (new)
  - DartsTranslationalHinge (new)
  - DartsFull6DofHinge (new)
  - DartsCustomHinge (new)

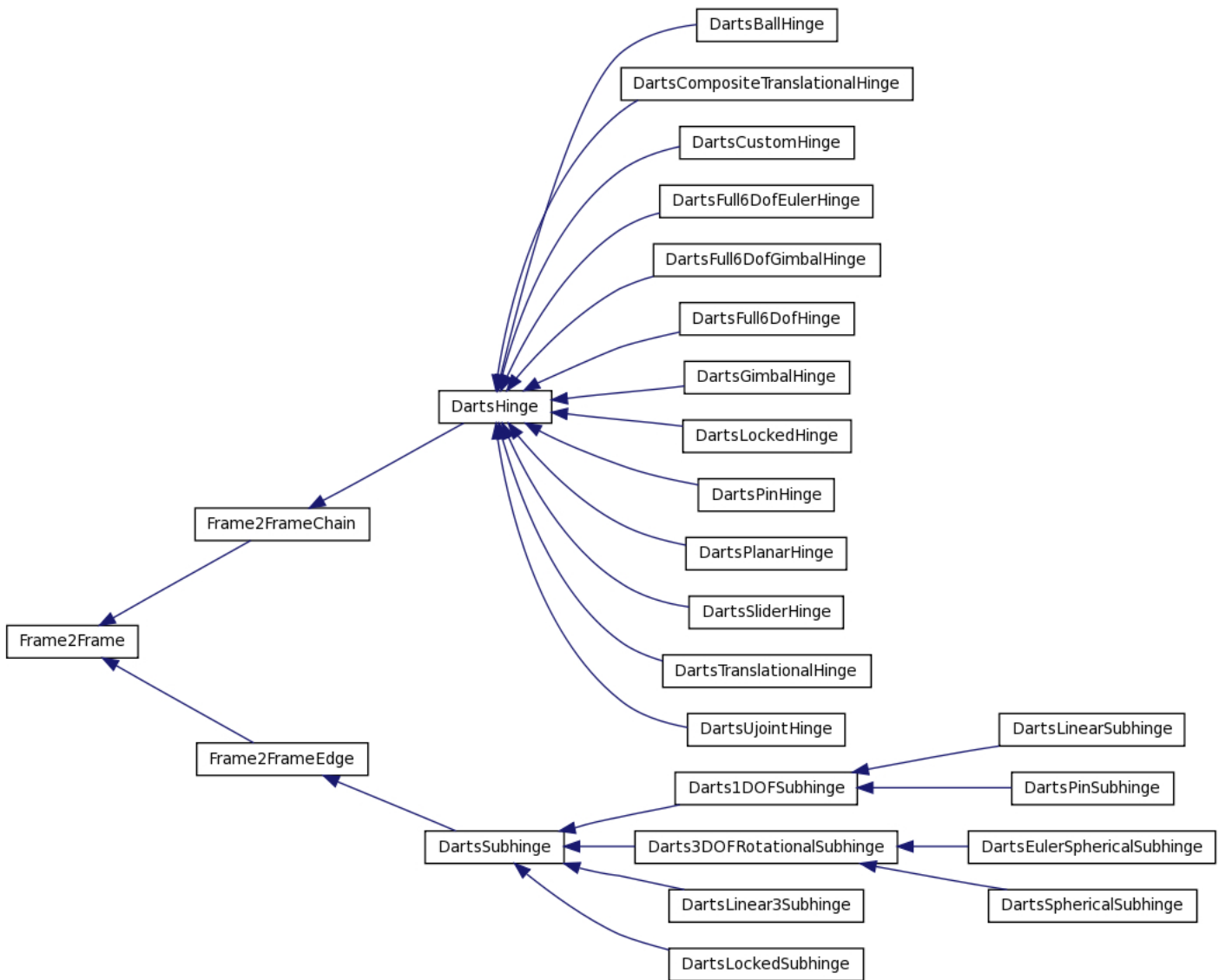


Figure 74. Hinge and subhinge classes

#### 6.6.5.2.5. Body Creation Process

- The root body's frame is connected to the global root frame
- The steps for the creation of a body include
  - Create the body and body frame. Temporarily make the body frame the child of the root frame
  - Create a pnode for the body and (temporarily) make the pnode frame a child of the body frame. Delete the body frame to pnode frame edge and make the body frame a child of the pnode frame. The pnode frame at this point is dangling and not attached to the frames tree.



- Create a 6dof hinge connecting the root body to the new body.
  - Detach the pnode frame from the body frame if it is attached to the tree (so it can be used as a pframe for the sub-hinges).
  - Detach the body frame from its current parent frame if the current parent is not the pnode frame since it needs to be a child of the pnode frame.
  - Create an onode for the body in the root body. Its frame is a child of the root body's frame.
  - Create a sequence of sub-hinges that go from the onode and end at the pnode. At this point, the pnode frame is connected back to the frames tree.
  - Connect the body frame as a child of the pnode frame. Now the body frame is connected to the frames tree.
- Now create the proper joint type by deleting the existing hinge and creating one specified by the joint type and connected to correct inboard body.
  - Delete the existing hinge by deleting all the subhinges and the onode. The subhinge deletion will delete all the frame edges and intermediate oframes. It will leave the pnode frame unattached to the frame tree.
  - Create a new hinge of the correct type between the new parent and the body.

### 6.6.5.3. Body Shape Classes

Each DShape shape represents a parameterized geometry object. Can add a number of geometry objects to a DartsBody instance. Each shape can have its own SOAHomTran offset transform. Usage of DShape classes include:

- Contact kinematics and constraints
- partGraphics
- DBullet interface
- TopoDem terrain vehicle placement, configuration kinematics.
- Imposing task space motion constraints (manipulator, vehicle)
- General mesh surface (eg. rocks, VRML parts) contact.

Non-penetrating contact between shapes requires that the tangent planes at the points of contact be parallel. The dimensionality of a DShape instance is defined by the number of parameters needed to specify the contact point on the surface. The following shape classes are currently defined:

- DShapePoint : 0-dimensional point.
- DShape1D : 1-dimensional surfaces
  - DShapeStraightLine : Linear straight line.
  - DShape1DCircle : Thin circular disc.
  - DShape1DEllipse : Thin ellipse
- DShape2D : 2-dimensional surfaces
  - DShape2DPlane : 2D plane
  - DShape2DSphere : Spherical surface
  - DShape2DEllipsoid : A general ellipsoid
  - DShape2DTorus : A torus
  - DShapeDem : TopoDem surface
  - DShapeMesh : General mesh surface (BD)
- DShape3D : 3-dimensional surfaces with overhangs (TBD)

DShape classes have a DShape::partGraphics Python method to return a part graphics dictionary appropriate for the shape type.

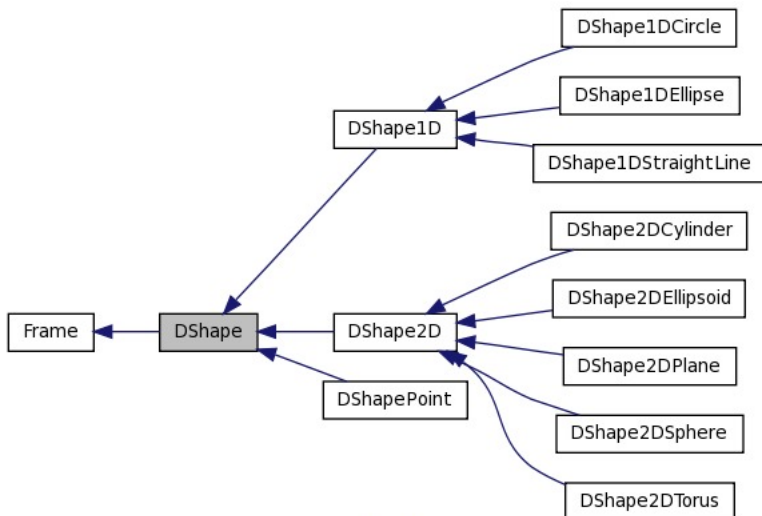


Figure 75. DShape body geometry classes

#### 6.6.5.4. Constraints

Constraint classes are used to enforce kinematics and dynamics constraints on the articulation and motion of the multibody system. The current family of constraint classes is as follows:

- DartsFramePairConstraint : Base class for frame to frame constraints.
- DartsClosureConstraint : Limits motion between a pair of frame defined by a DartsHinge type.
  - Single Node to Frame constraint between a multibody node and an arbitrary Frame (moving or fixed).
  - Node pair constraint between a pair of multibody nodes.
- DartsContactConstraint : Constraint between a multibody node and a DShape surface.
- Vehicle configuration kinematics for system level constraints on vehicle/manipulator placement and motion
  - Place rover
  - KDriver constraint
  - Driving Athlete with level hex
  - Driving Athlete to come to dock
  - Driving 2 Athletes in docked configuration
- Constraint embedding
  - General loop constraints
    - Four bar linkage/wishbone constraint
  - GenCoordsConstraint - algebraic constraint between arbitrary gen coord values (TBD)
    - For local constraints (gearing, differential), use constraint embedding
    - More generally need to handle this
    - Treat prescribed motion as a special case of this constraint on a single hinge
    - Examples
      - Prescribed motion constraint (or joint locking)
      - ATRV driven wheels constraint
      - Differential constraint
      - Gearing constraint

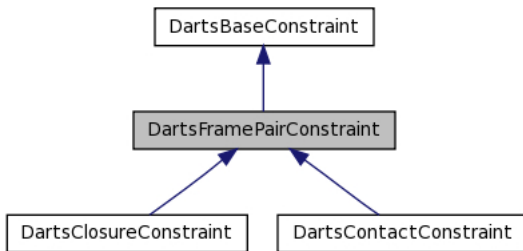


Figure 76. Constraint classes

##### 6.6.5.4.1. Node to Fixed frame Constraint

SingleNode constraint - one node is on a multibody system and the other Frame is not on a multibody (but could be a moving frame). The constraint is expressed as a hinge specifying the allowable motion in the constraint. Enforces constraint relations such as:

$${}^I T_{ec}(\theta) * {}^{ee} T_c(\theta) * T_{offsets} = {}^I T_{tgt}$$

- This constraint takes a node, frame, and hinge as arguments.
- It is simpler than the NodePair constraint in that the Jacobian of only one node needs to be computed.
- Examples
  - Articulate antenna to be pointing towards a moving s/x or the sun
  - Arm inverse kinematics
  - Move arm so the end effector meets a distance/attitude constraint to a desired task frame.

##### 6.6.5.4.2. Node to Node Constraint (dual arm)

NodePair constraints - where both nodes are on multibody systems and there is a hinge specifying the constraint between them. Enforces constraint relations such as:

$${}^I T_{ec}(\theta) * {}^{ee} T_c(\theta) * T_{offsets} = {}^I T_{tgt}(\theta)$$

- This constraint takes a node pair and a hinge as arguments
- Examples
  - Place/move rovers in formation offset by a fixed transform
  - Move arm so its end-effector is a certain distance from a wheel

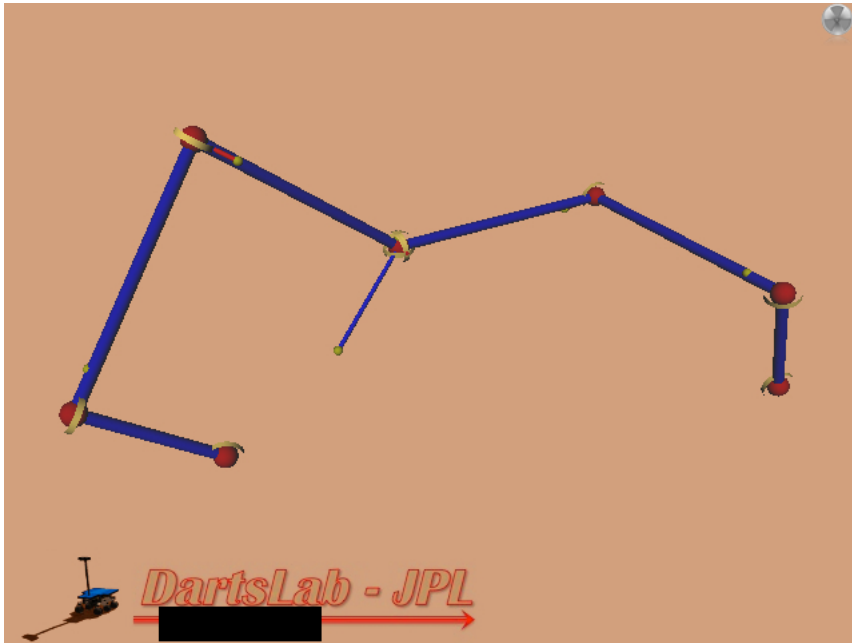


Figure 77. Example of a node to node constraint on dual-arm end-effectors

#### 6.6.5.4.3. Contact constraints

Constraints between a DShape (0, 1, or 2D) and a DShape2D 2D surface

- For a 0d surface, the constraint is that the distance between the surfaces is zero.
- For a 1d surface, the constraint is that the distance between the surfaces be zero, and at a point of contact, the 1d tangent be orthogonal to the 2D surface normal.
- For a 2d surface, the constraint is that the distance between the surfaces be zero, and at the point of contact, the tangent planes for both surfaces be parallel.

The following describes examples of different surface to surface contact constraint

##### 6.6.5.4.3.1. Point 0d contact

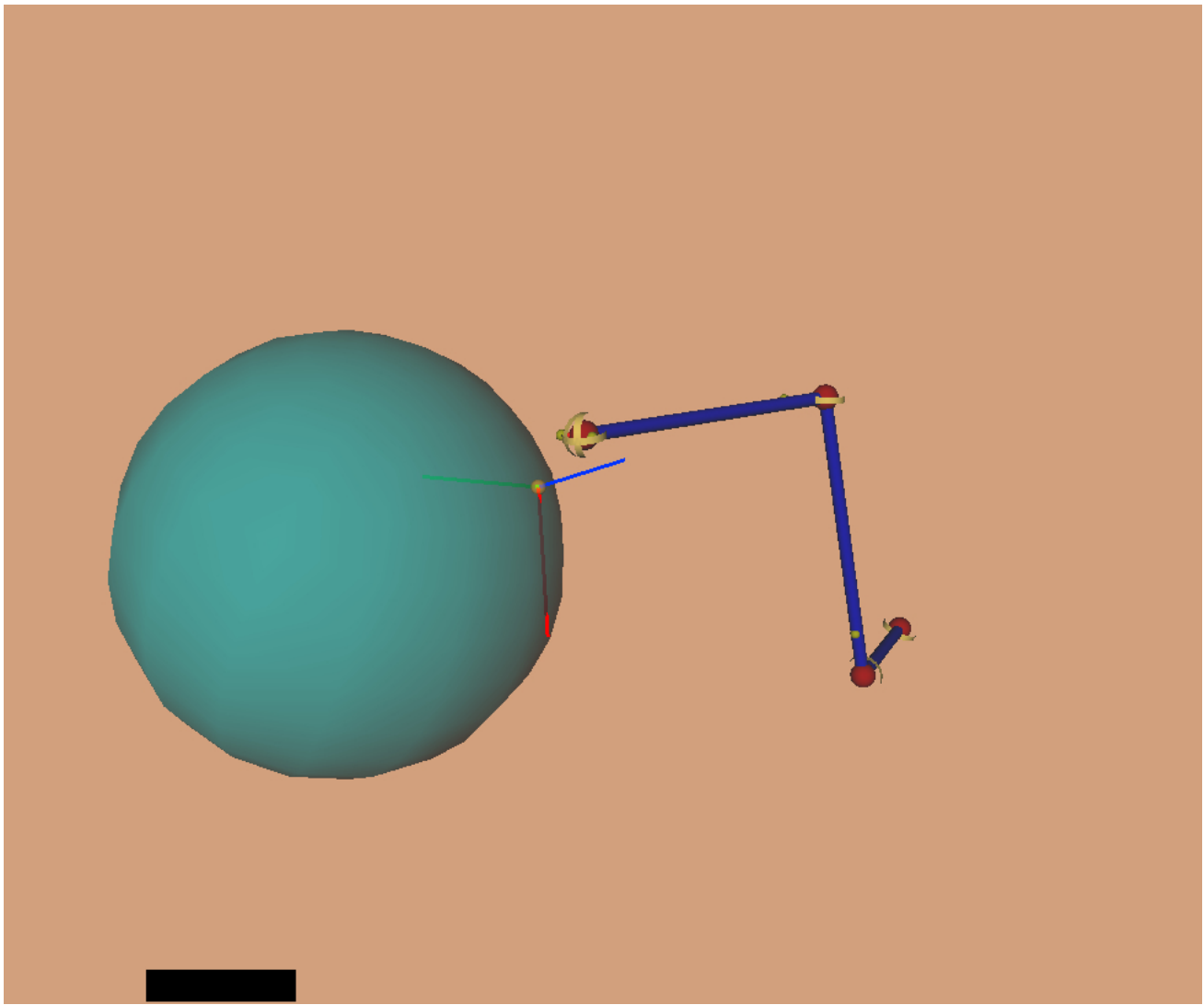


Figure 78. Example of a point contact constraint between the end-effector and a spherical surface

6.6.5.4.3.2. Circle 1d contact

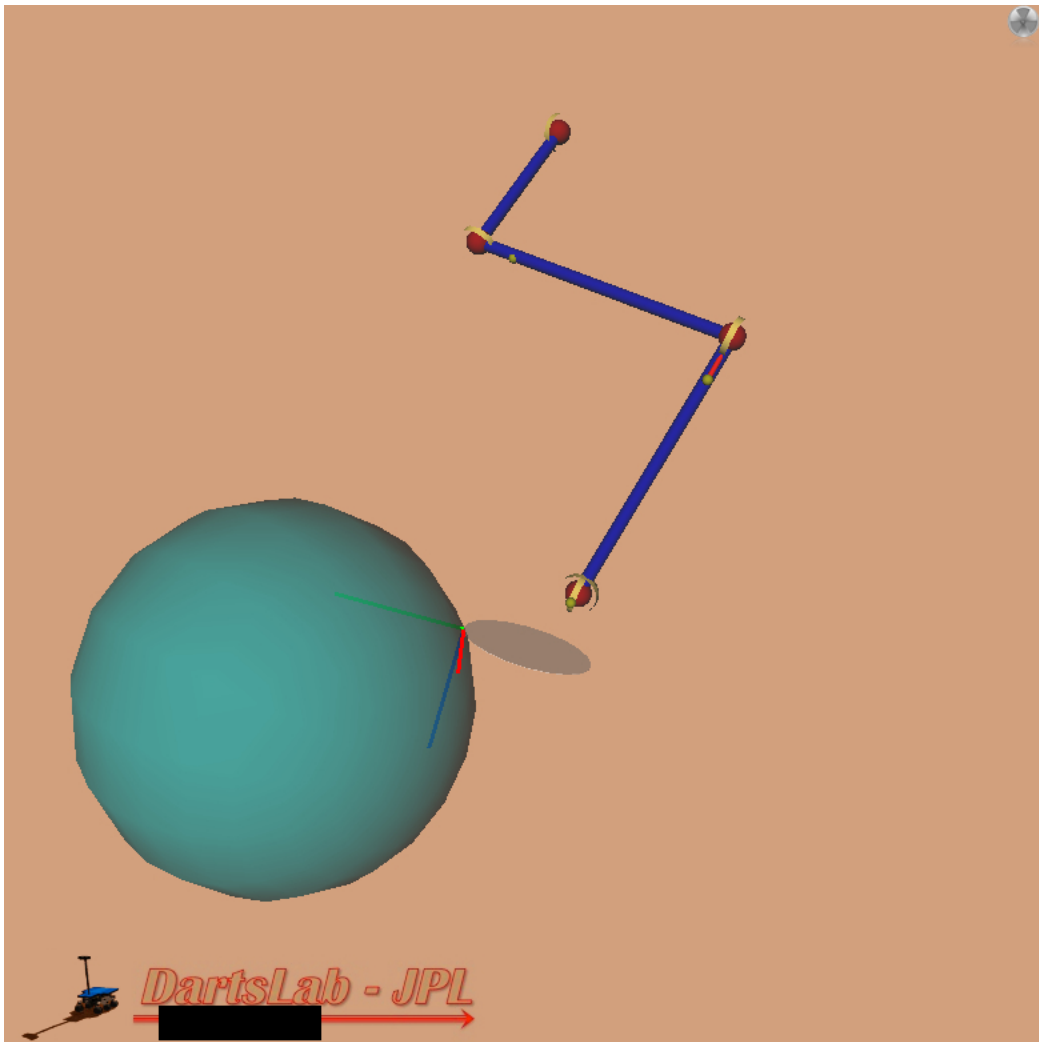


Figure 79. Example of a 1D circle contact constraint between the end-effector and a spherical surface

6.6.5.4.3.3. Ellipse 1d contact

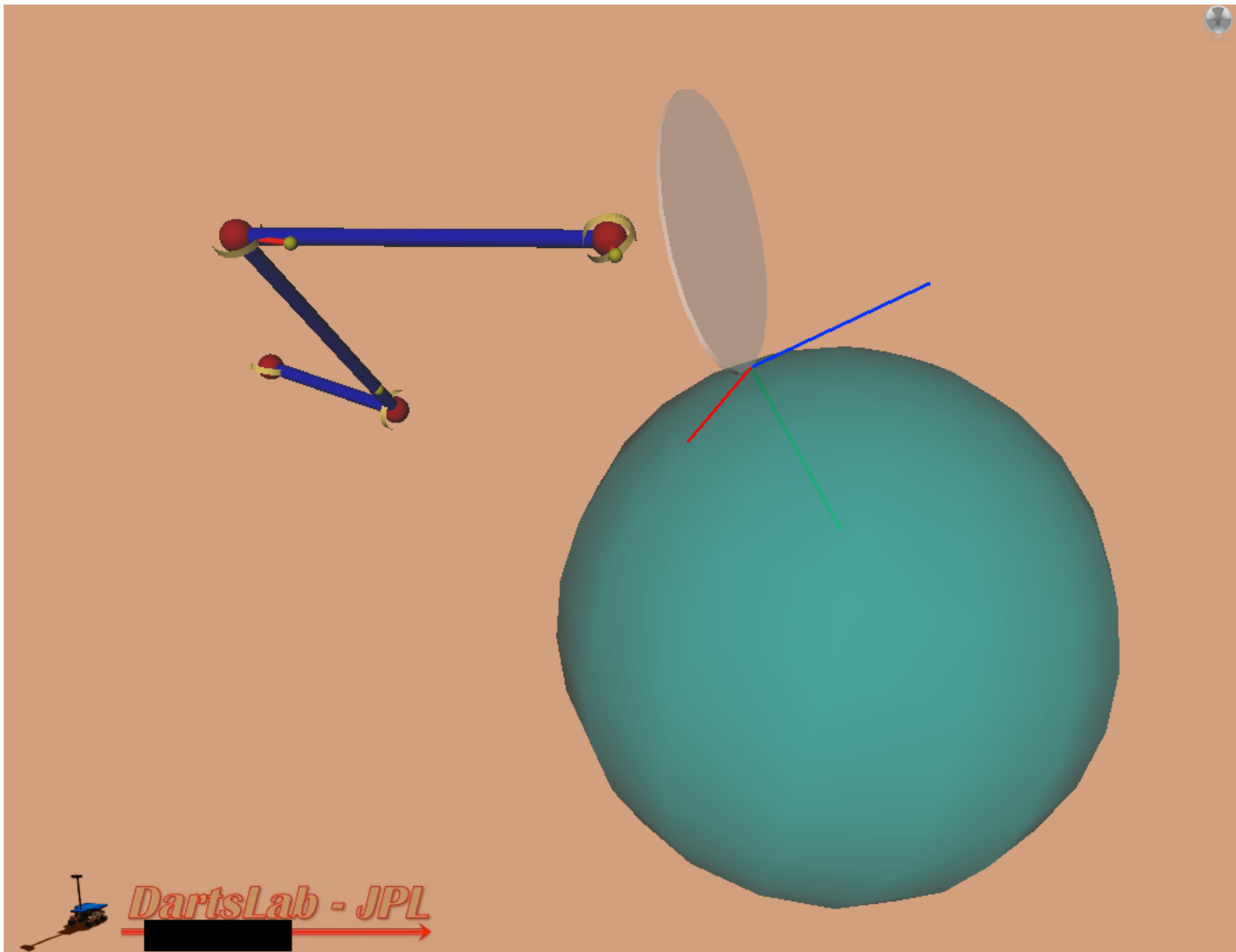


Figure 80. Example of a 1D ellipse contact constraint between the end-effector and a spherical surface

6.6.5.4.3.4. Planar 2d contact

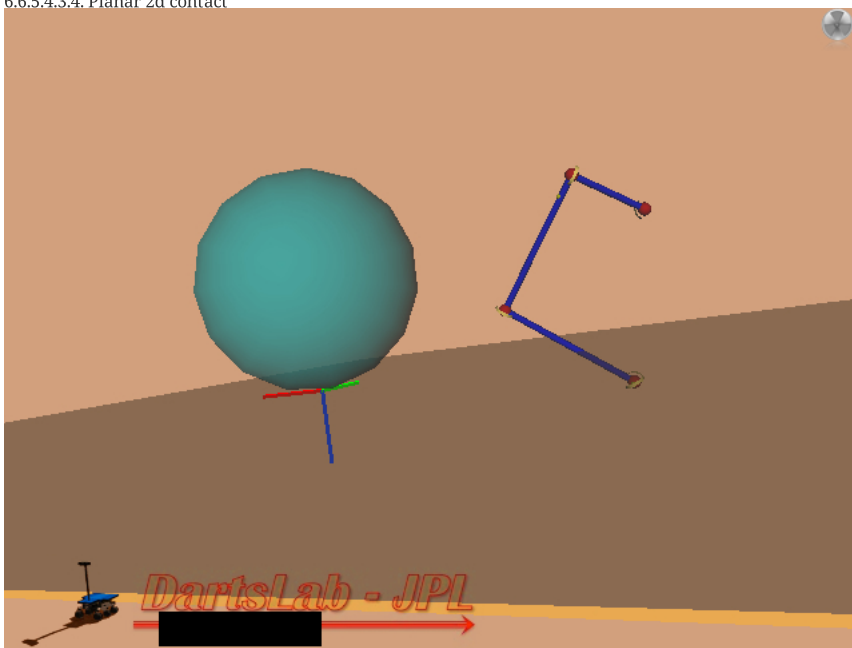


Figure 81. Example of a 2D plane contact constraint between the end-effector and a spherical surface.

6.6.5.4.3.5. Spherical 2d contact

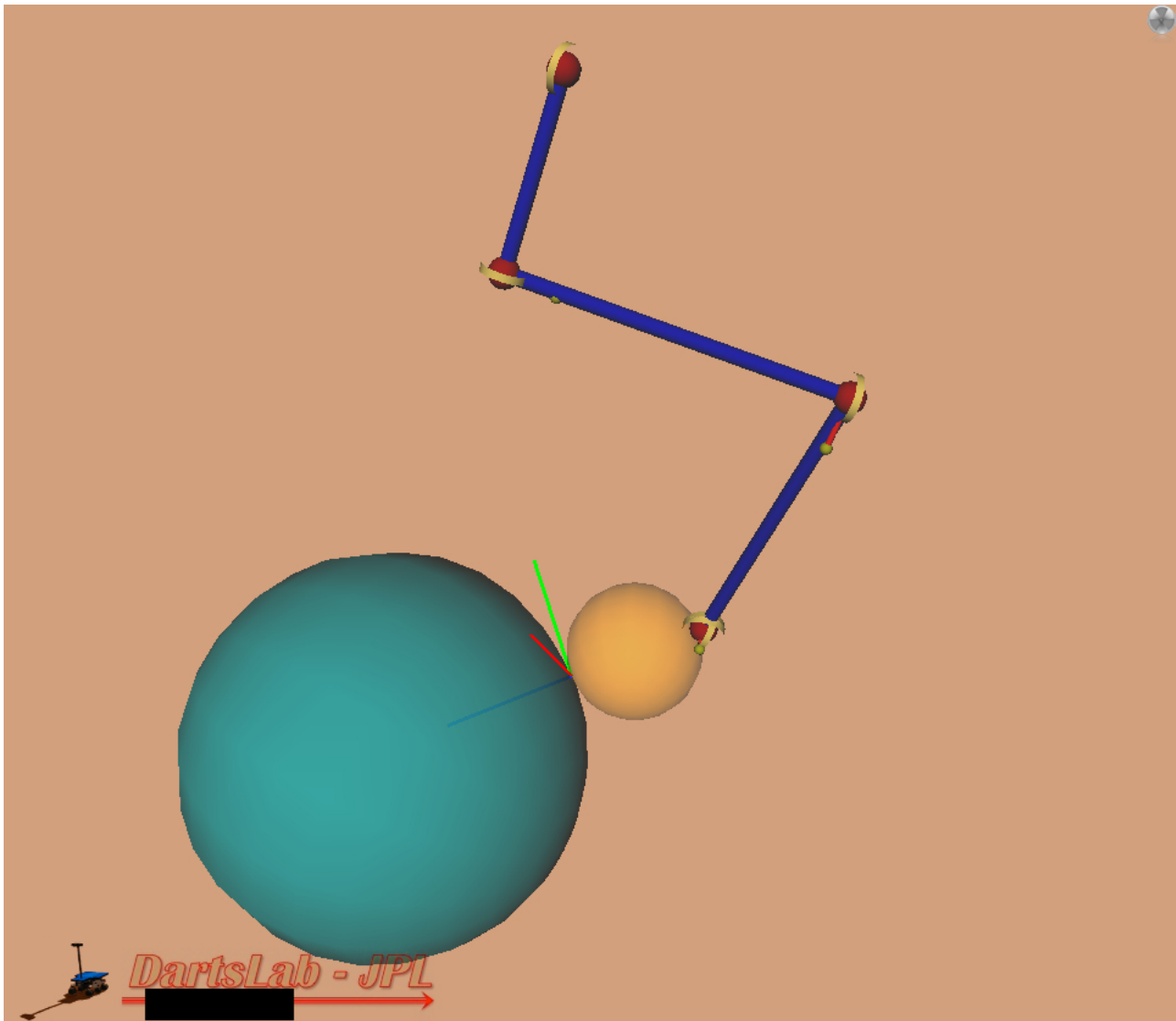


Figure 82. Example of a 2D spherical contact constraint between the end-effector and a spherical surface.

6.6.5.4.3.6. Cylindrical 2d contact

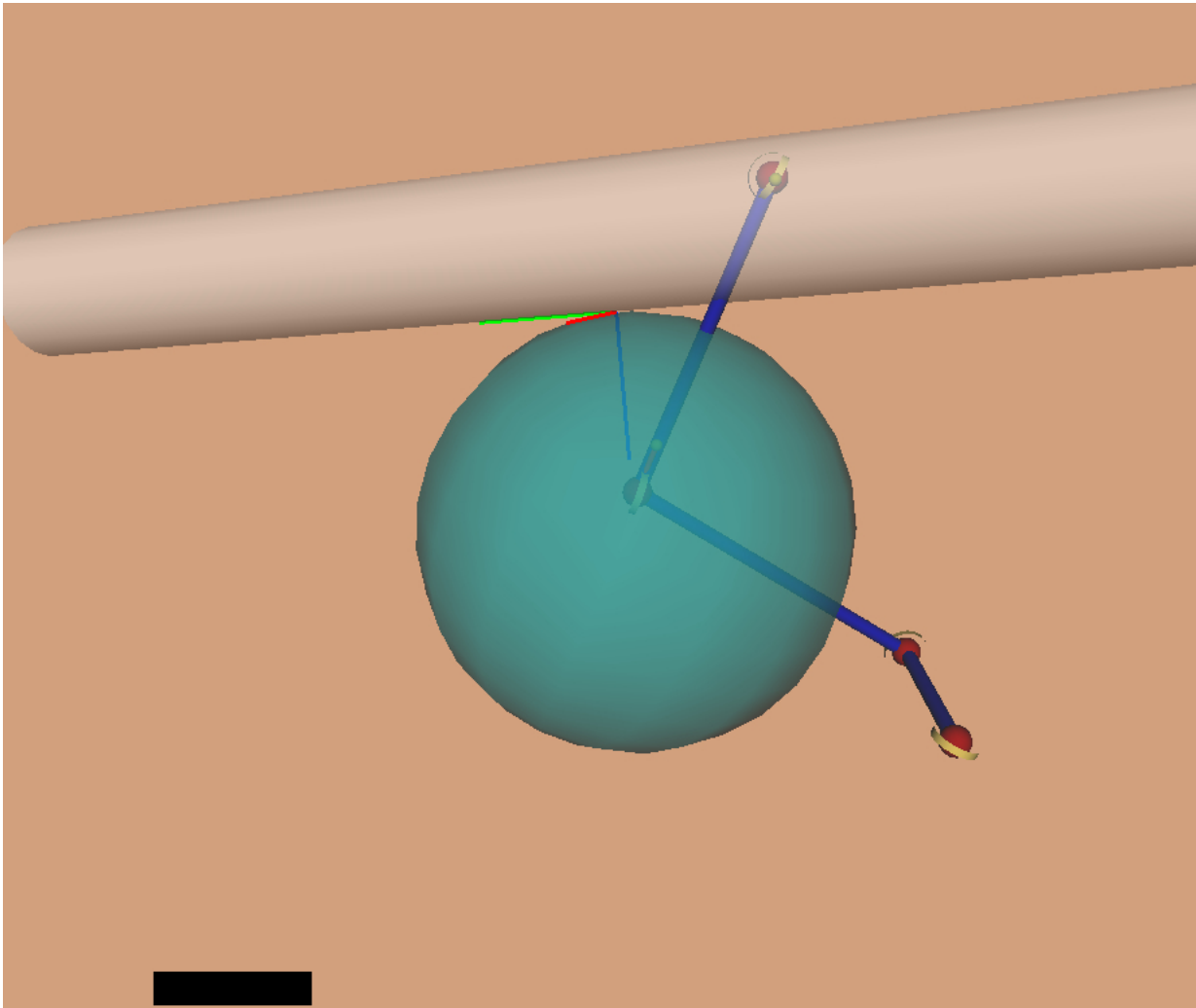


Figure 83. Example of a 2D cylindrical contact constraint between the end-effector and a spherical surface.

6.6.5.4.3.7. Ellipsoidal 2d contact



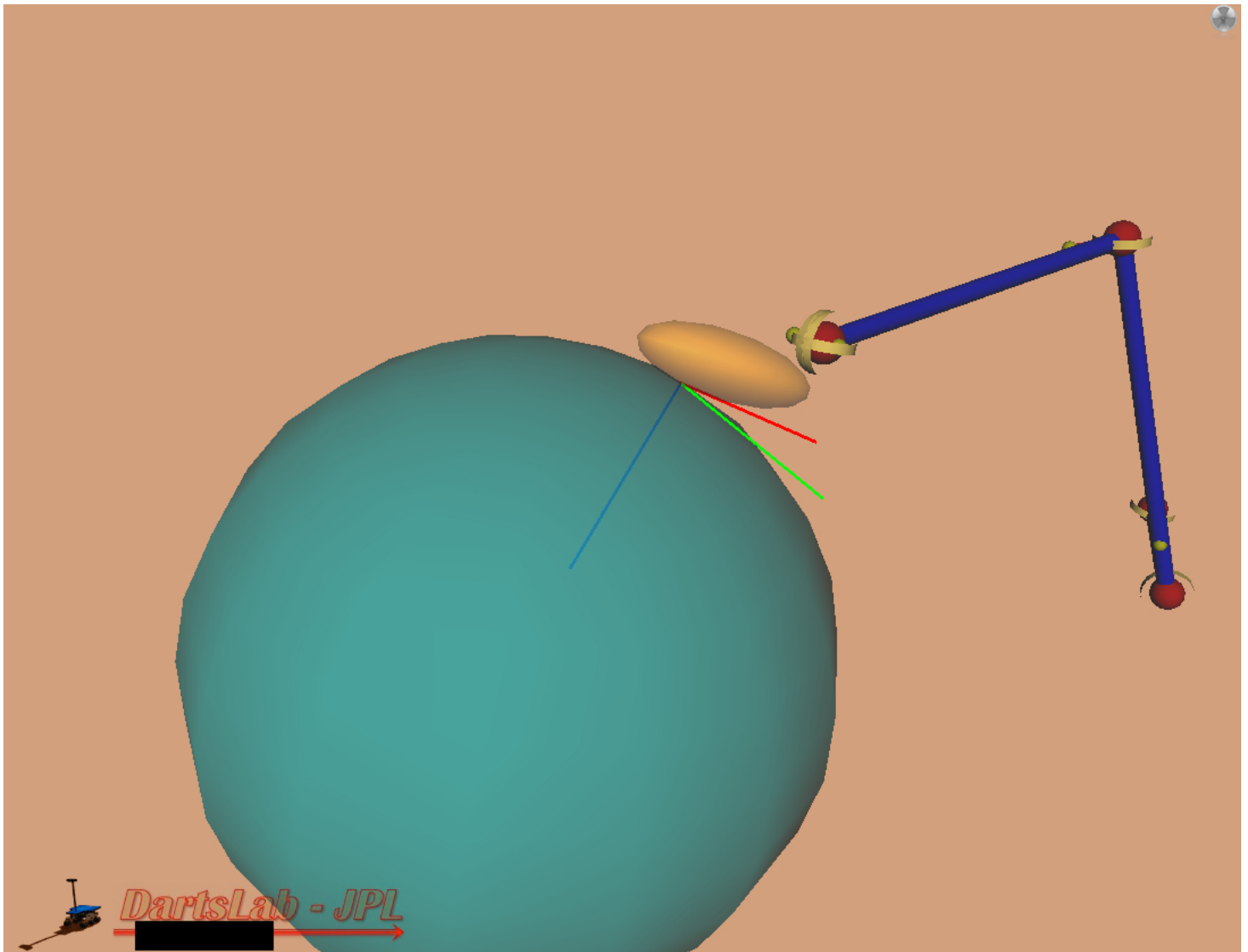


Figure 84. Example of a 2D ellipsoidal contact constraint between the end-effector and a spherical surface.

6.6.5.4.3.8. Toroidal 2d contact

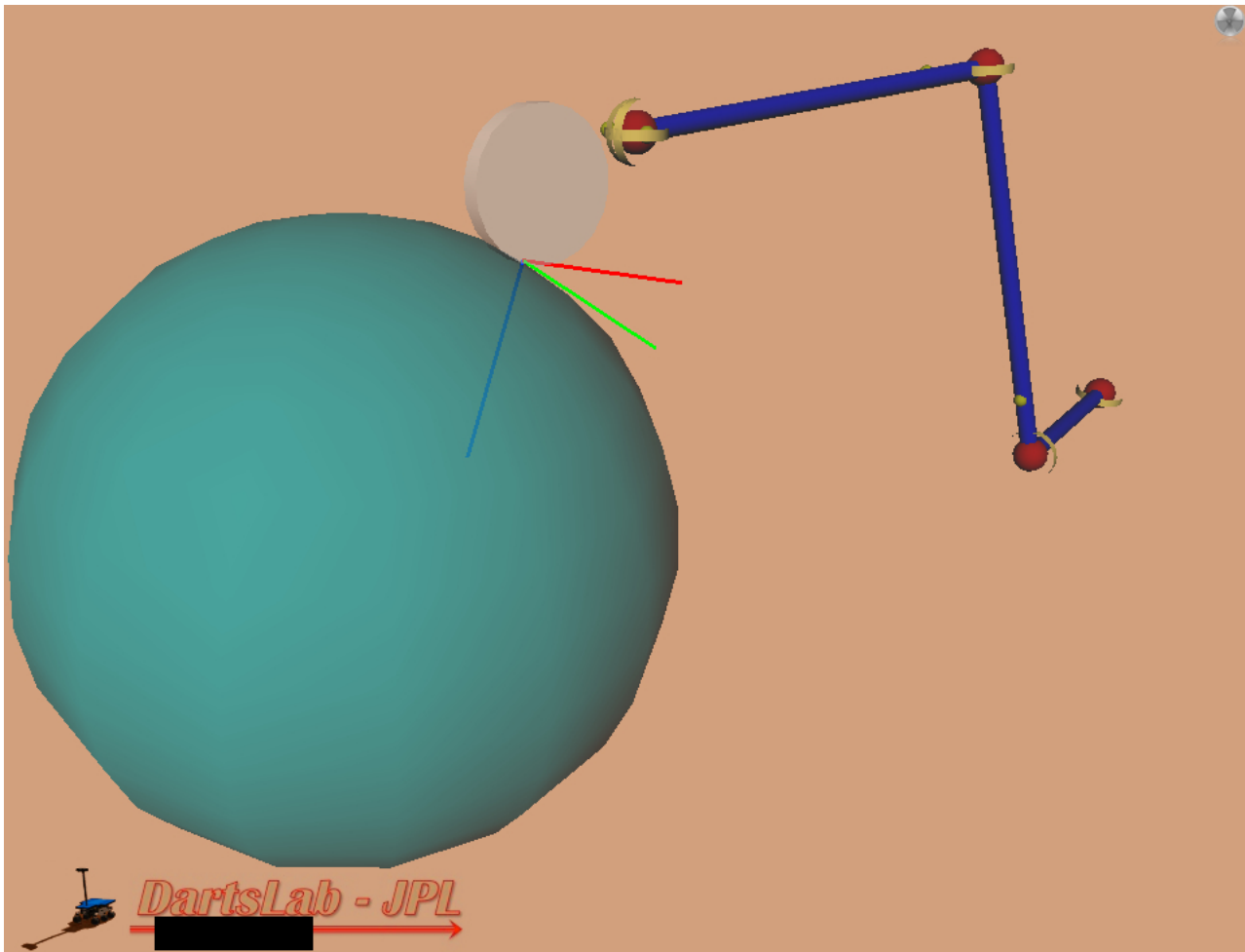


Figure 85. Example of a 2D torus contact constraint between the end-effector and a spherical surface (incorrect torus graphics - needs to be fixed).

6.6.5.4.3.9. TopoDem 2D contact

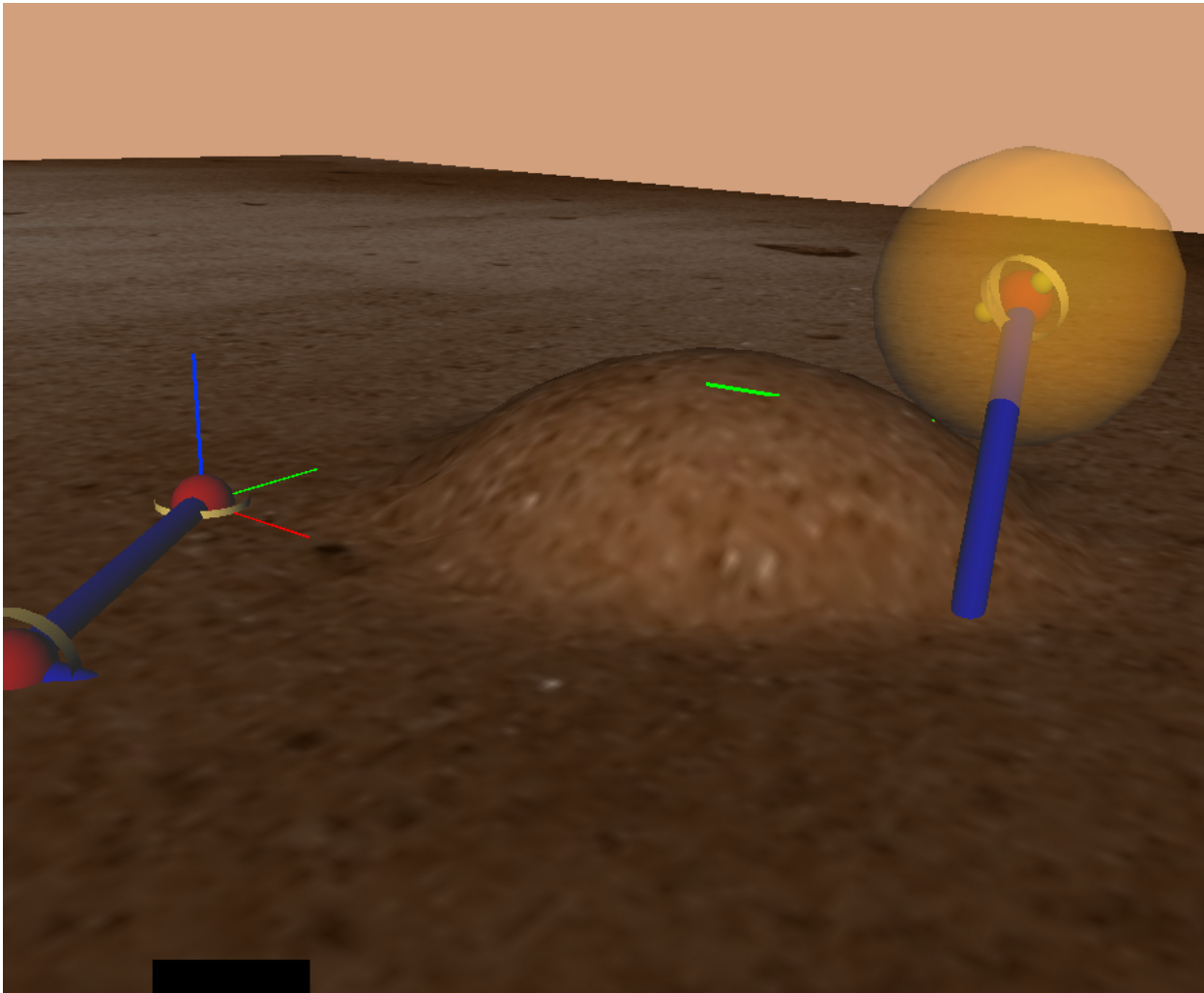


Figure 86. Example of a topodem contact constraint between the end-effector and a spherical surface.]

#### 6.6.5.5. Constraint Embedding

Constraint embedding

- Four bar linkage/wishbone constraint
- Gearing constraint
- Differential constraint

##### 6.6.5.5.1. Four Bar Linkages

TBD

##### 6.6.5.5.2. Geared Pin Joints

TBD

##### 6.6.5.5.3. Joint-Level Constraints

Algebraic constraint between arbitrary gen coord values. Example usage:

- For local constraints (gearing, differential), use constraint embedding
- More generally need to handle this
- Treat prescribed motion as a special case of this constraint on a single hinge
- Examples:
  - Prescribed motion constraint (or joint locking)
  - ATRV locked wheels constraint
  - Differential constraint
  - Gearing constraint

##### 6.6.5.5.4. Vehicle Configuration Kinematics

Example usage of vehicle configuration kinematics:

- Place rover

- KDriver constraint
- Driving Athlete with level hex
- Driving Athlete to come to dock
- Driving 2 Athletes in docked configuration

## 6.6.6. Ndarts Primary Object Classes API Reference

### 6.6.6.1. DartsBody API Reference

#### 6.6.6.1.1. Introduction

DartsBody class is inherited from the DartsNode (which in turn is inherited from a DFrame). Bodies can be assembled into larger multibody or subgraph structures using hinges to connect bodies to one another.

#### 6.6.6.1.2. Inter-body forces

In order to get the forces between two bodies (which are joint by a joint of some type), do this:

```
from Math.SOA_Py import SOASpatialVector
f = SOASpatialVector()
body.parentHinge().getInterBodyForce(f)
```

PYTHON

This gives you the total force,  $f$ , across the joint between 'body' and its parent body, including constraint forces/torques in the non-articulated degrees of freedom as well as joint forces/torques in the articulated degrees of freedom. By default the result is given in the child body's frame.

Note that this function takes a second optional argument: the frame you would like the results in. For instance, if you want the results in the parent body frame, you could do this:

```
from Math.SOA_Py import SOASpatialVector
f = SOASpatialVector()
body.parentHinge().getInterBodyForce(f, pbf)
```

PYTHON

where 'pbf' is the parent body frame.

Although these examples are in Python, the C++ code is almost identical.

#### 6.6.6.1.3. DartsBody Class API Documentation



#### Note

For Doxygen documentation, please see: [DartsBody](#)

```
class Dshell.Ndarts_Py.DartsBody(DartsBody self, DartsMbody mb, std::string const & _name, bool rootBody=False) → DartsBody
class Dshell.Ndarts_Py.DartsBody(DartsBody self, DartsBody body, DartsMbody other_mb, DartsBody other_parent_body=None) → DartsBody
```

PYTHON

The Ndarts body class.

C++ includes: DartsBody.h

Creates a DartsBody with the specified name.

If rootBody is true, then the virtual root body is created.

#### 6.6.6.1.4. Related Regression Tests

##### Ndarts regtest: Jacobians

▶ [Click to see the NdartsTest/test/test\\_Python/test\\_basic/test\\_jacobians/script.py script](#)

##### Ndarts regtest: Jacobian Derivatives

▶ [Click to see the NdartsTest/test/test\\_Python/test\\_basic/test\\_jacobdot/script.py script](#)

##### Ndarts regtest: Change Model Params

▶ [Click to see the NdartsTest/test/test\\_Python/test\\_basic/test\\_changeModel/script.py script](#)

##### Ndarts regtest: Procedural serial chain

▶ [Click to see the NdartsTest/test/test\\_Python/test\\_basic/test\\_automated/test\\_serialChain/script.py script](#)

##### Ndarts regtest: Procedural tree system

▶ [Click to see the NdartsTest/test/test\\_Python/test\\_basic/test\\_automated/test\\_treeSystem/script.py script](#)

##### Ndarts regtest: Integration

▶ [Click to see the NdartsTest/test/test\\_Python/test\\_basic/test\\_integrate/script.py script](#)

##### Ndarts regtest: Inverse Dynamics

▶ [Click to see the NdartsTest/test/test\\_Python/test\\_basic/test\\_inv\\_dyn/script.py script](#)

Ndarts regtest: Data Cache Forward Dynamics

▶ [Click to see the NdartsTest/test/test\\_Python/test\\_basic/test\\_dataCaching/script.py script](#)

Ndarts regtest: Data Cache Inverse Dynamics

▶ [Click to see the NdartsTest/test/test\\_Python/test\\_basic/test\\_dataCaching/scriptInverseDyn.py script](#)

Ndarts regtest: Operational Space Inertias

▶ [Click to see the NdartsTest/test/test\\_Python/test\\_basic/test\\_osi/script.py script](#)

Ndarts regtest: Algorithmic multibody

▶ [Click to see the NdartsTest/test/test\\_Python/test\\_basic/test\\_algmbody/script.py script](#)

Ndarts regtest: Tumbling Rigid Body

▶ [Click to see the NdartsTest/test/test\\_Python/test\\_basic/test\\_rigidbody/test\\_tumbling/script.py script](#)

Ndarts regtest: Tumbling Rigid Body (Semi Implicit Euler integrator)

▶ [Click to see the NdartsTest/test/test\\_Python/test\\_basic/test\\_rigidbody/test\\_tumbling/scriptEulerSemiImplicit.py script](#)

### 6.6.6.2. DartsSpatialInertia API Reference

#### 6.6.6.2.1. Introduction

DartsSpatialInertia class.

#### 6.6.6.2.2. DartsSpatialInertia Class API Documentation



*Note*

For Doxygen documentation, please see: [DartsSpatialInertia](#)

#### 6.6.6.2.3. Related Regression Tests

- Ndarts regtest: Puma General
- Ndarts regtest: Puma General Dump

### 6.6.6.3. DartsConstraint API Reference

#### 6.6.6.3.1. Introduction

DartsConstraint class.

#### 6.6.6.3.2. DartsConstraint Class API Documentation



*Note*

For Doxygen documentation, please see: [DartsBaseConstraint](#)



*Note*

For Doxygen documentation for specific constraints, please see: | [DartsFramePairConstraint](#) | [DartsClosureConstraint](#)

#### 6.6.6.3.3. Related Regression Tests

Ndarts regtest: Ball joint Constraint

▶ [Click to see the NdartsTest/test/test\\_Python/test\\_constraints/test\\_BallConstraint/script.py script](#)

Ndarts regtest: Ball joint Constraint

▶ [Click to see the NdartsTest/test/test\\_Python/test\\_constraints/test\\_BallConstraint/script1.py script](#)

Ndarts regtest: Custom Constraint

▶ [Click to see the NdartsTest/test/test\\_Python/test\\_constraints/test\\_CustomConstraint/script.py script](#)

Ndarts regtest: UJoint Constraint (Custom)

▶ [Click to see the NdartsTest/test\\_Python/test\\_constraints/test\\_CustomConstraint/script\\_u\\_joint.py script](#)

Ndarts regtest: Locked joint Constraint

▶ [Click to see the NdartsTest/test/test\\_Python/test\\_constraints/test\\_LockedConstraint/script.py script](#)

Ndarts regtest: Pin joint Constraint

▶ [Click to see the NdartsTest/test/test\\_Python/test\\_constraints/test\\_PinConstraint/script.py script](#)

## Ndarts regtest: Pin joint Constraint

▶ [Click to see the NdartsTest/test/test\\_Python/test\\_constraints/test\\_PinConstraint/script7.py script](#)

## Ndarts regtest: UJoint Constraint

▶ [Click to see the NdartsTest/test/test\\_Python/test\\_constraints/test\\_UjointConstraint/script.py script](#)

## Ndarts regtest: dualArm system

▶ [Click to see the NdartsTest/test/test\\_Python/test\\_constraints/test\\_dualArm/script.py script](#)

## Ndarts regtest: dualArm system

▶ [Click to see the NdartsTest/test/test\\_Python/test\\_constraints/test\\_dualArm/script\\_old.py script](#)

### 6.6.6.4. DartsMbody API Reference

#### 6.6.6.4.1. Introduction

DartsMbody class is a collection of bodies connected using hinges that form a system. Ideally each simulation should have one multibody that contains all bodies in a scene. Once made, the multibody can be split into smaller subgraphs (DartsSubGraph's) for individual analysis.

#### 6.6.6.4.2. DartsMbody Class API Documentation



#### Note

For Doxygen documentation, please see: [DartsMbody](#)

```
class Dshell.Ndarts_Py.DartsMbody(DartsMbody self, std::string const & name, Frame frame) → DartsMbody
----
```

PYTHON

This is the multibody object in Ndarts and derived from the Darts base class.

C++ includes: DartsMbody.h

default constructor

#### 6.6.6.4.3. Related Regression Tests

- [Ndarts regtest: Jacobians](#)
- [Ndarts regtest: Jacobian Derivatives](#)
- [Ndarts regtest: Change Model Params](#)
- [Ndarts regtest: Procedural serial chain](#)
- [Ndarts regtest: Procedural tree system](#)
- [Ndarts regtest: Integration](#)
- [Ndarts regtest: Inverse Dynamics](#)
- [Ndarts regtest: Data Cache Forward Dynamics](#)
- [Ndarts regtest: Data Cache Inverse Dynamics](#)
- [Ndarts regtest: Operational Space Inertias](#)
- [Ndarts regtest: Algorithmic multibody](#)

### 6.6.6.5. DartsSubGraph API Reference

#### 6.6.6.5.1. Introduction

DartsSubGraph class is a collection of connected bodies. Any subgraph can be used for the same type of calculations that a full DartsMbody object supports but restricted to the set of connected bodies that form the subgraph.

A subgraph is connected to a single parent body (or virtual body in case of the root body).

All the bodies in a subgraph must be connected!



The DartsMbody class is derived from DartsSubGraph.

#### 6.6.6.5.2. Normal use

If you wish to use a subgraph for multibody calculations, you need to do the following steps:

1. Save the previously active subgraph (usually the main mbody object)
2. Tell mbody to use the subgraph for its computations

3. Perform the desired calculations
4. Restore the original previously active subgraph

Here is an example in C++:

```

// Get a pointer to the main mbody object
Ndarts::DartsMbody *mbody_sg = static_cast<Ndarts::DartsMbody *>(simulation().mbody());

// 1. Save the previously active subgraph (usually the main mbody object)
Ndarts::DartsSubGraph &previous_sweeps_sg = mbody_sg->sweepsSubGraph();

// 2. Tell mbody to use the subgraph for its computations
mbody_sg->sweepsSubGraph(*_computational_sg);

// 3. Perform the desired calculations
const Ndarts::DartsSpatialInertia isp = *_computational_sg->inertialFrameCompositeSpatialInertia();
SOAVector3 pbPositionOfCmWrtPb = isp.B2CM();
mass = isp.mass();

// 4. Restore the original previously active subgraph
mbody_sg->sweepsSubGraph(previous_sweeps_sg);

```

PYTHON



**Note**

Due to recent improvements in Ndarts, temporarily swapping in the subgraph is not necessary to access the position of the subgraph CM. Every subgraph has a CM frame that is updated automatically—even when the mass of one of the subgraph bodies changes.

However, accessing any of the subgraph inertia properties (including mass) will need the temporary subgraph swap outlined above.

6.6.6.5.3. Issues

One issue that can complicate using a subgraph is if any of the bodies in the subgraph become disconnected. Since all the bodies in subgraph must be connected, if one body in the subgraph is detached, it is no longer connected to the other bodies in the subgraph—and the subgraph becomes invalid. Any attempt to lock the mbody object (via `lockObject()`) will fail. Since it is not possible to delete a body from a subgraph, the subgraph must be deleted and re-created without the offending body so that it becomes a valid subgraph—then `lockObject()` should work.

See the `Ndarts-subgraph-topology-regtest-detach` regtest for an example of dealing with a subgraph with a detached body (in python).

6.6.6.5.4. DartsSubGraph Class API Documentation



**Note**

For Doxygen documentation, please see: [DartsSubGraph](#)

```

class Dshell.Ndarts_Py.DartsSubGraph(DartsSubGraph self, std::string const & name, DartsSubGraph graph, DartsBodyBaseList1 bodyList, bool aggregationMode) -> DartsSubGraph
----

```

PYTHON

The multibody subgraph class.

C++ includes: `DartsSubGraph.h`

Constructor for a subgraph of a graph defined from its list of bodies.

The smallest spanning tree for the input list of bodies is used to define the subgraph's bodies. If `aggregationMode` is true, then the spanning tree's root is used as the virtual root for the subgraph, and is not included in the subgraph's internal bodies. If `aggregationMode` is false, the full spanning tree makes up the bodies in the subgraph, and its ancestor is used for the virtual root.

When the input list of bodies does not have a common ancestor body, then the multibody virtual root is the subgraph's virtual root as well, and the spanning forest bodies make up the subgraph's bodies.

6.6.6.5.5. Related Regression Tests

**Ndarts regtest: Ndarts subgraphs**

- ▶ [Click to see the NdartsTest/test/test\\_Python/test\\_basic/test\\_subgraphs/test\\_subgraph\\_topology/script.py script](#)

**Ndarts regtest: Ndarts dynamics simulation on a subgraph**

- ▶ [Click to see the NdartsTest/test/test\\_Python/test\\_basic/test\\_subgraphs/test\\_subgraph\\_topology/scriptDyn.py script](#)

**Ndarts regtest: Ndarts subgraph properties**

- ▶ [Click to see the NdartsTest/test/test\\_Python/test\\_basic/test\\_subgraphs/test\\_subgraph\\_properties/scriptDirect.py script](#)

**Ndarts regtest: Ndarts subgraph summation of forces and torques**

- ▶ [Click to see the NdartsTest/test/test\\_Python/test\\_basic/test\\_subgraphs/test\\_subgraph\\_forces/script.py script](#)

6.6.6.6. DartsHinge API Reference

6.6.6.6.1. DartsHinge Class API Documentation



*Note*

For Doxygen documentation about hinges, please see: [DartsHinge](#)



*Note*

| For Doxygen documentation of specific hinge types, please see: | [DartsLockedHinge](#) | [DartsPinHinge](#) | [DartsUjointHinge](#) | [DartsGimbalHinge](#) | [DartsBallHinge](#) | [DartsSliderHinge](#) | [DartsPlanarHinge](#) | [DartsTranslationalHinge](#) | [DartsCompositeTranslationalHinge](#) | [DartsFull6DofHinge](#) | [DartsInertialFull6DofHinge](#) | [DartsFull6DofEulerHinge](#) | [DartsFull6DofGimbalHinge](#) | [DartsCustomHinge](#)

```
class Dshell.Ndarts_Py.DartsHinge(DartsHinge self, DartsMbody mb, Frame oframe, Frame pframe, std::string const & nm="") → DartsHinge
-----
```

PYTHON

The hinge class. This is a container class of the sub-hinges.

C++ includes: [DartsHinge.h](#)

Constructor - registers nodes as well.

6.6.6.2. Related Regression Tests

**Ndarts regtest: Custom Hinges comparison with Puma**

- ▶ [Click to see the NdartsTest/test/test\\_Python/test\\_basic/test\\_customhinge/test\\_multihinge\\_compare/script.py script](#)

**Ndarts regtest: Custom Hinges comparison with Puma (general hinges)**

- ▶ [Click to see the NdartsTest/test/test\\_Python/test\\_basic/test\\_customhinge/test\\_multihinge\\_compare/script\\_genhinges.py script](#)

**Ndarts regtest: Custom Ujoint Hinge**

- ▶ [Click to see the NdartsTest/test/test\\_Python/test\\_basic/test\\_customhinge/test\\_ujoint\\_compare/script.py script](#)

**Ndarts regtest: Custom Ujoint Hinge**

- ▶ [Click to see the NdartsTest/test/test\\_Python/test\\_basic/test\\_customhinge/test\\_ujoint\\_compare/script\\_u\\_joint.py script](#)

**Ndarts regtest: Generic Hinges**

- ▶ [Click to see the NdartsTest/test/test\\_Python/test\\_basic/test\\_genhinges/script.py script](#)

**Ndarts regtest: Puma**

- ▶ [Click to see the NdartsTest/test/test\\_Python/test\\_basic/test\\_puma/script.py script](#)

**Ndarts regtest: Puma Ball Joint**

- ▶ [Click to see the NdartsTest/test/test\\_Python/test\\_basic/test\\_puma\\_balljt/script.py script](#)

**Ndarts regtest: Puma Ball Joint Dump**

- ▶ [Click to see the NdartsTest/test/test\\_Python/test\\_basic/test\\_puma\\_balljt/script1.py script](#)

**Ndarts regtest: Puma Free Flying 6DoF**

- ▶ [Click to see the NdartsTest/test/test\\_Python/test\\_basic/test\\_puma\\_fflying/script.py script](#)

**Ndarts regtest: Puma Free Flying 6DoF Dump**

- ▶ [Click to see the NdartsTest/test/test\\_Python/test\\_basic/test\\_puma\\_fflying/script1.py script](#)

**Ndarts regtest: Puma Free Flying 6DoF Inertial**

- ▶ [Click to see the NdartsTest/test/test\\_Python/test\\_basic/test\\_puma\\_fflying/scriptInertial.py script](#)

**Ndarts regtest: Puma General**

- ▶ [Click to see the NdartsTest/test/test\\_Python/test\\_basic/test\\_puma\\_general/script.py script](#)

**Ndarts regtest: Puma General Dump**

- ▶ [Click to see the NdartsTest/test/test\\_Python/test\\_basic/test\\_puma\\_general/script1.py script](#)

**Ndarts regtest: Puma with Internal 6dof hinge**

- ▶ [Click to see the NdartsTest/test/test\\_Python/test\\_basic/test\\_puma\\_internal6dof/script.py script](#)

**Ndarts regtest: Puma with Internal 6dof hinge (inertial)**



▶ [Click to see the NdartsTest/test/test\\_Python/test\\_basic/test\\_puma\\_internal6dof/scriptInertial.py script](#)

Ndarts regtest: Puma with Locked hinge

▶ [Click to see the NdartsTest/test/test\\_Python/test\\_basic/test\\_puma\\_locked/script.py script](#)

Ndarts regtest: Puma with Prescribed hinge

▶ [Click to see the NdartsTest/test/test\\_Python/test\\_basic/test\\_puma\\_prescribed/script.py script](#)

### 6.6.6.7. DartsSubhinge API Reference

#### 6.6.6.7.1. Introduction

DartsSubhinge class is derived from the EdgeFrame2Frame. Subhinges are used as building blocks to make the complete hinges.

#### 6.6.6.7.2. DartsSubhinge Class API Documentation



##### Note

For Doxygen documentation, please see: [DartsSubhingePhysical](#) and [DartsSubhingeFixed](#)



##### Note

For Doxygen documentation for specific subhinges, please see: | [DartsLockedSubhinge](#) | [DartsPinSubhinge](#) | [DartsLinearSubhinge](#) | [DartsSphericalSubhinge](#) | [DartsEulerSphericalSubhinge](#) | [DartsLinear3Subhinge](#)

#### 6.6.6.7.3. Related Regression Tests

- Ndarts regtest: Custom Hinges comparison with Puma
- Custom Hinges comparison with Puma (general hinges)
- Ndarts regtest: Custom Ujoint Hinge
- Ndarts regtest: Custom Ujoint Hing
- Ndarts regtest: Generic Hinges
- Ndarts regtest: Puma
- Ndarts regtest: Puma Ball Joint
- Ndarts regtest: Puma Ball Joint Dump
- Ndarts regtest: Puma Free Flying 6DoF
- Ndarts regtest: Puma Free Flying 6DoF Dump
- Ndarts regtest: Puma Free Flying 6DoF Inertial
- Ndarts regtest: Puma General
- Ndarts regtest: Puma General Dump
- Ndarts regtest: Puma with Internal 6dof hinge
- Ndarts regtest: Puma with Internal 6dof hinge (inertial)
- Ndarts regtest: Puma with Locked hinge
- Ndarts regtest: Puma with Prescribed hinge

/regtests/test\_basic/test\_puma\_prescribed/script.rst

### 6.6.6.8. DartsNode API Reference

#### 6.6.6.8.1. Introduction

DartsNode class is inherited from the DFrame. Nodes are simply points that are attached to the body. There are many different types of nodes, each with its own particular purpose and function. For example, the pnode and onode are connection points between hinges and bodies.

#### 6.6.6.8.2. DartsNode Class API Documentation



##### Note

For Doxygen documentation, please see: [DartsNode](#)



##### Note

For Doxygen documentation for specific nodes, please see: | [DartsSensorNode](#) | [DartsActuatorNode](#) | [DartsConstraintNode](#) | [DartsHingeNode](#) | [DartsHingePnode](#) | [DartsHingeOnode](#)

```
class Dshell.Ndarts_Py.DartsNode(DartsNode self, DartsBody body, std::string const & name, Ndarts::DartsNode::NodeType node_type) → DartsNode
-----
```

PYTHON

This is the Ndarts Node class, derived from the DartsBaseNode which models the Darts++ node object.

C++ includes: DartsNode.h

Constructor for a body node.

#### 6.6.6.8.3. Related Regression Tests

- Ndarts regtest: Jacobians
- Ndarts regtest: Jacobian Derivatives
- Ndarts regtest: Change Model Params
- Ndarts regtest: Procedural serial chain
- Ndarts regtest: Procedural tree system
- Ndarts regtest: Integration
- Ndarts regtest: Inverse Dynamics
- Ndarts regtest: Data Cache Forward Dynamics
- Ndarts regtest: Data Cache Inverse Dynamics
- Ndarts regtest: Operational Space Inertias
- Ndarts regtest: Algorithmic multibody

#### 6.6.6.9. DartsBStar API Reference

##### 6.6.6.9.1. Introduction

DartsBStar class.

##### 6.6.6.9.2. DartsBStar Class API Documentation



*Note*

For Doxygen documentation, please see: [DartsBStar](#)

##### 6.6.6.9.3. Related Regression Tests

#### Ndarts regtest: BStar

- ▶ [Click to see the NdartsTest/test/test\\_Python/test\\_basic/test\\_BStar/script.py script](#)

#### Ndarts regtest: BStar Body2Joint

- ▶ [Click to see the NdartsTest/test/test\\_Python/test\\_basic/test\\_BStar/scriptBodyToJoint.py script](#)

#### Ndarts regtest: BStar Tree

- ▶ [Click to see the NdartsTest/test/test\\_Python/test\\_basic/test\\_BStar/scriptTree.py script](#)

#### 6.6.6.10. DartsInverseKinematicsSolver API Reference

##### 6.6.6.10.1. Introduction

DartsInverseKinematicsSolver class.

##### 6.6.6.10.2. DartsInverseKinematicsSolver Class API Documentation



*Note*

For Doxygen documentation, please see: [DartsInverseKinematicsSolver](#)

##### 6.6.6.10.3. Related Regression Tests

- Ndarts regtest: Custom Constraint
- Ndarts regtest: UJoint Constraint (Custom)
- Ndarts regtest: UJoint Constraint

#### 6.6.6.11. DartsTreeDynamicsSolver API Reference

##### 6.6.6.11.1. Introduction

DartsTreeDynamicsSolver class.

##### 6.6.6.11.2. DartsTreeDynamicsSolver Class API Documentation



*Note*

For Doxygen documentation, please see: [DartsTreeDynamicsSolver](#)

##### 6.6.6.11.3. Related Regression Tests

- Ndarts regtest: Puma
- Ndarts regtest: Puma General

## Ndarts regtest: Granular n-Particles

▶ [Click to see the NdartsTest/test/test\\_Python/test\\_basic/test\\_granular/test\\_ngranular/script.py script](#)

## Ndarts regtest: Granular n-Particles Sim

▶ [Click to see the NdartsTest/test/test\\_Python/test\\_basic/test\\_granular/test\\_ngranular\\_sim/script.py script](#)

## Ndarts regtest: Tumbling Rigid Body

## Ndarts regtest: Tumbling Rigid Body (Semi Implicit Euler integrator)

### 6.6.6.12. DartsSubhingeData API Reference

#### 6.6.6.12.1. Introduction

DartsSubhingeData class.

#### 6.6.6.12.2. DartsSubhingeData Class API Documentation



#### Note

For Doxygen documentation, please see: [DartsSubhingeData](#)

#### 6.6.6.12.3. Related Regression Tests

TBD

### 6.6.6.13. DShape API Reference

#### 6.6.6.13.1. Introduction

DShape class is for shape models used by Ndarts for analytical shapes used for constraints (eg, a ball rolling on a surface).

#### 6.6.6.13.2. DShape Class API Documentation



#### Note

For Doxygen documentation, please see: [DShape](#)

#### Note

For Doxygen documentation for specific shapes, please see:

[DShapePoint](#)

[DShape1D](#)

[DShape1DStraightLine](#)

[DShape1DCircle](#)

[DShape1DEllipse](#)



[DShape2D](#)

[DShape2DPlane](#)

[DShape2DSphere](#)

[DShape2DCylinder](#)

[DShape2DTorus](#)

[DShape2DTorus2](#)

[DShape2DEllipsoid](#)

```
class Dshell.Ndarts_Py.DShape(DShape self, FrameContainer fc, std::string const & name, size_t dims) → DShape
----
```

PYTHON

The base class for shapes.

C++ includes: DShape.h

Constructor:

#### 6.6.6.13.3. Related Regression Tests

**Ndarts regtest: 2D Cylinder shape**

- ▶ [Click to see the NdartsTest/test/test\\_Python/test\\_shapes/test\\_cylinder2D/script.py script](#)

Ndarts regtest: 2D Ellipsoid shape

- ▶ [Click to see the NdartsTest/test/test\\_Python/test\\_shapes/test\\_ellipsoid2D/script.py script](#)

Ndarts regtest: 2D Plane shape

- ▶ [Click to see the NdartsTest/test/test\\_Python/test\\_shapes/test\\_plane2D/script.py script](#)

Ndarts regtest: Rolling contact kinematics between a plane and a sphere

- ▶ [Click to see the NdartsTest/test/test\\_Python/test\\_shapes/test\\_roll/script.py script](#)

Ndarts regtest: 2D Sphere shape

- ▶ [Click to see the NdartsTest/test/test\\_Python/test\\_shapes/test\\_sphere2D/script.py script](#)

Ndarts regtest: 2D Torus shape

- ▶ [Click to see the NdartsTest/test/test\\_Python/test\\_shapes/test\\_torus2D/script.py script](#)

## 7. DFrame

### 7.1. Background

#### 7.1.1. Reference & Source material

- [DFrame Doxygen docs](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/DFrame/html/) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/DFrame/html/)
- [Team talks](https://dlabportal.jpl.nasa.gov/resources/darts-lab-talks/) (https://dlabportal.jpl.nasa.gov/resources/darts-lab-talks/)
- [2019, 2020 course slides](https://dlabportal.jpl.nasa.gov/documentation/coursetutorial-links/) (https://dlabportal.jpl.nasa.gov/documentation/coursetutorial-links/)
- [Sphinx docs](https://dartslab.jpl.nasa.gov/dlabdocs/) (https://dartslab.jpl.nasa.gov/dlabdocs/)
- [Jupyter notebooks](https://dlabnotebooks.jpl.nasa.gov/hub/login) (https://dlabnotebooks.jpl.nasa.gov/hub/login)
- [DARTS Lab QA site](https://dartslab.jpl.nasa.gov/qa/) (https://dartslab.jpl.nasa.gov/qa/)

/home/dlab/repo/www/DLabDocs

### 7.2. Design

#### 7.2.1. DartsFacadeScene

##### 7.2.1.1. Redraw and close registries

DartsFacadeScene comes equipped with an registry and a close registry. Each of these registries is implemented as a `DScene::CallbackRegistry`, which has a dictionary-like storage scheme where the keys are strings and the values are callback functions, which take no arguments and return nothing. Whenever the `updateSceneFrameTransform` method is called, every callback from the update registry is also called. Whenever the FacadeScene is closed, every callback in the close registry is called; note, callbacks in the close registry are called in the opposite order they were registered in.

The close registry is designed to remove objects that are created by or needed for the update callbacks. Hence, close callbacks can only be registered when registering an update callback, and likewise, can only be unregistered when unregistering an update callback.

### 7.3. Usage

#### 7.3.1. DartsFacadeScene

##### 7.3.1.1. Redraw and close registries

DartsFacadeScene has the following functions that can be used to modify the update and close registries:

- `registerUpdateCallack` - used to register a callback function that will run each time the `updateSceneFrameTransforms` is called. Optionally, a close callback can be registered as well.
- `unregisterUpdateCallack` - used to remove a callback function from the update registry and from the close registry if applicable.
- `updateCallbackExists` - returns True if the name matches the name of an update callback. Returns False otherwise.
- `getUpdateCallack` - returns the update callback associated with the given name.

For more information on how these registries work, see the DartsFacadeScene design.

### 7.4. Software

### 7.5. Raw documentation



**TBD:** Need scrubbing before integration.

#### 7.5.1. DFrame: Add method to look up frame from a partial frame names path



**TBD:** Needs scrubbing. Notes brought over from [issue](https://dartsgitlab-internal.jpl.nasa.gov/darts-lab/jpl-internal/all-access/development/numerics/dframe/-/issues/3) (https://dartsgitlab-internal.jpl.nasa.gov/darts-lab/jpl-internal/all-access/development/numerics/dframe/-/issues/3).

There are times when we want to look up a frame by name, but the frame name may not be unique. One way to do this would be to narrow down the look up context to make the search unique. One way to do this would be to use the names of ancestor frames to define a partial path this is sufficient to find the desire frame. The path does not need to contain all the frames in the path, but just enough to make the result unique. For instance to find the `imu` frame on `rvr3`, we could look up via the `rvr3.imu` path.

This can be done by adding 2 methods:

- `Frame::frameFromPartialPath(const std::string& path)`: this would return the downstream frame identified by the partial path. It's implementation would be to recursively look up the frame from the head of the partial path, and call this method again on the remainder path until the unique frame is found. The method such throw an exception if a unique frame is not found.
- `FrameContainer::frameFromPartialPath(const std::string& path)`: this would return call the root frame's method to find the frame.

##### 7.5.1.1. Solution

Added `Frame.frameFromPath()` method to do this. Skipping the `FrameContainer` method since it can call the `Frame` level method on the `rootFrame()`.

#### 7.5.2. Dshell++: Support calling `updateSceneFrameTransform()` at the granularity required by individual models



TBD: Needs scrubbing. Notes brought over from [issue](#)

([https://dartsgitlab-internal.jpl.nasa.gov/darts-lab/jpl-internal/all-access/development/simulation-framework/dshellpp/-/issues/15#note\\_9058](https://dartsgitlab-internal.jpl.nasa.gov/darts-lab/jpl-internal/all-access/development/simulation-framework/dshellpp/-/issues/15#note_9058)).

I believe this item is done and ready to close. The final solution also required issues raised in issues <https://dartsgitlab-internal.jpl.nasa.gov/darts-lab/jpl-internal/all-access/development/scene-geometry/dscene/-/issues/21> and <https://dartsgitlab-internal.jpl.nasa.gov/darts-lab/jpl-internal/licensed/licensed-modules/cadre/-/issues/184>. The refactoring summary is as follows:

- models that depend on scenes now need to call `updateSFT` at the appropriate time with a time stamp
- added the public `Simulation::updateSceneFrameTransforms()` method that models can call for this. They just need to call the `updateSceneFrameTransforms()` directly whenever they are making a scene related call. This can be done by adding code such as

```
DFrame::DartsFacadeSceneFrame* sf = simulation().frameContainer().rootFrame().sceneFrame();
sf->scene().updateSceneFrameTransforms(t);
```

- added a `Scene::resetTime(time)` for such special situations where we need to reset the clock? For example, the EELS placement algorithm can run for up to 10 seconds, and then the time gets reset to 0.0.
- overhauled how scene managers send updated transform info to client scenes to better support client scenes that do not intrinsically support scene graph hierarchies for frames and objects, and instead assume a flat list
- now client scenes can set a flag to indicate that they do not support scene graphs, so that their scene frames do not have children scene frames, and the scene manager sends absolute pose info for the scene frames.
- updated `OptixScene`, `DBullet` and `DMeshScene` to set the non-scene-graph flag
- with this done, streamlined and simplified `OptixScene` interface so that updates happen properly
- the `Cadre/test/test-ScoutNG2/test_cfg_gnc_zeroimnoise` test case verifies that this is working

## 7.6. Sphinx documentation

### 7.6.1. Darts/Dshell Frames

#### 7.6.1.1. Frame System Basics

Frames are a convenient way to locate, view and transform Cartesian vector quantities across different entities (possibly in motion) within a simulation. \* entities have a physical location and orientation \* entities implicitly or explicitly have frames associated with them

Examples of such entities include:

- Celestial bodies (eg. Sun, Earth, Moon)
- Spacecraft (eg. ascent vehicles, orbiters, landers)
- Spacecraft elements (eg. antennas, sensors)
- Topography elements (eg. launch sites, landing sites, DEM patches)
- User specified frames of interest (eg. goals, markers for line-of-sight analysis)
- 3D graphics elements (eg. light location, viewing camera location for chase views)

##### 7.6.1.1.1. Design Goals

Key design goals for the Frame system are to:

- Represent Cartesian vector quantities. Other types of coordinate systems (eg. polar/spherical coordinates) must be handled in other ways.
- Decouple the frame information from the individual physical object classes, so that all physical objects work with the frames in the same way and with the same API.
- Manage the frames family as its own global backbone layer that physical objects are bound to. Thus all transform relations between any pair of objects reduces to queries on the frames they are attached to.
- Build in methods into the frame classes to make it easy to get/set transform relationships among frames so that the simulation objects can make use of this lower level capabilities.
- Make it easy to trigger callbacks when the relative transforms between frame pairs change.

##### 7.6.1.1.2. Key Requirements

Some of the key requirements on the Frame classes are:

- Be able to get relative transform information of any frame with respect to another frame.
- Support different modalities for propagating the parent/child frame relative data.
- Be able to use frames uniformly across all simulation elements including multibody components, terrain data, visualization elements etc.
- Only recompute data when necessary for better performance.
- Be able to change parent/child frame relationships.

#### 7.6.1.2. Frame Object and Trees

Frame objects are attached to physical objects or locations in the world. A **tree** of Frame object instances defines the (potentially time varying) relationship between the frames.

The Figure illustrates a tree of frames that is typical of a DSEDS simulation setup, with the frames being represented by the circular nodes and the edges denote the parent/child `Frame2Frame` transformations.

### 7.6.1.2.1. Example Frame Tree

Frame Tree with Darts and Non-Darts Propagated Frames

image::images/Frames\_Spice\_and\_Darts.\*[Frame Tree with Darts and Non-Darts Propagated Frames]

See the table in the next section for the acronyms in this figure

#### 7.6.1.2.1.1. Example Frame Tree Objects

| Frame Type            | Notes                   |   |
|-----------------------|-------------------------|---|
| SSBC                  | Spice Frame             | Solar System Barycenter   |
| Target Body           | Spice Frame             | Target centered, Mean Equator and Prime Meridian                          |
| Target Body Unrotated | Spice Frame             | Aligned with solar system barycenter                                      |
| INERTIAL              | Frame                   | Root of the Darts dynamics,   |
| PCI                   | Frame of Body Object    | Target centered, Mean Equator and Prime Meridian of epoch                 |
| PCR                   | Frame of Body           | Rotated by Darts a using prescribed kinematics                            |
| Target Shape Model    | Frame of Terrain Object | A SimScape TopoPlanet that encodes the topography of the target           |
| DEM                   | Frame of Terrain Object | A high-resolution Digital Elevation Map                                   |
| Target Location       | Frame of Node Object    | Target location specified in TargetLocation param instance                |
| CapsuleBase0          | Frame of Body Object    | The canonical top-most body of the spacecraft with zero mass and inertial |
| CapsuleBase           | Frame of a Body Object  | Child of the CapsuleBase with non-zero mass and inertia                   |
| Ideal IMU node        | Frame of Node Object    | A node on the CapsuleBase used to affix an IMU                            |

#### 7.6.1.2.1.2. Example Frame Tree Object Access in Dsends

Each frame may be accessed in the simulation by the following calls at the DSENDS Python prompt:

| Frame                 | Python access  |
|-----------------------|--|
| SSBC                  | DsendsSim._ssbcFrame   |
| Target Body           | DsendsSim._targetpcrFrame  |
| Target Body Unrotated | DsendsSim._targetJ2000pciFrame   |
| INERTIAL              | DsendsSim._targetpciFrame  |
| PCI                   | DsendsSim.dsendsassembly_asmTARGET.pci.refNode().frame()                           |
| PCR                   | DsendsSim.dsendsassembly_asmTARGET.pcr.refNode().frame()                           |
| Target Shape Model    | DsendsSim.simparams['Target']._topoPlanetGlobalDrm.coordInfo().frame()             |
| DEM                   | DsendsSim.simparams['Target']._localDem.coordInfo().frame()                        |
| Target Location       | DsendsSim.dsendsassembly_asmTARGET_asmSiteSensor.nodeFrame                         |
| CapsuleBase0          | DsendsSim.dsendsassembly_asmSC.topbody.refNode().frame()                           |
| CapsuleBase           | DsendsSim.mbody().body('CapsuleBase',0,True).refNode().frame()                     |
| Ideal IMU             | DsendsSim.mbody().body('CapsuleBase',0,True).sensorNode('IdealIMU').frame().name() |

Only a subset of the frame relationships are managed by the Darts dynamics engine with other relationships managed by other software. In DSENDS:

- Frames associated with the target body and spacecraft are managed by Darts
- Frames associated with other celestial bodies (if present) are managed by the NAIF Spice facility (Note that the target body frame could have been managed by Spice but the DSENDS implementation uses the dynamics engine so as to provide a more sophisticated control of its time evolution)

#### 7.6.1.2.2. Frame Class

- [Frame Class Doxygen documentation](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/DFrame/html/classDFrame_1_1Frame.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/DFrame/html/classDFrame\_1\_1Frame.html)
- [Frames and Frame2Frames Doxygen page](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/DFrame/html/DFrameFrames.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/DFrame/html/DFrameFrames.html)

#### 7.6.1.3. The Frame-To-Frame Relationship

- A Frame-To-Frame relationship encapsulates the following relative

## information between frames

- spatial position
  - attitude
  - linear velocity
  - angular velocity
  - linear acceleration
  - angular acceleration
- In a Frame-to-Frame relationship the convention is to designate:
    - the first frame has an "o" prefix
    - the second frame has "p" prefix.

This is an alphabetical convention that the joint node on the inboard body is labelled "o" and the joint node on the outboard body is labeled "p" (since "O" is before "P" alphabetically).

For examples of Frame2Frame uses, see the regression tests in DFrame, DshellCommon, and EduSims.

### 7.6.1.3.1. Frame2Frame Class

- [Frame2Frame Class Doxygen documentation](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/DFrame/html/classDFrame_1_1Frame2Frame.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/DFrame/html/classDFrame\_1\_1Frame2Frame.html)
- [Frames and Frame2Frames Doxygen page](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/DFrame/html/DFrameFrames.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/DFrame/html/DFrameFrames.html)

### 7.6.1.3.2. Specialization

- When the two Frames in tree have a *direct parent-child* relationship then a *edge* relationship exists (In the Figure the CapsuleBase0 and CapsuleBase frames have this relationship).
  - Further specialization is possible depending on whether the relationship is:
    - fixed
    - prescribed using Darts kinematics
    - defined by non-Darts entities such as Spice.
  - [EdgeFrame2Frame Class Doxygen documentation](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/DFrame/html/classDFrame_1_1EdgeFrame2Frame.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/DFrame/html/classDFrame\_1\_1EdgeFrame2Frame.html)
  - [Frame2Frame Classes Doxygen page](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/DFrame/html/DFrame_Frame2Frame_Class_page.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/DFrame/html/DFrame\_Frame2Frame\_Class\_page.html)
- When the two frames are arbitrarily located in a Frame tree then a *chain* relationship exists (In the Figure the CapsuleBase and Target Location have this relationship).
  - The chain is defined by the sequence of edge relationships obtained from the minimal traversal of the Frame tree between the two frames.
  - [ChainedFrame2Frame Class Doxygen documentation](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/DFrame/html/classDFrame_1_1ChainedFrame2Frame.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/DFrame/html/classDFrame\_1\_1ChainedFrame2Frame.html)
  - [Frame2Frame Classes Doxygen page](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/DFrame/html/DFrame_Frame2Frame_Class_page.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/DFrame/html/DFrame\_Frame2Frame\_Class\_page.html)

## 7.6.1.4. Frame Implementation Issues

### 7.6.1.4.1. Computational Efficiencies

A number of techniques are used to minimize the re-computation of quantities when a frame or inter-frame quantity is requested. A detailed discussion of the caching, lazy computing, data freshness and related issues in the implementation may be found at the links:

- [Data caching notes](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/DFrame/html/DataCaching.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/DFrame/html/DataCaching.html)
- [Frame implementation notes](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/DFrame/html/DFrame_Frame_Implementation_Notes_page.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/DFrame/html/DFrame\_Frame\_Implementation\_Notes\_page.html)

### 7.6.1.5. Frame Tutorial

- xref:Sphinx\_DFrame\_oldframes\_regtest[Test for Frame2Frame objects]

#### 7.6.1.5.1. How to get vehicle-relative information

To get the relative position and attitude between any two bodies (or nodes) in the system, follow these steps:

- Get the frame for each of the two bodies or nodes, simply use the node or body. In Ndarts, both nodes and bodies are frames. The 'frame' of a body corresponds to the reference coordinate system for the body.

for example, assume we want to get the relative transform between body1 and node2 (which could be on some other body).

```
frame1 = body1.refNode().frame()
frame2 = node2.frame()
```

PYTHON

- Construct a Frame2Frame object between the two frames

```
f2f = frame1.frame2Frame(frame2)
```

PYTHON

this is the frame2Frame of body2 (frame2) with respect to body1 (frame1).

- Retrieve the relative transformation between the two frames:

```
reltrans = f2f.relTransform()
```

PYTHON



'reltrans' now contains a `SOAHomTran` object for the position and attitude of node2 with respect to body1 (in the coordinates of body1).



#### WARNING

While the `Frame2Frame` object 'f2f' is a live object that is updated when the multibody system changes, the 'reltrans' object that is obtained with the function call 'relTransform()' is not. 'reltrans' is a static value that is not updated when the multibody system changes. Therefore, it is up to the user to get a new value of the relative transform each time they need it to guarantee that it is current.

- To retrieve the position and attitude of node2 with respect to body1 :

```
rpos = reltrans.getTranslation()
quat = reltrans.getQuaternion()
```

PYTHON

- To retrieve the velocity of node2 with respect to body1 (in coordinates of the body1 reference frame):

```
rvel = f2f.oframeDerivRelSpVel()
```

PYTHON

This returns a `SOASpatialVector` vector object. Its first three values are the relative rotational angular velocity and its last three values are its relative velocity.

- Similarly, the relative acceleration can be obtained:

```
racc = f2f.oframeDerivRelSpAcc()
```

PYTHON

This also returns a `SOASpatialVector` vector object. Its first three values are the relative rotational angular acceleration and its last three values are its relative acceleration.

### 7.6.1.6. Frame-related API Overview

#### **`DFrame::Frame`** ([https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/DFrame/html/classDFrame\\_1\\_1Frame.html](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/DFrame/html/classDFrame_1_1Frame.html)) **API Highlights (selected functions)**

**`DFrame::Frame::isSpiceFrame()`** ([https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/DFrame/html/classDFrame\\_1\\_1Frame.html#a43be400bca005158d1c198bdb1e02e59](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/DFrame/html/classDFrame_1_1Frame.html#a43be400bca005158d1c198bdb1e02e59))

Returns True if the frame is a Spice frame

**`DFrame::Frame::getParentSpiceFrame()`** ([https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/DFrame/html/classDFrame\\_1\\_1Frame.html#ac0e1379d6230f59100869f20f23daa92](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/DFrame/html/classDFrame_1_1Frame.html#ac0e1379d6230f59100869f20f23daa92))

Return the first ancestor frame that is a `SpiceFrame` instance

**`DFrame::Frame::getParentFrame()`** ([https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/DFrame/html/classDFrame\\_1\\_1Frame.html#af766ff03d864a480452fce1730775338](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/DFrame/html/classDFrame_1_1Frame.html#af766ff03d864a480452fce1730775338))

Return the parent frame for this frame

**`DFrame::Frame::getEdgeFrame2Frame()`** ([https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/DFrame/html/classDFrame\\_1\\_1Frame.html#a24f8de3e70b536e5daed888751e3bdd6](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/DFrame/html/classDFrame_1_1Frame.html#a24f8de3e70b536e5daed888751e3bdd6))

Return the `Frame2Frame` object from the immediate parent to this frame

**`DFrame::Frame::childFrames()`** ([https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/DFrame/html/classDFrame\\_1\\_1Frame.html#a42e0cfd2cec26b799fea1b5fcd34316](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/DFrame/html/classDFrame_1_1Frame.html#a42e0cfd2cec26b799fea1b5fcd34316))

Returns a list of child frames

**`DFrame::Frame::dumpTree()`** ([https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/DFrame/html/classDFrame\\_1\\_1Frame.html#a9f6bb27a79405090a01f123d9d9da175](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/DFrame/html/classDFrame_1_1Frame.html#a9f6bb27a79405090a01f123d9d9da175))

Prints out frames tree starting with the specified base frame

#### **`DFrame::Frame2Frame`** **API Highlights (selected functions)** ([https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/DFrame/html/classDFrame\\_1\\_1Frame2Frame.html](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/DFrame/html/classDFrame_1_1Frame2Frame.html))

**`DFrame::Frame2Frame::isEdge()`** ([https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/DFrame/html/classDFrame\\_1\\_1Frame2Frame.html#aefc0efc2131d56a17ed67d1a4fcb9dd1](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/DFrame/html/classDFrame_1_1Frame2Frame.html#aefc0efc2131d56a17ed67d1a4fcb9dd1))

Returns True if there is a direct frame to frame connection between the frames involved.

**`DFrame::Frame2Frame::relTransform()`**

([https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/DFrame/html/classDFrame\\_1\\_1Frame2Frame.html#a3524b7d48f31bfb0f90d5bf94bc22c8e](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/DFrame/html/classDFrame_1_1Frame2Frame.html#a3524b7d48f31bfb0f90d5bf94bc22c8e))

Returns the homogeneous transform `SOAHomTran`. This allows you to determine the relative position and orientation between two frames.

### 7.6.2. Regression tests

#### Test for the `FrameContainer` class

- ▶ [Click to see the DFrame/test/test\\_FrameContainer/script.py script](#)

#### Test for `Frame2Frame` objects

- ▶ [Click to see the DFrame/test/test\\_old/script.py script](#)

## 8. DVar

### 8.1. Background

#### 8.1.1. Reference & Source material

- [DVar Doxygen docs](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/DVar/html/) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/DVar/html/)
- [Team talks](https://dlabportal.jpl.nasa.gov/resources/darts-lab-talks/) (https://dlabportal.jpl.nasa.gov/resources/darts-lab-talks/)
- [2019, 2020 course slides](https://dlabportal.jpl.nasa.gov/documentation/coursetutorial-links/) (https://dlabportal.jpl.nasa.gov/documentation/coursetutorial-links/)
- [Sphinx docs](https://dartslab.jpl.nasa.gov/dlabdocs/) (https://dartslab.jpl.nasa.gov/dlabdocs/)
- [Jupyter notebooks](https://dlabnotebooks.jpl.nasa.gov/hub/login) (https://dlabnotebooks.jpl.nasa.gov/hub/login)
- [DARTS Lab QA site](https://dartslab.jpl.nasa.gov/qa/) (https://dartslab.jpl.nasa.gov/qa/)

### 8.2. Design

### 8.3. Usage



**TBD:** Needs scrubbing. Notes brought over from [issue](#)

([https://dartsgitlab-internal.jpl.nasa.gov/darts-lab/jpl-internal/all-access/development/simulation-framework/dshellpp/-/issues/15#note\\_9058](https://dartsgitlab-internal.jpl.nasa.gov/darts-lab/jpl-internal/all-access/development/simulation-framework/dshellpp/-/issues/15#note_9058)).

### 8.4. Software

### 8.5. Raw documents

### 8.6. Sphinx documentation

#### 8.6.1. Darts/Dshell Data Inspection and Updating with DVar

##### 8.6.1.1. Motivation for DVar

- **What is DVar?**

The DVar C class provides functions to peek/poke C variables. Here's an example how DVar works: +

```
// This is C++ code
// Register a C/C++ variable with DVar
static double mass;
DVarDoubleLeaf dvar_mass("mass", "any description here", &mass);
```

```
+
You can also access the C/C++ variable through python:
+
```

```
# this is python code
from Dutils import DVar_Py
dvar_mass = DVar_Py.getDvar('mass') # the leading period is important
dvar_mass(10.0) # set the mass
print dvar_mass() # display the mass value
```

PYTHON

- **What DVar is used for**

- Access to all exposed data products
- Peek and poke values
- Consistent interface for all simulation data
- Connection for data-related call-backs

#### 8.6.1.2. Basics of data inspection using DVar

- **DVar Leaf Objects**

**DVar Leaf objects encapsulate most C basic types.**

- `BoolLeaf` - boolean
- `LongLongLeaf` - long integer
- `IntLeaf` - integer
- `PointerLeaf` - void pointer
- `EnumLeaf` - maps a string to/from an integer value
- `UIntLeaf` - unsigned integer
- `DoubleLeaf` - double
- `StringLeaf` - fixed length string
- `StdStringLeaf` - C++ string

- BoolVectorLeaf - array of boolean
- IntVectorLeaf - array of integers
- LongVectorLeaf - array of long integers
- UIntVectorLeaf - array of unsigned integers
- DoubleVectorLeaf - array of doubles
- FloatVectorLeaf - array of floats
- StringVectorLeaf - array of fixed length strings

- **DVar Branch Objects**

A Branch object is a container (a tree) of Leaf and Branch objects and is useful for representing C structures. + For example this C structure: +

```
struct {
int flag;
double quaternion[4];
}
```

```
+
can be represented by a DVar Branch object containing a DvarIntLeaf
and a DvarDoubleVectorLeaf.
* DVar call-back functions
```

+ Leaf objects allow some optional callback functions to be invoked when their data values are changed. The following types of callbacks are supported:

+

**watch**

A 'watch' callback will be invoked immediately after a DVar Leaf's value is changed.

**preGet**

A 'preGet' callback is invoked before the DVar Leaf's value is changed and has the opportunity to change the value that is for the new value of the DVar Leaf.

For examples of these two types of callbacks, see the DVar tutorial in the following section.

Other types of callbacks are available at the C++ level but only 'watch' and 'preGet' are available via Python. \* DVar Tutorial

+

**DVar basics tutorial**

► [Click to see the DshellEnv/test/test\\_DVar/script2.py script](#)

- Tips on using DVar + The DVar Branch class should be used to hold structures. + The DvarPointerLeaf class can be used to represent binary data (such as an array of terrain pixels). + If you need to create a new DVar class, it should be subclassed from the DVar Leaf base class. See how the DVar IntLeaf class (in src/DVar/IntLeaf.h) is implemented for an example.

## 9. DataRecorder

### 9.1. Background

#### 9.1.1. Reference & Source material

- [DataRecorder Doxygen docs](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/DataRecorder/html/) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/DataRecorder/html/)
- [Team talks](https://dlabportal.jpl.nasa.gov/resources/darts-lab-talks/) (https://dlabportal.jpl.nasa.gov/resources/darts-lab-talks/)
- [2019, 2020 course slides](https://dlabportal.jpl.nasa.gov/documentation/coursetutorial-links/) (https://dlabportal.jpl.nasa.gov/documentation/coursetutorial-links/)
- [Sphinx docs](https://dartslab.jpl.nasa.gov/dlabdocs/) (https://dartslab.jpl.nasa.gov/dlabdocs/)
- [Jupyter notebooks](https://dlabnotebooks.jpl.nasa.gov/hub/login) (https://dlabnotebooks.jpl.nasa.gov/hub/login)
- [DARTS Lab QA site](https://dartslab.jpl.nasa.gov/qa/) (https://dartslab.jpl.nasa.gov/qa/)

### 9.2. Design

#### 9.2.1. PlotJugglerRecorder

The PlotJugglerRecorder uses the 3rd party library PlotJuggler to save and view data; see their [documentation](https://www.plotjuggler.io/) (https://www.plotjuggler.io/) for more details on how to use PlotJuggler. The PlotJugglerRecorder is created with a set of parameters that can be used to modify its behavior; these are:

- port (*int, optional*) - Used to modify the PlotJuggler's default UDP port. (Default = 9870)
- launch (*bool, optional*) - Indicates whether to launch PlotJuggler or not. (Default = true)
- layout (*str, optional*) - If specified, indicates which PlotJuggler layout file to use. (Default = "")
- time\_dvar (*DVar pointer, optional*) - If specified, indicates which DVar to use for time (the x-axis in PlotJuggler). (Default = nullptr)

All of these parameters can be modified when creating the recording by specifying them as key-value pairs to `DataRecorder_Py.createRecorder`. Similarly, they can be passed as key-value pairs to `sim.record`. Note that while these are the defaults as specified in `PlotJugglerRecorder.h`, if invoked via `sim.record`, the `time_dvar` will be automatically be modified to point to `sim.specNode()["time"]`.

### 9.3. Usage

#### 9.3.1. PlotJugglerRecorder

It is easiest to create an instance of PlotJugglerRecorder via the `sim.record` method. It can be created using the following.

```
sim.record(data, "plotjuggler", **kwargs)
```

PYTHON

where `kwargs` are key-value pairs to pass on to the recorder itself: see the PlotJugglerRecorder design for more details. Once the PlotJugglerRecorder is created, you will receive a message that states: "Launching plotjuggler: Start the UDP stream then press enter to continue..." This message indicates that you must manually start the UDP stream on PlotJuggler; and then return to the terminal and press any button to continue the simulation: for a visual example of how to do so, see [this recording](https://dartslab.jpl.nasa.gov/technotes/Talks/2021-12-02-PlotJuggler-leake.mp4) (https://dartslab.jpl.nasa.gov/technotes/Talks/2021-12-02-PlotJuggler-leake.mp4).

The data passed to plot juggler can be one of two types: 1. A list of DVar's 2. A dictionary whose keys are strings and values are lists of DVars. If using the first option, then the name keyword argument to `sim.record` can be used to name the list. If using the second option, then the keys specify the name of the associated list of DVars.

### 9.4. Software

### 9.5. Raw Documents

### 9.6. Sphinx Documentation



#### Note

For doxygen module documentation for DataRecorder, see: [DataRecorder Module<index.html>](#)

#### DataRecorder Logging

##### 9.6.1. DataRecorder Logging

###### 9.6.1.1. Goals

DataRecorder is meant to support HDF5 and perform better than the Dwatch and Dstore.

- C++ solution for speed
- Save Dvar items and Python callables
- HDF5 output for post-analysis
- Save-as-you-go to avoid memory limitations
- Dwatch-style interface for user-friendly setup
- Partially implemented (basic functionality, some limitations)

###### 9.6.1.2. Specifying Variables to Log

- DataRecorder uses ONLY DVar spec strings (because logging is done in C++)

- Full DVar spec strings, eg:

```
.mbody.darts.bodies.Base.hinge.Q
```

- Strings that can be evaluated to produce DVar spec strings (ending in `.specString()`):

```
body.parentHinge().subhinge(0)['relVel'].specString()
```

- Python objects using Py\*Leaf objects (doubles, vectors of doubles, strings, etc)

- PyDoubleLeaf, PyDoubleVectorLeaf, PyStringLeaf, etc



MISSING DOXYGEN LINK: ``PyDoubleLeaf`, `PyDoubleVectorLeaf`, `PyStringLeaf`` doxygen links

- \*In future DataRecorder will support other python callables by converting to spec strings behind the scenes

### 9.6.1.3. HDF5 Data Structure

Each Dwatch-style `.cfg` file will create a group. For each group, DataRecorder stores the data in HDF5 Tables

Each table contains:

- DATA (for data, one row per instant of time)
  - Column headers are short names (from `.cfg` files or created from the end of the specstring).
- DATA\_ATTRIBUTES
  - Data type, number of elements, units, spec-string
  - Name (from Dwatch config files or from spec strings)

Data recording is optimized using the HDF5 Packet Table API. \* Assumes same data type is to be written again and again; ideal for data logging applications. \* Avoids redundant open/close calls that normal appending HDF5 API does behind the scenes. \* Each packet is composed of a compound data type that encapsulates the data types we plan to record. \* At each logging step, we pack the row of data into contiguous memory and then write it as a packet. \* This improves performance by a factor of five compared to the basic HDF5 API.

### 9.6.1.4. Dwatch-style interface

DataRecorder currently supports a Dwatch-style interface, DwatchHDF5 (which uses the `parametersFromDwatchHDF5` function).

- Reads Dwatch-style config data and fields to construct HDF5 file, group, and, fields
- Uses simulation to control writing output to DataRecorder HDF5

For an example of this, please see the `test_Dwatch` regtest `DataRecorder_regtest_Dwatch`.

## 9.6.2. DataRecorder Logging Example

### 9.6.2.1. Dwatch-style `.cfg` file

Note that most of these examples directly parallel the code in the `DataRecorder_regtest_Dwatch` regtest. If you have any problems implementing the following items, check the actual regtest script to get the latest, working version.

If you want to log a variable which is associated with a DVar object, by inserting lines like the following in a `.cfg` file (and handling the `.cfg` file as will be described below):

#### example.cfg

```
[state]
time = [ '.Dshell.time', "sec" ]

state = """{
  'type': 'string',
  'maxLength': 20,
  'function': 'getCurrentState',
  'description': 'current fsm state',
  'units': '',
}"""

Q = [ '.mbody.darts.bodies.CannonballBase.hinge.Q', 'm, rad' ]

xy = [ '.mbody.darts.bodies.CannonballBase.hinge.Q(0-1)', 'm' ]

x = [ '.mbody.darts.bodies.CannonballBase.hinge.Q(0)', 'm' ]

pos = ["cbbase.parentHinge().subhinge(0).specNode()['translation'].specString()", "m"]

vel = [ '.mbody.darts.bodies.CannonballBase.hinge.U(0-2)', "m/s, rad/sec" ]
```

INI

This will be processed later to create DataRecorder objects to log the specified items. A few notes:

- `time`, `Q`, and `vel` are straightforward spec strings

- `state` uses a user-defined python function that returns the current FSM state. For example, the function might look like this:

```
def getCurrentState():
    if len(fsm.current_state) > 0:
        state = fsm.current_state[0].name()
    else:
        state = ''
    return state
```

PYTHON

- The `xy` spec string includes slicing syntax: Only the first two elements will be logged. A single index can be specified like: `'(1)'` to get only the second element.
- Since getting the full specstring for `pos` could be tricky in some cases, here we get the `'cbbase'` body object and query it as shown to get the desired DVar object, then we invoke `'.specString()'` at the end to return the actual spec string.
- The trailing empty string means that this data has no units.

### 9.6.2.2. Supported Conversions

However, if we want to log something that is not a direct DVar object, we will need to construct a 'Py\*Leaf' object to create DVar object which will evaluate the underlying function as necessary for logging. The DataRecorder DwatchHDF5 has conversion mechanisms to do the underlying DVar work from a simple specification, as shown for the `time` variable above.

It is still possible to create DVar object in the script and pass its spec string in the `.cfg` variable definitions (as we did for `time` above). Once a DVar object is defined, the spec string can be obtained by using its `'.specString()'` function.

The general form of the conversion specification is:

```
varname = """{
    'type': '<type>',
    '<other arguments>',
    'description': '<description, optional>',
    'units': '<units, optional>',
}"""
```

PYTHON

where `<type>` is `'string'`, `'double'`, `'doubleVector'`, `'doubleNewUnits'`, or `'split'`.

For these specifications, the information about the data item is in a simple dictionary. The definition is included in triple-quotes as shown in order to be split on separate lines in the `.cfg` file. If you use single quotes, the definition must be one one line.

#### 9.6.2.2.1. Handling a python callable

Examples of the supported conversion mechanisms:

#### “ • String functions

```
strname = """{
    'type': 'strung',
    'maxLength': 20,
    'function': '<name of function that returns a string>',
    'description': '<description, optional>'
}"""
```

PYTHON

Where `'<name of function>'` is the string name of the function or callable object (without parentheses). The function or callable cannot take arguments.

`'maxLength'` is the maximum expected length that the function will return. The underlying DVar object cannot be created without this length.

#### • Double functions

If you have a function that returns a single double value, use this specification:

```
dname = """{
    'type': 'double',
    'function': '<name of function that returns a double>',
    'description': '<description, optional>',
    'units': '<name of units for the returned value, optional>'
}"""
```

PYTHON

Where `'<name of function>'` is the string name of the function or callable object (without parentheses) that returns the double. The function or callable cannot take arguments.

#### • Double vector functions

If you have a function that returns a vector or array of doubles, use this specification:

```

dvname = """{
    'type': 'doubleVector',
    'length' : 3,
    'function': '<name of function that returns the vector of doubles>',
    'description': '<description, optional>',
    'units': '<name of units for the returned values, optional>'
}"""

```

Where '<name of function>' is the string name of the function or callable object (without parentheses) that returns the double. The function or callable cannot take arguments.

The 'length' of the resulting vector of doubles must be specified.

#### 9.6.2.2.2. Handling unit conversion on the fly

If you need to log data with unit conversion, this can be accomplished using the following syntax in your .cfg file:

```

“““
state = """{
    'type' : 'doubleNewUnits',
    'specString':
'.dot.separated.dvar.specstring',
    'new_units': 'm',
    'scaleFactor': 1000.0
}"""
PYTHON

```

#### A few notes

- All arguments shown are required
- 'specString' can be either for a single double or a vector of doubles.
- 'specString' supports the '.specString0' syntax or slicing as described above.
- 'new\_units' is the string name of the new units
- 'scaleFactor' is the conversion factor to be used. It is up to the user to specify the conversion constant. Note that this may be made optional in the future by using the builtin quantity info.
- $converted\_value = scaleFactor * old\_value$

#### 9.6.2.2.3. Splitting a vector into scalar components

If you need to log data and split a vector into several scalar components, this can be accomplished using the following syntax in your .cfg file:

```

“““
state = """{
    'type' : 'split',
    'specString': '.dot.separated.dvar.specstring',
    'new_units': 'km', # OPTIONAL
    'scaleFactor': 1/1000.0, # ONLY needed if 'new_units' is
given
    'suffixes': ('_x', '_y', '_z')
}"""
PYTHON

```

The 'suffixes' will be appended on the name of the original DVar item to make the names of the separate components. In this example, the 'state' vector will be split into 3 scalars named: 'state\_x', 'state\_y', 'state\_z'.

#### A few notes

- 'specString' must be for a vector of doubles (or ints)
- 'specString' supports the '.specString0' syntax or slicing as described above.
- the number of suffixes given must match the length of the vector/slice.
- If 'new\_units' is given, the data must be a vector of doubles. If omitted, no unit conversion occurs.
- It is up to the user to specify the conversion constant.

#### 9.6.2.3. Creating the DataRecorder logging objects

In order to use this .cfg file to set up logging in the run script, we add lines like this to the run script:

```

from DataRecorder.parametersFromDwatchHDF5Cfg import setupDwatchHDF5
from Dshell.Dshell_Py import CallbackLoggingHandler

dw_objects = []

# Set up regular logging
for cfg in ['dwatch/trajSpecs_withpython.cfg']:
    dw = setupDwatchHDF5(cfg, sim.dvarContainer(), namespace)
    drlogger = CallbackLoggingHandler()
    drlogger.addLogger(dw.logger())
    drlogger.registerLoggers(sim)
    dw_objects.append(dw)

```

PYTHON

where `namespace` is a namespace for objects that will be accessed by the Dwatch specs. For instance, based on the example at the top of this page, `namespace` might be defined like this:

```

namespace = {
    'sim' : sim,
    'cbbase' : sim.mbody().body('CannonballBase', 0, True),
    'getCurrentState' : getCurrentState
}

```

PYTHON

Each `dw` created in the loop above will create one HDF5 file, as specified by the top-level `.cfg` file `'filename'` setting.

Note that these loggers are invoked at the end of every IO step (in the `EndIOStep` phase).

#### 9.6.2.3.1. FSM Controlled Logging

You may also create loggers that are invoked by FSMs. For example, add this to your run script:

```

# Set up the FSM-based recorder
# (These Dwatch items will only be logged as FSMs decide)
dw2 = setupDwatchHDF5('dwatch/eventSpecs.cfg', sim.dvarContainer(), namespace)
sim.resultSpecs([dw2])
dw_objects.append(dw2)

```

PYTHON

The line `sim.resultSpecs([dw2])` lets the simulation object take control of the DataRecorder objects so that it can do logging after each step.

The FSM will do the invoke the `dw2` DataRecorder logging object when the FSM requests. For instance, if the `watchAllOnExitStateAction` FSM action function from `DshellCommon/fsm/FSMUtils.py` is used in the state table, then it will invoke the function `onState()` from `DataRecorder/python/DwatchHDF5.py` which then calls the logger's `update()` function on all of the loggers that the simulation has had registered using the `sim.resultsSpecs()` call.

Alternately, your FSM functions can call the logger's `.update()` functions directly as needed.

#### 9.6.2.4. Running the simulation with logging

Later when the simulation is ready to run, simply step the simulation and logging will occur at the end of each IOStep (for the regular logging). FSM logging will occur as FSMs dictate.

```

sim.reset() # initializes state machine and time
fsm.step()

# Run the sim
for i in range(10000):
    sim.step():
    if sim.isTerminated():
        break

# Close all loggers
for dwobj in dw_objects:
    dwobj.close()

```

PYTHON

#### 9.6.2.5. Logging the initial state

Normally the logging happens at the end of every IO step. However, run scripts can force immediate logging at any point like this:

```

for dwobj in dw_objects:
    dwobj.logger().update()

```

PYTHON

#### 9.6.2.6. Recording Simulation-Level Meta Data

In the HDF5 files produced by DataRecorder objects, the meta data for a column of data (from the DATA data set) is defined in the DATA\_ATTRIBUTES data set. Each item in the DATA\_ATTRIBUTES data set that correspond to data item 'xyz' will the `fieldName` attribute 'xyz', then attribute name and value. Items that are applicable to the entire run can be saved in the DATA\_ATTRIBUTES data set by using a blank `fieldName` name. For instance, in order to embed the start time and a few other attributes of the simulation in the HDF5 file, we can add the following lines to the run script:

```

# Write attributes to the HDF5 DATA_ATTRIBUTES table
# The attributes are passed through a dictionary of key/value pairs.
attribute_dict = {}
attribute_dict['start_time'] = time.strftime("%d/%m/%Y %H:%M:%S")
attribute_dict['host_name'] = socket.gethostname()
attribute_dict['script_filename'] = sys.argv[0]

from DataRecorder.parametersFromDwatchHDF5Cfg import writeDwatchHDF5Attributes
writeDwatchHDF5Attributes(dw_objects, attribute_dict)

```

PYTHON



A similar operation can be done at the end of the simulation to embed the ending time of the simulation.

DataRecorder Module Regtests

9.6.3. DataRecorder Regression Tests

9.6.3.1. test\_Dwatch

**Example Dwatch usage script**

▶ [Click to see the test/test\\_Dwatch/script\\_withpython.py script](#)

**The corresponding Dwatch config file**

▶ [Click to see the test/test\\_Dwatch/dwatch/state\\_withpython.cfg script](#)

# 10. Dshell++Scripts

## 10.1. Background

### 10.1.1. Reference & Source material

- [Dshell++Scripts Doxygen docs](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/Dshell++Scripts/html/) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/Dshell++Scripts/html/)
- [Team talks](https://dlabportal.jpl.nasa.gov/resources/darts-lab-talks/) (https://dlabportal.jpl.nasa.gov/resources/darts-lab-talks/)
- [2019, 2020 course slides](https://dlabportal.jpl.nasa.gov/documentation/coursetutorial-links/) (https://dlabportal.jpl.nasa.gov/documentation/coursetutorial-links/)
- [Sphinx docs](https://dartslab.jpl.nasa.gov/dlabdocs/) (https://dartslab.jpl.nasa.gov/dlabdocs/)
- [Jupyter notebooks](https://dlabnotebooks.jpl.nasa.gov/hub/login) (https://dlabnotebooks.jpl.nasa.gov/hub/login)
- [DARTS Lab QA site](https://dartslab.jpl.nasa.gov/qa/) (https://dartslab.jpl.nasa.gov/qa/)

## 10.2. Design

## 10.3. Usage



TBD: Dshell++Scripts documentation TBD.

## 10.4. Software

## 10.5. Raw documents

## 10.6. Sphinx documentation

### 10.6.1. Logging with DebugLog

Our DebugLog system is based on the Boost Log library. See:

<http://boost-log.sourceforge.net> (http://boost-log.sourceforge.net/libs/log/doc/html/index.html)

The [DebugLog](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/Dshell++Scripts/html/namespaceDebugLog.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/Dshell++Scripts/html/namespaceDebugLog.html) logging library is in C. We use {DshellppScripts\_DebugLog\_namespace\_doxygen\_ref} in C as well as in Python (via SWIG wrapping).

#### 10.6.1.1. Logging in C++

```
// Get the global Source. We could also create our own instance if we wanted.
DebugLog::Source &dsrc = DebugLog::Source::debugLogSource()

// Instead of printf("My debug message.")
dsrc.debug("My debug message.")

// Instead of printf("My trace message.")
dsrc.trace("My trace message.")

// Instead of printf("My deprecated message.")
dsrc.deprecated("My deprecated message.")

// Instead of printf("My info message.")
dsrc.info("My info message.")

// Instead of printf("My warning message.")
dsrc.warning("My warning message.")

// Instead of printf("My error message.")
dsrc.error("My error message.")
```

C++

#### 10.6.1.2. Logging in Python

```
from Dshell import DebugLog_Py
dsrc = DebugLog_Py.Source.debugLogSource()

dsrc.debug('debug message')
dsrc.trace('trace message')
dsrc.deprecated('deprecated message')
dsrc.info('info message')
dsrc.warning('warning message')
dsrc.error('error message')
```

PYTHON

#### 10.6.1.3. Changing Verbosity

By default the {DshellppScripts\_DebugLog\_Source\_debugLogSource\_method\_doxygen\_uri[DebugLog::Source::debugLogSource]} is connected to standard out. The verbosity of the source is by default at level ERROR. This means only messages higher than ERROR will be sent to the log sink. The verbosity of the standard out sink is by default also at level ERROR. This means the sink will only print out error messages higher than ERROR.

The available levels in increasing order of severity are defined by the [DebugLog::Verbosity](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/Dshell++Scripts/html/namespaceDebugLog.html#a8ac9951d612c31fb3eafda006fd4035d)

(https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/Dshell++Scripts/html/namespaceDebugLog.html#a8ac9951d612c31fb3eafda006fd4035d) enum and are as follows:

- `DebugLog::Verbosity::ALL` - All [DebugLog](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/Dshell++Scripts/html/namespaceDebugLog.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/Dshell++Scripts/html/namespaceDebugLog.html) messages are enabled.

- DebugLog::Verbosity::DEBUG - All debug level messages are generated. Used for verbose output to support debugging needs.
- DebugLog::Verbosity::TRACE - All messages to trace program execution are generated. Typically used to trace assemblies creation, SWIG module loading etc.
- DebugLog::Verbosity::DEPRECATED - All messages about deprecated usage are generated. Used to warn out the use of deprecated methods that may go way in the near future.
- DebugLog::Verbosity::INFO - All 'info' messages about usage are generated. Used to provide non-warning information to users.
- DebugLog::Verbosity::WARNING - All warning messages are generated. This is the default setting for normal development use.
- DebugLog::Verbosity::ERROR - All error messages are generated.
- DebugLog::Verbosity::NONE - All [DebugLog](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/Dshell++Scripts/html/namespaceDebugLog.html) messages are disabled.

Choosing severity levels lower in the list result in less messages being outputted. For example, WARNING will allow warning(), deprecated(), and error() messages to be outputted. Messages from trace() and debug() will not be outputted.

To change the verbosity of the logging source do the following.

```
from Dshell import DebugLog_Py
dsrc = DebugLog_Py.Source.debugLogSource()
dsrc.verbosity(DebugLog_Py.WARNING)
```

PYTHON

To change the verbosity of the standard out sink do the following.

```
from Dshell import DebugLog_Py
sink = DebugLog_Py.StdoutSink.singleton()
sink.verbosity(DebugLog_Py.ALL)
```

PYTHON

#### 10.6.1.4. DebugLog Tutorials

The following tutorial demonstrates using [DebugLog](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/Dshell++Scripts/html/namespaceDebugLog.html) with Python. The zipfile download shown at the top of the tutorial includes a C test program that demonstrates using DebugLog from C.

{

#### 10.6.2. Regression tests

##### Basic DebugLog regtest

▶ [Click to see the Dshell++ScriptsTest/test/test\\_debug\\_log/script.py script](#)

##### Basic DebugLog regtest with DVar

▶ [Click to see the Dshell++ScriptsTest/test/test\\_debug\\_log\\_dvar/script.py script](#)

##### Dshell++ Model auto-generator regtest

▶ [Click to see the Dshell++ScriptsTest/test/test\\_dshell\\_auto\\_gen/script.py script](#)

## 11. SOA

### 11.1. Background

#### 11.1.1. Reference & Source material

- [SOA Doxygen docs](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SOA/html/) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SOA/html/)
- [Team talks](https://dlabportal.jpl.nasa.gov/resources/darts-lab-talks/) (https://dlabportal.jpl.nasa.gov/resources/darts-lab-talks/)
- [2019, 2020 course slides](https://dlabportal.jpl.nasa.gov/documentation/coursetutorial-links/) (https://dlabportal.jpl.nasa.gov/documentation/coursetutorial-links/)
- [Sphinx docs](https://dartslab.jpl.nasa.gov/dlabdocs/) (https://dartslab.jpl.nasa.gov/dlabdocs/)
- [Jupyter notebooks](https://dlabnotebooks.jpl.nasa.gov/hub/login) (https://dlabnotebooks.jpl.nasa.gov/hub/login)
- [DARTS Lab QA site](https://dartslab.jpl.nasa.gov/qa/) (https://dartslab.jpl.nasa.gov/qa/)
- Release notes appendices

### 11.2. Design

The key classes supported are described below:

#### 11.2.1. SOAVector vector class



**MISSING:** Add description of SOAVector class

See the [SOAVector Doxygen documentation](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SOA/html/classSOAVector.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SOA/html/classSOAVector.html) for the C++ documentation of this class.

#### 11.2.1.1. SOAVector3 vector class



**MISSING:** Add description of SOAVector3 class

#### 11.2.1.2. SOASpatialVector vector class



**MISSING:** Add description of SOASpatialVector class

#### 11.2.2. SOAMatrix matrix class



**MISSING:** Add description of SOAMatrix class

#### 11.2.2.1. SOAMatrix33 matrix class



**MISSING:** Add description of SOAMatrix33 class

#### 11.2.2.2. SOARotationMatrix matrix class



**MISSING:** Add description of SOARotationMatrix class

#### 11.2.3. SOAQuaternion unit quaternion class



**MISSING:** Add description of SOAQuaternion class

#### 11.2.4. SOAHomTran homogeneous transform class



**MISSING:** Add description of SOAHomTran class

See the [SOAHomTran Doxygen documentation](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SOA/html/classSOAHomTran.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SOA/html/classSOAHomTran.html) for the C++ documentation of this class.

#### 11.2.5. SOASpatialInertia spatial inertia class



**MISSING:** Add description of SOASpatialInertia class

See the [SOASpatialInertia Doxygen documentation](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SOA/html/classSOASpatialInertia.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SOA/html/classSOASpatialInertia.html) for the C++ documentation of this class.

### 11.3. Usage

**THE FOLLOWING IS TO BE DISCARDED (TESTING)**

First reference is Simulation time stepping

second reference is Multi-rate models

## 11.4. Software

### 11.4.1. SOARodriguesParam

#### 11.4.1.1. Can we limit SOARodriguesParam angle to +/- pi range?



TBD: Flesh out this section on SOARodriguesParam

The magnitude of the coords give us the angle. One would think that there is no physical reason to allow angles outside the  $[-\pi, \pi]$  range. Keeping the angle limited also allows us to avoid the singularity near the  $2\pi$  multiples. Previously we only worked with +ive angles and unlimited values. This would need to change to allow + and - values. We still would need to use an epsilon near angle of 0 since the velocity formulas need to behave properly. This change will also mean that there is no reason to change charts. We will always be centered around 0.

**NOT SO FAST:** There is a problem with this. Even though the transform value does not change with  $2\pi$  changes, the omega to  $\dot{u}$  value does depend on the value of  $u(n, \theta)$ . For while the SE3 for  $\theta$  or  $(\theta + 2\pi)$  is the same, the value of  $\{\dot{u}\}$  is different for  $u(n, \theta)$  and  $u(n, \theta + 2\pi)$ !. Why this matter in our usage is that if the integrator is free-running with arbitrary  $\theta$ , while we are restricting  $\theta$  to  $\pm\pi$  range, then the computed value of  $\{\dot{u}\}$  with the restricted  $\theta$  value will be wrong for the unrestricted  $\theta$ , and the integration will be incorrect. So we **cannot** internally restrict the range of  $\theta$  within the SOARodriguesParam implementation. The integrator values and the SOARodriguesParam values and usage have to be **consistent**. Currently the integrator supplies the  $u$  to the hinge via `setGenCoord`, and the hinge gives back the  $\dot{u}$  to the integrator. So if the hinge internally does the  $2\pi$  wrapping, then it will be giving back the wrong  $\{\dot{u}\}$  to the integrator.

- **OPTIONS:** We have two different use cases where this is an issue.
  - **Integration:** For this we need to avoid singularities to keep the  $\{\dot{u}\}$  computation working. This has to be balanced against the need for soft resets required when charts are changed.
  - **Iteration:** This is needed when implicit integrators and their iterators or inverse kinematics are used. Here we cannot be switching charts since the iterator wants to sample the coordinate space and do what it needs to do in an unconstrained way. The good news here is that there is no need for  $\{\dot{u}\}$ , and so it does not matter if we are at singularities. **Resolution:** Chart switching can **only** be done in between integration steps and nowhere else. So if can tell the hinge that it is free mode (i.e. in between steps) and make a function call to allow the hinge to switch charts and signal a soft reset, and lock down the chart for the rest of the time we should be OK. So we should
    - add a `sanitizeCoords()` method to subhinges that is typically a noop, but whose job is to sanitize the coords by switching charts for Rodrigues params, or Euler method for Euler angles.
    - before taking a step, add a `stateFromIntegrator` call, then a call to `sanitizeCoords()` for all subhinges for them to check and change charts if necessary, and request a soft reset.
      - and after this make the regular integrate call.

## 11.5. Raw documentation



TBD: Need scrubbing before integration.

### 11.5.1. SOA: Implement default object value printouts using `pydump()`



TBD: Needs scrubbing. Notes brought over from [issue](https://dartsgitlab-internal.jpl.nasa.gov/darts-lab/jpl-internal/all-access/development/numerics/soa/-/issues/2) (https://dartsgitlab-internal.jpl.nasa.gov/darts-lab/jpl-internal/all-access/development/numerics/soa/-/issues/2).

This is a feature proposal to implement default object value printouts using `pydump()` with the `str()` and `repr()` methods.

E.g.

```
import SOA_Py
foo = SOA_Py.SOAVector((1, 2, 3))
foo
```

PYTHON

would produce:

```
<SOA_Py.SOAVector; proxy of <Swig Object of type 'SOAVector *' at 0x7f5188b0eb10> >
array([1., 2., 3.])
```

PYTHON

and

```
import SOA_Py
foo = SOA_Py.SOAVector((1, 2, 3))
print(foo)
```

PYTHON

would produce:

```
[1. 2. 3.]
```

PYTHON

The reason for proposing to use `pydump()` as opposed to `dumpString()` is because Python string dumps gracefully handle large arrays by default, so, e.g.

```
import SOA_Py
foo = SOA_Py.SOAMatrix(100, 100)
foo
```

PYTHON

would produce something like:

```
<SOA_Py.SOAMatrix; proxy of <Swig Object of type 'SOAMatrix *' at 0x7f518a8954e0> >

array([[2.50185949872382e-316, 6.91633697090979e-310,
        6.91633653521029e-310, ..., 6.91633653496207e-310,
        6.91633653495575e-310, 6.91633653513914e-310],
       [6.91633653494626e-310, 6.91633653330912e-310,
        6.91633653492096e-310, ..., 6.91633699547078e-310,
        6.91633653366880e-310, 6.91633698004644e-310],
       [6.91633653320636e-310, 6.91633653320241e-310,
        6.91633698005119e-310, ..., 6.91633653304272e-310,
        6.91633653303640e-310, 6.91633653312336e-310],
       ...,
       [0.00000000000000e+000, 0.00000000000000e+000,
        0.00000000000000e+000, ..., 0.00000000000000e+000,
        0.00000000000000e+000, 0.00000000000000e+000],
       [0.00000000000000e+000, 0.00000000000000e+000,
        0.00000000000000e+000, ..., 0.00000000000000e+000,
        0.00000000000000e+000, 0.00000000000000e+000],
       [0.00000000000000e+000, 0.00000000000000e+000,
        0.00000000000000e+000, ..., 0.00000000000000e+000,
        0.00000000000000e+000, 0.00000000000000e+000],
       [0.00000000000000e+000, 0.00000000000000e+000,
        0.00000000000000e+000, ..., 0.00000000000000e+000,
        0.00000000000000e+000, 0.00000000000000e+000]])
```

PYTHON

and

```
import SOA_Py
foo = SOA_Py.SOAMatrix(100, 100)
print(foo)
```

PYTHON

would produce something like:

```
[[2.50185949872382e-316 6.91633697090979e-310 6.91633653521029e-310 ...
  6.91633653496207e-310 6.91633653495575e-310 6.91633653513914e-310]
 [6.91633653494626e-310 6.91633653330912e-310 6.91633653492096e-310 ...
  6.91633699547078e-310 6.91633653366880e-310 6.91633698004644e-310]
 [6.91633653320636e-310 6.91633653320241e-310 6.91633698005119e-310 ...
  6.91633653304272e-310 6.91633653303640e-310 6.91633653312336e-310]
 ...
 [0.00000000000000e+000 0.00000000000000e+000 0.00000000000000e+000 ...
  0.00000000000000e+000 0.00000000000000e+000 0.00000000000000e+000]
 [0.00000000000000e+000 0.00000000000000e+000 0.00000000000000e+000 ...
  0.00000000000000e+000 0.00000000000000e+000 0.00000000000000e+000]
 [0.00000000000000e+000 0.00000000000000e+000 0.00000000000000e+000 ...
  0.00000000000000e+000 0.00000000000000e+000 0.00000000000000e+000]
 [0.00000000000000e+000 0.00000000000000e+000 0.00000000000000e+000 ...
  0.00000000000000e+000 0.00000000000000e+000 0.00000000000000e+000]]
```

PYTHON

This could be implemented by adding default `str()` and `repr()` methods to `SOABase` as follows:

```
import abc
.
.
.
@abc.abstractmethod
def pydump(self):
    """Abstract pydump() method."""
    return "Default object value dump unavailable: pydump() method unimplemented."

def __str__(self):
    return str(self.pydump())

def __repr__(self):
    return "\n\n".join((_swig_repr(self), repr(self.pydump())))
```

PYTHON

### 11.5.1.1. Implementation



This has now been implemented.

The main additions in `SOA/swig/SOA.i`:

Line 66:

```
%pythonbegin %{
# from __future__ import print_function
# from __future__ import absolute_import
# from __future__ import division
import re
import abc
import Dshell.DebugLog_Py, numpy; numpy.set_printoptions(precision=16)
%}
```

PYTHON

Line 77:

```
%pythoncode %{
def _swig_repr(self):
    """This overrides the SWIG-generated _swig_repr() method in order to
    provide custom default object value outputs using pydump()."""
    name = self.__module__ + "." + self.__class__.__qualname__
    value = repr(self.pydump())
    value = re.sub("^array\\(", name + "\\n" + (len("array(") * " "), value)
    return value
%}
```

PYTHON

Line 236:

```
%pythoncode %{

def pydump(self, indent=''):
    result = {}
    result['mass'] = self.mass()
    result['b2cm'] = self.B2CM().pydump(indent)
    result['mp'] = self.mp().pydump(indent)
    result['cmInertia'] = self.cm_inertia().pydump(indent)
    result['inertia'] = self.inertia().pydump(indent)
    return result

def __repr__(self):
    name = self.__module__ + "." + self.__class__.__qualname__
    cm_inertia_name = self.cm_inertia().__module__ + "." + self.cm_inertia().__class__.__qualname__
    inertia_name = self.inertia().__module__ + "." + self.inertia().__class__.__qualname__
    cm_inertia_offset = len("cmInertia: " + cm_inertia_name + "[")
    inertia_offset = len("inertia: " + inertia_name + "[")
    return (name + "\\n  {" +
            ("\\n" + " " * 5).join((
                "mass": " + str(self.mass()),
                "b2cm": " + re.sub("\\n\\s*", "", self.B2CM().__repr__()),
                "mp": " + re.sub("\\n\\s*", "", self.mp().__repr__()),
                "cmInertia": " + re.sub("\\n\\s*", "\\n" + (" " * (5 + cm_inertia_offset)),
                    re.sub("\\n\\s*", "", self.cm_inertia().__repr__(1))),
                "inertia": " + re.sub("\\n\\s*", "\\n" + (" " * (5 + inertia_offset)),
                    re.sub("\\n\\s*", "", self.inertia().__repr__(1)))) + "}")

def __str__(self):
    return (" + "\\n ".join((str(self.mass()),
                            self.B2CM().__str__(),
                            self.mp().__str__(),
                            self.cm_inertia().__str__().replace("\\n", "\\n "),
                            self.inertia().__str__().replace("\\n", "\\n "))) + ")"

%}
```

PYTHON

Line 277:

```
@abc.abstractmethod
def pydump(self):
    """Abstract pydump() method."""
    return "Default object value dump unavailable: pydump() method unimplemented."

def __repr__(self):
    try:
        strthis = "proxy of " + self.this.__repr__()
    except __builtin__.Exception:
        strthis = ""
    value = repr(self.pydump())
    return "<%s.%s; %s >\\n\\n%s" % (self.__class__.__module__, self.__class__.__name__, strthis, value)

def __str__(self):
    return str(self.pydump())

%}
```

PYTHON

The main additions in SOA/swig/SOAHomTran.i :

Line 66:

```

def __repr__(self):
    name = self.__module__ + "." + self.__class__.__qualname__
    if self.getTranslation().isZero() and self.getQuaternion().isIdentity():
        return name + "()"
    elif self.getTranslation().isZero():
        return name + "\n" + (" " * 4) + "(" + re.sub("\n\s*", "", self.getQuaternion().__repr__()) + ",)"
    elif self.getQuaternion().isIdentity():
        return name + "\n" + (" " * 4) + "(" + re.sub("\n\s*", "", self.getTranslation().__repr__()) + ",)"
    return (name + "\n" + (" " * 4) + "(" + re.sub("\n\s*", "", self.getQuaternion().__repr__()),
            re.sub("\n\s*", "", self.getTranslation().__repr__())) + ")")

def __str__(self):
    if self.getTranslation().isZero() and self.getQuaternion().isIdentity():
        return "()"
    elif self.getTranslation().isZero():
        return "(" + self.getQuaternion().__str__() + ",)"
    elif self.getQuaternion().isIdentity():
        return "(" + self.getTranslation().__str__() + ",)"
    return "(" + "\n ".join((self.getQuaternion().__str__(),
                            self.getTranslation().__str__())) + ")"

```

## 11.6. Sphinx documentation

Go to active rot spot. The key classes supported are described below:

### 11.6.1. SOAVector vector class



**MISSING:** Add description of SOAVector class

See the [SOAVector Doxygen documentation](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SOA/html/classSOAVector.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SOA/html/classSOAVector.html) for the C++ documentation of this class.

#### 11.6.1.1. SOAVector3 vector class



**MISSING:** Add description of SOAVector3 class

#### 11.6.1.2. SOASpatialVector vector class



**MISSING:** Add description of SOASpatialVector class

### 11.6.2. SOAMatrix matrix class



**MISSING:** Add description of SOAMatrix class

#### 11.6.2.1. SOAMatrix33 matrix class



**MISSING:** Add description of SOAMatrix33 class

#### 11.6.2.2. SOARotationMatrix matrix class



**MISSING:** Add description of SOARotationMatrix class

### 11.6.3. SOAQuaternion unit quaternion class



**MISSING:** Add description of SOAQuaternion class

### 11.6.4. SOAHomTran homogeneous transform class



**MISSING:** Add description of SOAHomTran class

See the [SOAHomTran Doxygen documentation](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SOA/html/classSOAHomTran.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SOA/html/classSOAHomTran.html) for the C++ documentation of this class.

### 11.6.5. SOASpatialInertia spatial inertia class



**MISSING:** Add description of SOASpatialInertia class

See the [SOASpatialInertia Doxygen documentation](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SOA/html/classSOASpatialInertia.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SOA/html/classSOASpatialInertia.html) for the C++ documentation of this class.

### 11.6.6. Rotational Conventions

The Darts/Dshell framework assumes specific rotational conventions. In the analysis of rotations, there are several issues:



- Whether rotations are "passive" or "active"
- Which element of a quaternion is the scalar part

### 11.6.6.1. Active vs Passive Rotations

In *active* rotations, the new coordinate system is assumed to be the same as the old coordinate system but the rotation matrix describes how the points in the old coordinate system are transformed in the new coordinate system. Active rotations are often used in graphics computations.

In *passive* rotations, the point remains in the same place (conceptually) but the rotation defines how its coordinate representation changes. This is the approach that is often used in robotics and is also used in the Dartslab code (particularly the Dartslab SOA library).

For a good reference on active and passive rotations, please see this Wikipedia article on active and passive transformations:

[Wikipedia article on active and passive transformations](http://en.wikipedia.org/wiki/Active_and_passive_transformation) ([http://en.wikipedia.org/wiki/Active\\_and\\_passive\\_transformation](http://en.wikipedia.org/wiki/Active_and_passive_transformation))

### 11.6.6.2. Quaternion Scalar

The other issue is which element of the quaternion 4-vector is the scalar part. In the `SOAQuaternion` class, the 3-vector part of the quaternion is actually stored in a separate member variable from the scalar part. However, `SOAQuaternion` does support some vector-like operations such as array-access operations. In these operations, in the Dvar representation of quaternions, and the conversions of a quaternion to a vector for output, the scalar part is assumed to be the last entry in the vector.

### 11.6.6.3. Rotation Conventions Reference

For a complete description of how rotations are defined and used in Darts/Dshell, please consult the *Attitude Representations* appendix from Abhi Jain's book on **Robot and Multibody Dynamics: Analysis and Algorithms** (Springer, 2011):

- [Robot and Multibody Dynamics: Analysis and Algorithms](http://www.amazon.com/Robot-Multibody-Dynamics-Analysis-Algorithms/dp/1441972668) (<http://www.amazon.com/Robot-Multibody-Dynamics-Analysis-Algorithms/dp/1441972668>)
- [Appendix B - Attitude Representations <attitude-representations-jain.pdf>](#) (== click to download)

### 11.6.6.4. Quaternion Operations

To see how to do basic quaternion operations (in python), see this tutorial:

#### SOAQuaternion regtest

▶ [Click to see the SOA/test/test\\_SOAQuaternion/quaternion\\_tutorial.py script](#)

### 11.6.7. Inertia Tensor

See the following for information concerning the inertia tensor, products of inertia integral sense, and inputs/outputs to SOASpatialInertia:

Inertia tensor negative/positive sense

## 11.7. Inertia Tensor and Products of Inertia Integral Sense

### 11.7.1. Inertia Tensor

The inertia tensor ( $\mathcal{J}$ ) is like the effective "rotational mass" of a rigid body and relates the angular momentum ( $L$ ) to the angular velocity of the body ( $\omega$ )—all about the center of mass:

$$L = \mathcal{J}\omega$$

In the book [Robot and Multibody Dynamics: Analysis and Algorithms](http://www.amazon.com/Robot-Multibody-Dynamics-Analysis-Algorithms/dp/1441972668) (<http://www.amazon.com/Robot-Multibody-Dynamics-Analysis-Algorithms/dp/1441972668>) [1], the inertia tensor about a point  $k$  is defined as:

$$\mathcal{J}(k) = - \int_{\Omega} \tilde{\ell}(k, \mathbf{x}) \tilde{\ell}(k, \mathbf{x}) \rho(\mathbf{x}) d\Omega(\mathbf{x})$$

where

$\mathbf{x}$  = the location of a point on the body  $\Omega$

and [2]

$$\ell = \ell(k, \mathbf{x}) = \text{position of point } \mathbf{x} \text{ with respect to point } k = \begin{bmatrix} x \\ y \\ z \end{bmatrix}$$

$$\tilde{\ell} = \begin{pmatrix} 0 & -z & y \\ z & 0 & -x \\ -y & x & 0 \end{pmatrix}$$

and

$\rho(\mathbf{x})$  = density at point  $\mathbf{x}$

$d\Omega(\mathbf{x})$  = a differential mass at point  $\mathbf{x}$  on the body  $\Omega$

Therefore,

$$\tilde{\ell}\tilde{\ell} = \ell\ell^* - \ell^*\ell I = \begin{pmatrix} -y^2 - z^2 & xy & xz \\ xy & -x^2 - z^2 & yz \\ xz & yz & -x^2 - y^2 \end{pmatrix}$$

Replacing  $\rho(x)d\Omega(x)$  with  $dm$ ,

$$\mathcal{J}(k) = \begin{bmatrix} \int(y^2 + z^2)dm & -\int xydm & -\int xzdm \\ -\int xydm & \int(x^2 + z^2)dm & -\int yzdm \\ -\int xzdm & -\int yzdm & \int(x^2 + y^2)dm \end{bmatrix}$$

Thus, the inertia tensor is a symmetric tensor that depends on the mass distribution of the body and on the choice of origin for the coordinate system.

### 11.7.2. Products of Inertia - Negative Integral Sense

The inertia tensor is commonly represented as follows [3]:

$$\mathcal{J} = \begin{bmatrix} \mathcal{J}_{xx} & \mathcal{J}_{xy} & \mathcal{J}_{xz} \\ \mathcal{J}_{yx} & \mathcal{J}_{yy} & \mathcal{J}_{yz} \\ \mathcal{J}_{zx} & \mathcal{J}_{zy} & \mathcal{J}_{zz} \end{bmatrix}$$

Where

$$\begin{aligned} \mathcal{J}_{xx} &= \int(y^2 + z^2)dm \\ \mathcal{J}_{xy} &= -\int xydm \\ &\text{etc.} \end{aligned}$$

The off-diagonal elements ( $\mathcal{J}_{xy}$ ,  $\mathcal{J}_{xz}$ , etc.) of the inertia tensor are often referred to as the *products of inertia*. Since these products of inertia include the negative sign, they are referred to as being in "negative integral sense". This is the sense for [SOA\_inertia\_tensor\_eq\_ref].

### 11.7.3. Products of Inertia - Positive Integral Sense

Another common representation of the inertia tensor  $\mathcal{I}$  is [4]:

$$\mathcal{I} = \begin{bmatrix} \mathcal{I}_{xx} & -\mathcal{I}_{xy} & -\mathcal{I}_{xz} \\ -\mathcal{I}_{yx} & \mathcal{I}_{yy} & -\mathcal{I}_{yz} \\ -\mathcal{I}_{zx} & -\mathcal{I}_{zy} & \mathcal{I}_{zz} \end{bmatrix}$$

Where,

$$\begin{aligned} \mathcal{I}_{xx} &= \int(y^2 + z^2)dm = \mathcal{J}_{xx} \\ \mathcal{I}_{xy} &= +\int xydm = -\mathcal{J}_{xy} \\ &\text{etc.} \end{aligned}$$

The off-diagonal elements ( $\mathcal{I}_{xy}$ ,  $\mathcal{I}_{xz}$ , etc.) of the inertia tensor are the products of inertia. Since the products of inertia terms do not include the negative sign, they are referred to as being in the "positive integral sense". Note that the products of inertia, given in positive integral sense, must be negated when creating the inertia tensor. The resulting inertia tensor is exactly the same as that for the products of inertia given in negative integral sense.

### 11.7.4. Simulations

Mass properties reference documents provide products of inertia in either positive or negative integral sense. The sense is important when generating the inertia tensor since the products of inertia must be negated when given in the positive integral sense.

Some simulations have a flag that designates the products of inertia integral sense. This allows the simulation to properly handle the products of inertia when creating the inertia tensor.

Other simulations do not utilize an integral sense flag but expect the products of inertia to be input in a specific integral sense. This may require a user to modify the products of inertia prior to entering the data to the simulation in order to match the expected integral sense.

Therefore, it is important for the simulation to designate the expected sense of the input data especially when the product of inertia integral sense is not explicit i.e. no integral sense flag is used, default case, etc.

### 11.7.5. SOASpatialInertia

SOASpatialInertia expects the inertia tensor to be input with the products of inertia in negative integral sense (see setSpatialInertia and setCMSpatialInertia methods).

- **Users must convert products of inertia from positive to negative integral sense prior to input**

SOASpatialInertia outputs the inertia tensor with the products of inertia in negative integral sense (see inertia and cm\_inertia methods).

### 11.7.6. Additional Information

Regardless of the products of inertia integral sense,

- Evaluation of the products of inertia integral may result in a positive or negative value depending on the mass distribution of the body and choice of origin.
- The coordinate system in which the inertia tensor was derived may not align with the coordinate system of the environment in which it is used. For example, the mass property data may be computed in coordinates with the x-axis pointed aft, whereas the simulation coordinate system may define the x-axis as pointed forward. In this case, a frame transformation is required to properly utilize the inertia tensor. [5]

#### 11.7.7. Footnotes

[1] Jain, Abhinandan. [Robot and Multibody Dynamics: Analysis and Algorithms](http://www.amazon.com/Robot-Multibody-Dynamics-Analysis-Algorithms/dp/1441972668) (<http://www.amazon.com/Robot-Multibody-Dynamics-Analysis-Algorithms/dp/1441972668>). Springer, 2010.

[2]  $\ell$  is defined by A. Jain as  $\ell = \begin{bmatrix} a \\ b \\ c \end{bmatrix}$ . For convenience,  $\ell = \begin{bmatrix} x \\ y \\ z \end{bmatrix}$  in this computation.

[3] For example, see [https://en.wikipedia.org/wiki/Moment\\_of\\_inertia](https://en.wikipedia.org/wiki/Moment_of_inertia).

[4] Ginsberg, Jerry H., and Joseph Genin. *Statics ; And, Dynamics: Combined Version* (2nd ed.). New York: Wiley, 1984.

[5] The 'bodyToJointQuat' may be used to align a body with the parent.

## 12. Spice

### 12.1. Background

The `Spice` module provides an interface to JPL's `Spice/NAIF` ephemerides toolkit for obtaining the poses of planetary bodies and spacecraft. The classes in this module are specializations of the `frames` classes in the `DFrame` module.

#### 12.1.1. Bodies and Frames in NAIF

In the NAIF toolkit, NAIF bodies represent objects with a position and velocity and NAIF frames represent orientations. NAIF bodies can represent many types of objects: the solar system barycenter, planets, moons, asteroids, spacecraft, locations on the surface of the Earth... Each NAIF body is identified by a single numeric ID (e.g. 0) and by at least one name (e.g. `SOLAR SYSTEM BARYCENTER`, `SSB`, ...). NAIF bodies **may** be associated with a default frame that defines the orientation for said body. For example, for the planets, these are usually the IAU frames (e.g. for `Earth`, the default frame `IAU_EARTH` is set). However, it is also possible for objects to have no default frame associated with them.

Similarly, NAIF frames can represent different kinds of orientations: inertial frames, rotation of a planet, orientation of a spacecraft bus, of an instrument... NAIF frames are identified by a single numeric ID and a single name. Moreover, NAIF frames are always associated with a single body, which is termed its center.

#### 12.1.2. SpiceFrameContainer

A specialization of the `DFrame::FrameContainer` class to hold body regular and Spice aware frames.

#### 12.1.3. SpiceFrame

The `SpiceFrame` class is a specialization of the `DFrame::Frame` class. Much like other `DFrame::Frame` classes, `SpiceFrame` has the responsibility of holding enough information for a `DFrame::EdgeFrame2Frame` class (in this case `SpiceFrame2Frame`) to obtain the position, velocity, orientation, and angular rate of the frame with respect to another frame (which we can assume will be another `SpiceFrame`).

As discussed in `Bodies and Frames in NAIF`, within the NAIF toolkit, position and velocities are obtained between two NAIF bodies, while orientations and angular rates are obtained between NAIF frames. Thus, for a `SpiceFrame` to provide all the necessary information of a `DFrame::Frame`, it must know the appropriate NAIF ID of a NAIF body **and** of a NAIF frame.

`SpiceFrames` can also be built as "unrotated". This means that they will use the same orientation as the parent frame, i.e. the frame is regarded as being parallel to its parent frame.

#### 12.1.4. SpiceFrame2Frame

This class is a specialization of the `DFrame::EdgeFrame2Frame` class, and can be used to provide the relative pose between the `oframe/pframe` `SpiceFrame` pair. The transform and spatial velocity values are automatically obtained from the `Spice` library using the current epoch value.

#### 12.1.5. Reference & Source material

- [NAIF Spice Webpage](https://naif.jpl.nasa.gov/naif/) (<https://naif.jpl.nasa.gov/naif/>)
- [Spice Doxygen docs](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/Spice/html/) (<https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/Spice/html/>)
- [Team talks](https://dlabportal.jpl.nasa.gov/resources/darts-lab-talks/) (<https://dlabportal.jpl.nasa.gov/resources/darts-lab-talks/>)
- [2019, 2020 course slides](https://dlabportal.jpl.nasa.gov/documentation/coursetutorial-links/) (<https://dlabportal.jpl.nasa.gov/documentation/coursetutorial-links/>)
- [Sphinx docs](https://dartslab.jpl.nasa.gov/dlabdocs/) (<https://dartslab.jpl.nasa.gov/dlabdocs/>)
- [Jupyter notebooks](https://dlabnotebooks.jpl.nasa.gov/hub/login) (<https://dlabnotebooks.jpl.nasa.gov/hub/login>)
- [DARTS Lab QA site](https://dartslab.jpl.nasa.gov/qa/) (<https://dartslab.jpl.nasa.gov/qa/>)

## 12.2. Design

### 12.2.1. Spice Kernel Loading

This section is based on [https://docs.google.com/document/d/1y\\_ZP88FT7SsuM0xAKQr9Kvj\\_AqCfbrPO/edit#](https://docs.google.com/document/d/1y_ZP88FT7SsuM0xAKQr9Kvj_AqCfbrPO/edit#)

We discussed an approach to address the Spice kernel loading issue that Scott Nemeth brought up (in 4/27/20 email) and we discussed the following design:

- Add an additional argument `default_spice_kernels` to:
  - `SpiceFrameContainer::SpiceFrameContainer`
  - `SpiceFrameContainer::createSpiceFrameContainer`
- The `default_spice_kernels` argument would be a list of strings for the names/paths of the Spice kernels to load by default. Note that this could be a single string or a STL vector of strings (we have the capability for that via SWIG).
- The default value for `default_spice_kernels` (in the C++ .h file) would be the current list of paths (or updated ones): `de418/de418.bsp`, `cook_01.tls`, `pck00007.tpc`
- The behavior in `SpiceFrameContainer::SpiceFrameContainer` would change:
  - If `default_kernel_path` is not empty, then the full paths for `default_spice_kernels` would be constructed by combining the path with the name.
  - If `default_kernel_path` is empty AND `default_spice_kernels` is NOT empty, then treat each of the items in `default_spice_kernels` as a full/absolute path and load them.
  - If both `default_kernel_path` and `default_spice_kernels` are empty, not kernels will be loaded; it will be up to the user to load their own kernels.
- Update `SimulationExecutiveNdarts` and `SimulationExecutive` to support the additional `default_spice_kernels` argument.
- Continue to support the `SPICE_KERNELS_PATH` environment variable (for backwards compatibility). If the `default_kernel_path` is empty, then set its value to `SPICE_KERNELS_PATH` if it is defined. if both are empty then assume that the paths for the kernels are valid and use as is.
- Do not change the current kernel loading behavior in `TargetSpiceAssembly`. We need it for backwards compatibility and since it makes sense to have the kernels for a specific planetary body defined close to the rest of the definitions for that body.

- Finally, we should regularly update our current list of default kernels and get the latest versions of the leap-seconds file, etc. This should be on the checklist of items to be done for major releases. NOTE: This should be done and regression tests should be updated before doing the updates above.

The goals for this design are to:

- Allow existing simulations run scripts to work without changes
- Support the option of loading ALL spice kernels when the `SimulationExecutive` is created (in this case you would not specify any kernels in the parameters for `TargetSpiceAssembly` objects).
- Support loading a user-specified set of default kernels when the simulation is created and then load other kernels later as desired (eg. via `TargetSpiceAssembly`).

## 12.3. Usage

### 12.3.1. Using SpiceFrames in your simulation

#### 12.3.1.1. SpiceFrameContainer

When a simulation needs to use Spice, it will usually use the `use_spice=True` option when creating the simulation executive:

```
sim = SimulationExecutiveNdarts(..., use_spice=True)
```

This will automatically set up the simulation's frame container as a `SpiceFrameContainer`, which is capable of generating and handling `SpiceFrame`'s.

#### 12.3.1.2. Creating new SpiceFrames

As discussed in the Background section, a `SpiceFrame` must ultimately always refer to a NAIF body (for its position and velocity) and to a NAIF frame (for its orientation and velocity). We have three ways of providing this information:

##### 12.3.1.2.1. Providing the NAIF body and the NAIF frame

This is the most explicit method. It allows you the freedom to create any combination of NAIF bodies and NAIF frames. For example, you may create an Earth-centered frame that uses the inertial J2000 reference frame. For example:

```
earth = frame_container.getSpiceFrame(399, "IAU_Earth") # body, frame
```

##### 12.3.1.2.2. Providing the NAIF body

The frame used will be the default frame associated with the body. For example, for planets this is usually the IAU reference frame (e.g. for `Earth`, the `IAU_EARTH` reference frame). However, **not all bodies have a default reference frame**. Sometimes, kernel writers decide not to set a default frame so that users have to be explicit in what frame they want to use. This is especially true for spacecraft bodies. For example:

```
earth = frame_container.getSpiceFrame(399) # body
```

##### 12.3.1.2.3. Providing the NAIF frame

The body used will be the center of the provided frame. All NAIF frames have exactly one center associated with them. For example:

```
earth = frame_container.getSpiceFrameFromNaifFrame("IAU_Earth") # frame
```

##### 12.3.1.2.4. Numeric IDs or names?

You may always use either numeric IDs or names interchangeably. Moreover, names are case insensitive. The following two snippets are exactly equivalent:

```
sun = frame_container.getSpiceFrame(10)
mars = frame_container.getSpiceFrame(499, "iau_MaRs")
earth = frame_container.getSpiceFrameFromNaifFrame("IAU_Earth")
```

```
sun = frame_container.getSpiceFrame("SUN")
mars = frame_container.getSpiceFrame("mars", 10014)
earth = frame_container.getSpiceFrameFromNaifFrame(10013)
```

#### 12.3.1.3. Creating unrotated SpiceFrames

As mentioned in the Background section, sometimes it is useful to define `SpiceFrame` s that are oriented parallel to their parent frames. A `SpiceFrame` of this nature can be obtained with:

```
earth = frame_container.getSpiceFrameUsingParentOrientation("Earth")
```

#### 12.3.1.4. Creating Planet-Centered Inertial (PCI) frames

Sometimes, we want to create inertial frames from existing, rotating `SpiceFrame` . This can be done through `getPCIFrame` :

```
earth = frame_container.getSpiceFrameFromNaifFrame("IAU_EARTH")
earth_PCI = earth.getPCIFrame()
```

The `Frame earth_PCI` will be inertial (has a fixed rotation with respect to the inertial root frame), centered at the Earth, and its rotation is such that it corresponds to the rotation of `earth` at the epoch when `getPCIFrame` was called.

It is also possible to specify an epoch, instead of using the current epoch of the `SpiceFrameContainer` :

```
frame_container.setEpoch(1000)
earth_PCI_2 = earth.getPCIframe(2000)
```

`earth_PCI_2` 's orientation will coincide with `earth` 's rotation at epoch 2000 **not** at 1000 .

Finally, it is also possible to create a PCI frame that corresponds to a different NAIF frame to that of the PCR frame. For example:

```
earth_PCI_3 = earth.getPCIframe(None, "ITRF93")
earth_ITRF93 = frame_container.getSpiceFrameFromNaifFrame("ITRF93")
```

In this case, `earth_PCI_3` will **not** have the same orientation as `earth` at the current epoch. Instead, it will have the same orientation as `earth_ITRF93` (which is created here only for illustrative purposes). Note that `None` can be replaced by a specific epoch in `getPCIframe(None, "ITRF93")` .

### 12.3.1.5. Connecting SpiceFrames

Once you have two `SpiceFrame` s, you may connect them through a `SpiceFrame2Frame` . In the example below, we are connecting the root inertial frame (J2000) to the Earth, which is then connected to the moon:

```
j2000 = frame_container.rootFrame()
earth = frame_container.getSpiceFrame("EARTH")
moon = frame_container.getSpiceFrame("MOON")

j2earth = SpiceFrame_Py.SpiceFrame2Frame(j2000, earth, True)
earth2moon = SpiceFrame_Py.SpiceFrame2Frame(earth, moon, True)
```

### 12.3.1.6. Notes

- The appropriate `Spice` kernels must be loaded that support the `Spice` IDs before trying to create `SpiceFrame` s.
- While the `Solar System Barycenter` NAIF body (ID 0) doesn't have a default frame in SPICE, we always use the `J2000` (aka `EME2000`) frame by default for this body.
- Spice reserves ids 1400000 through 4000000 for **user defined frames** (see page 44 in [tutorial](#) (<http://www.astronomia.edu.uy/cospar07/material/5%20-%20Jose%20Luis%20Vazquez%20-%20Spice/tutorial/spice.pdf>)). These frames are usually defined in `FK` kernels such as in `/misc/ofDate.fk` which must be loaded during initialization.

### 12.3.2. Spice Utils

The `SpiceFrame_Py` module has a series of utility functions that might help you operate with `Spice` bodies and frames:

- **`SpiceFrame_Py.getBodyId`**: Returns the numeric ID associated with a body name in SPICE.
- **`SpiceFrame_Py.getBodyName`**: Returns a name associated with the numeric ID of a body in SPICE. Note that bodies can have more than one name. This function will always return the last name specified for a numeric ID.
- **`SpiceFrame_Py.getFrameId`**: Returns the numeric ID associated with a frame name in SPICE.
- **`SpiceFrame_Py.getFrameName`**: Returns the name associated with the numeric ID of a frame in SPICE.
- **`SpiceFrame_Py.getStandardBodyName`**: Returns always the same name given any of the valid names of a body in SPICE. For example, `getStandardBodyName("SSB")` , `getStandardBodyName(" SSB ")` , `getStandardBodyName("SOLAR_SYSTEM_BARYCENTER")` , and `getStandardBodyName("Solar System Barycenter")` will all return `"SOLAR_SYSTEM_BARYCENTER"` .
- **`SpiceFrame_Py.bodyvcd`**: Get the body parameter value specified. Note: Only is valid for single-valued parameters (eg, mass, GM, R, etc)
- **`SpiceFrame_Py.getDefaultFrameIdOfBody`**: Returns the ID of the default NAIF frame declared for a NAIF body. Not all bodies have a default frame. If this function is called for a body without a default frame, an error is raised.
- **`SpiceFrame_Py.getCenterIdOfFrame`**: Returns the ID of the NAIF body used as the center of a NAIF frame.
- **`SpiceFrame_Py.getPossibleFrameIdsOfBody`**: Returns a vector containing the numeric IDs of all NAIF frames that have the input NAIF body set as their center. This function is useful to discover what frames are available for a body.
- **`SpiceFrame_Py.getPositionVelocity`**: Uses Spice to compute the position and velocity of one NAIF body with respect to another NAIF body at a specified epoch in the given NAIF reference frame.
- **`SpiceFrame_Py.getRotationAngularRate`**: Uses Spice to compute the orientation and angular velocity of one NAIF frame with respect to another NAIF frame at a specified epoch.

### 12.3.3. spkcheck

The command-line tool `spkcheck` has been designed to assist engineers in visualizing SPICE SPK files and understanding connections between objects defined by these files.

With this tool, one can answer the following questions: \* Can the state of object A with respect to B be retrieved at time T? \* What is the path of state transformations that SPICE is following between objects A and B at time T? \* What SPK file(s) are enabling retrieving these states?

#### 12.3.3.1. Quick usage guide

The following command will display all distinct time periods in a kernel file `FILE` and whether the state of `target` can be retrieved w.r.t `center` . Note that more than one `FILE` can be specified, and each of these can be either an SPK file or a meta-kernel with `SPK` files. Information on the individual transformations between the two bodies and the file(s) that enable each transformation is also displayed.

```
srun spkcheck.py [FILE...] --target [target] --center [center]
```

Alternatively, all connections in `FILE` can be displayed with:

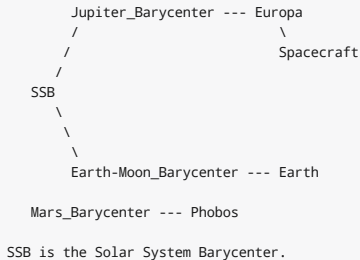
```
srunch spkcheck.py [FILE...] --file-graphs
```

A more detailed explanation is provided below.

### 12.3.3.2. Brief introduction to SPK

SPICE's [tutorial slides](https://naif.jpl.nasa.gov/pub/naif/toolkit_docs/Tutorials/pdf/individual_docs/18_spk.pdf) (https://naif.jpl.nasa.gov/pub/naif/toolkit\_docs/Tutorials/pdf/individual\_docs/18\_spk.pdf) on SPK files are highly recommended. Alternatively, the SPK [Required Reading](https://naif.jpl.nasa.gov/pub/naif/toolkit_docs/C/req/spk.html) (https://naif.jpl.nasa.gov/pub/naif/toolkit\_docs/C/req/spk.html) contains a more detailed explanation of the inner working of the SPK system. This section provides an extremely brief introduction.

Each SPK file contains the state of one or more objects for specific time periods. Each of these states is written with respect to another ephemeris object. The following graph illustrates these connections (graph modified from [SPICE](https://naif.jpl.nasa.gov/pub/naif/toolkit_docs/C/req/spk.html#Loading%20Files) (https://naif.jpl.nasa.gov/pub/naif/toolkit\_docs/C/req/spk.html#Loading%20Files)):



The previous graph can be interpreted as three one-directional paths: \* Spacecraft is given w.r.t Europa, which is given w.r.t the Jupiter Barycenter, which is given w.r.t the Solar System Barycenter. \* Earth is given w.r.t the Earth-Moon barycenter, which is given w.r.t the Solar System Barycenter. \* Phobos is given w.r.t the Mars Barycenter.

If one wants to retrieve the state of Spacecraft w.r.t the Jupiter Barycenter, SPK will follow the first path backward, chaining through Europa and performing all relevant state transformations. Alternatively, if one wants the state of Spacecraft w.r.t the Earth, SPK will iterate the first path backward until it reaches the Solar System Barycenter, at which point it will iterate forward the second path until reaching the Earth. Finally, if one wants to retrieve the state of Spacecraft w.r.t Phobos, SPICE will throw an error: This is because the first and third paths are completely disconnected.

`spkcheck` parses SPK files to reconstruct the state transformation chains that SPICE would use (i.e. the graph shown above). From this data, it is capable of finding whether two bodies are connected at a specific moment.

### 12.3.3.3. Example use case

The script is available under:

```
/src/Spice/python/spkcheck.py
```

One wants to retrieve the state of Neptune's moon Proteus (NAIF ID: 808) with respect to the Earth (NAIF ID: 399) in 1850 and 2022. To do this, we have a set of kernels and meta-kernels from the previous project we were working on:

```
kpool0.tm
/home/simscape/SimScape/NEO/Benu/benu_refdrmc_v1.bsp
kpool1.tm
```

where the contents of `kpool0.tm` and `kpool1.tm` are:

```
KPL/MK

File name: kpool0.tm

The SPICE kernels that allow us to get information
on Neptune's moons.

\begindata

PATH_VALUES = ('/home/dlab/pkgsrc/spice/KERNELS')

PATH_SYMBOLS = ('A')

KERNELS_TO_LOAD = ('$A/misc/de440s.bsp',
                  '$A/misc/nep095/nep095.bsp')
```

```

KPL/MK

File name: kpool1.tm

The SPICE kernels that allow us to get information
on Benu.

\begindata

PATH_VALUES = ('/home/dlab/pkgsrc/spice/KERNELS')

PATH_SYMBOLS = ('B')

KERNELS_TO_LOAD = ('$B/misc/de440s.bsp',
'$B/misc/sb-101955-118.bsp',
'$B/core/cook_01.tls')

```

These files are available under:

```
/src/Spice/test/test_spkcheck
```

### 12.3.3.3.1. Finding the priority of the files

We know that the order in which we load the kernels is important, so we would like to get an idea of the priority that each kernel file would have if we were to load each kernel to SPICE in the following order

```

kpool0.tm
/home/simscape/SimScape/NEO/Benu/benu_refdrmc_v1.bsp
kpool1.tm

```

To do so, we can run the command

```
srunk spkcheck.py kpool0.tm /home/simscape/SimScape/NEO/Benu/benu_refdrmc_v1.bsp kpool1.tm
```

while will print:

```

Loaded the following kernels:
kpool0.tm      : $A/misc/de440s.bsp           | Less priority
kpool0.tm      : $A/misc/nep095/nep095.bsp          |
-              : ...Scape/NEO/Benu/benu_refdrmc_v1.bsp |
kpool1.tm      : $B/misc/de440s.bsp           |
kpool1.tm      : $B/misc/sb-101955-118.bsp        ▼ More priority

```

As we can see, the priority with which SPK will use each file corresponds to the order in which they were input in the command and with the order in which the kernels were specified within each meta-kernel.

Now that we have a breakdown of all the possible files, we realize that we are not interested in those files that give the state of the Benu asteroid. Therefore, we do not need to load `benu_refdrmc_v1.bsp` or `sb-101955-118.bsp`. Moreover, we see that `de440s.bsp` is being loaded twice, which is unnecessary. Thus, for the rest of the examples, we will focus only on `kpool0.tm`.

### 12.3.3.3.2. Determining valid intervals between target and center of motion

We can check whether file `kpool0.tm` provides sufficient data to compute the state of Proteus w.r.t the Earth with the command:

```
srunk spkcheck.py kpool0.tm --target proteus --center earth
```

which will output data for three different time intervals. The first one, which covers from 1849 to 1900, is of interest since we want to compute the states in 1850:

```

1 =====
2 1849 DEC 26 00:00:00.000 - 1900 JAN 01 00:00:41.183 ***NOT VALID***
3 =====
4 Target Body      w.r.t Center of Motion      C      From file...
5 -----
6 Position of PROTEUS is not defined
7 -----
8 EARTH            w.r.t EARTH BARYCENTER      $A/misc/de440s.bsp
9 EARTH BARYCENTER w.r.t SOLAR SYSTEM BARYCENTER $A/misc/de440s.bsp
A =====

```

Note that the line indexes have been added for clarity. Line 2 shows the start and end dates of the interval in Gregorian Calendar format. Moreover, the flag `NOT VALID` shows that the state of Proteus **cannot** be retrieved w.r.t. the Earth for this interval. Details on why this is are contained in the following lines.

Line 6 reveals that, for this time period, the state of Proteus is not specified w.r.t any other body. The chain that defines the state of the Earth, on the other hand, is given by lines 8-9. As we can see, the file `de440s.bsp` defines the state of the Earth w.r.t the Earth Barycenter, and the Earth Barycenter w.r.t the Solar System Barycenter. This means that, for example, the state of the Earth w.r.t the Solar System Barycenter could be retrieved.

Maybe we are luckier for the second year of interest: 2022. In this case, the second interval is of interest, since it covers from 1900 to 2050:



```

1 =====
2 1900 JAN 01 00:00:41.183 - 2050 JAN 01 00:01:09.183  VALID
3 =====
4 Target Body      w.r.t Center of Motion      C      From file...
5 -----
6 PROTEUS          w.r.t NEPTUNE BARYCENTER      ...c/nep095/nep095.bsp
7 NEPTUNE BARYCENTER w.r.t SOLAR SYSTEM BARYCENTER * ...c/nep095/nep095.bsp
8 -----
9 EARTH            w.r.t EARTH BARYCENTER        ...c/nep095/nep095.bsp
A EARTH BARYCENTER w.r.t SOLAR SYSTEM BARYCENTER * ...c/nep095/nep095.bsp
B =====

```

In this case, line 2 shows the flag `VALID`, which means that the state of Proteus **can** be retrieved w.r.t. the Earth for this interval.

Lines 6-7 reveal that Proteus is defined for this period. In particular, the chain of definitions is Proteus ← Neptune Barycenter ← Solar System Barycenter. Even though the Earth is not contained in this chain, a common node between the two chains is present (the Solar System Barycenter), and therefore the state can be retrieved. The marker `*` in the column labeled `C` reveals the lines where the `C`ommon node appears.

Interestingly, we see that `nep095.bsp` is providing both Earth and Proteus state w.r.t the Solar System Barycenter. In the previous interval, however, we saw that the state of the Earth was given by `de440s.bsp`. This change happened because the file `de440s.bsp` was specified in the meta-kernel before `nep095.bsp`, which means that the latter's data should replace the former's. In this case, if only Proteus' state is of interest, it would be advantageous to **not** load `de440s.bsp`, as it provides no required information.

The previous commands output tables for the three unique periods defined in the SPK kernels. However, we were only interested in two intervals: from 1849 to 1900 and from 1900 to 2050. We could have used the option `--time` to avoid printing data for the third interval:

```

srun spkcheck.py kpool0.tm --target proteus --center earth --time 1875_JAN_01 --time 2000_JAN_01

```

Here, we are specifying two "instants" of interest (January 01, 1875 and January 01, 2000), so that `spkcheck` only prints information pertaining to those intervals that contain these dates.

### 12.3.3.3.3. Visualizing the connections

If one is interested in getting a more complete picture of a (set of) kernels, the `--graph` option can be used to visualize the tree of connections that SPICE would see if it were to load the file(s).

Using the file `kpool0.tm` as defined above, the following command:

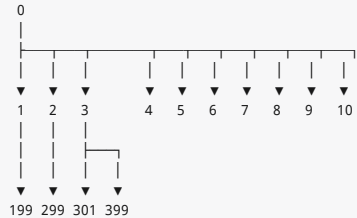
```

srun spkcheck.py kpool0.tm --graphs

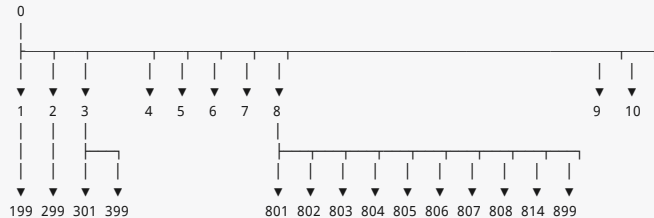
```

produces the following output:

```
[1849 DEC 26 00:00:00.000 - 1900 JAN 01 00:00:41.183]
```



```
[1900 JAN 01 00:00:41.183 - 2050 JAN 01 00:01:09.183]
```



```
[2050 JAN 01 00:01:09.183 - 2150 JAN 22 00:00:00.000]
```

```
...
```

Graph Body Legend

|                             |                        |
|-----------------------------|------------------------|
| 0 : SOLAR SYSTEM BARYCENTER | 1 : MERCURY BARYCENTER |
| 2 : VENUS BARYCENTER        | 3 : EARTH BARYCENTER   |
| 4 : MARS BARYCENTER         | 5 : JUPITER BARYCENTER |
| 6 : SATURN BARYCENTER       | 7 : URANUS BARYCENTER  |
| 8 : NEPTUNE BARYCENTER      | 9 : PLUTO BARYCENTER   |
| 10 : SUN                    | 899 : NEPTUNE          |
| 399 : EARTH                 | 801 : TRITON           |
| 802 : NEREID                | 803 : NAIAD            |
| 804 : THALASSA              | 805 : DESPINA          |
| 806 : GALATEA               | 807 : LARISSA          |
| 808 : PROTEUS               | 299 : VENUS            |
| 301 : MOON                  | 814 : 814              |
| 199 : MERCURY               |                        |

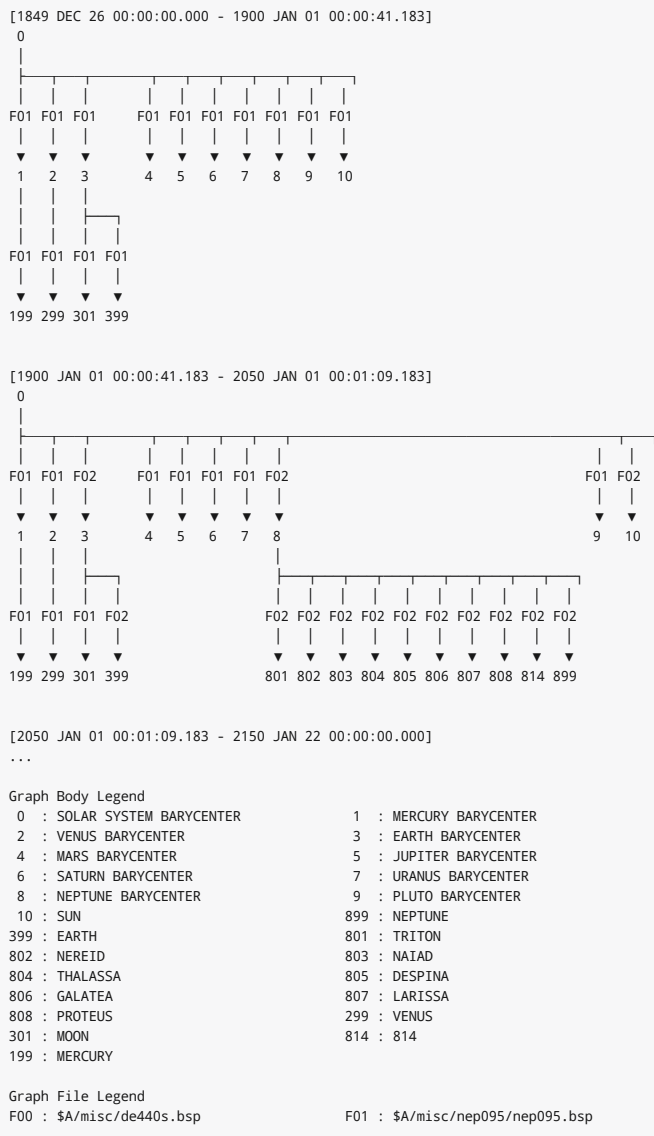
These graphs represent the connections available in each interval. For the first graph, for example, we can see how 0 flows to 3 and then to 399. Making use of the Graph Body Legend, we see that this translates to SOLAR SYSTEM BARYCENTER → EARTH BARYCENTER → EARTH. Therefore, the state of the Earth w.r.t the Solar System Barycenter could be obtained, just as the interval tables showed. Note that Proteus (NAIF ID: 808) does not appear in the graph, thus revealing that no data is available for the body.

For the second graph, we see how Neptune's Barycenter (8) is now connected to its moons, including Proteus (808). We can also see that the Earth and Proteus are connected in this graph, which means that the state of one can be obtained with respect to the other. Again, this coincides with the information shown in the tables.

Finally, a more complete graph can be obtained using the `--file-graph`. This will add the source file where the relation was obtained to each arrow in the graph:

```
srun spkcheck.py kpoo10.tm --file-graphs
```

which produces:



As we can see, each arrow is now labeled with either F00 or F01. These labels are the alias for the two files de440s.bsp and nep095.bsp, as shown in the Graph File Legend.

The first graph reveals that all relations are given by de440s.bsp (F00), thus loading nep095.bsp would be useless if this were the only interval of interest. The second graph shows how nep095.bsp (F01) defines new connections. Not only that, but it replaces definitions previously given by de440s.bsp (F00).

Note that, just like in the case of the connection tables, one can use the --time option to limit for which intervals these graphs should be printed.

#### 12.3.3.3.4. Exporting graphs to .dot

If you are using a graphical interface, it may be interesting to export the previous plots to the .dot format. These can then be rendered using Graphviz drawing software. Each graph will be exported as a unique file in a folder of your choosing:

```
srln spkcheck.py kpoo10.tm --graphs-output kpoo1_0_graphs
```

The above command will create the folder kpoo1\_0\_graphs, if it does not exist, and will populate it with three files:

```
1849_DEC_25_23:59:18.816_-_1900_JAN_01_00:00:00.000.dot
2050_JAN_01_00:00:00.000_-_2150_JAN_21_23:58:50.816.dot
1900_JAN_01_00:00:00.000_-_2050_JAN_01_00:00:00.000.dot
```

corresponding to the three unique intervals defined by kpoo10.tm. One can use the --time option to limit for which intervals these files should be generated.

Additionally, if the --target and --center options are set, the the graph will be colored to illustrate the path between target and center in the graph:

```
srln spkcheck.py kpoo10.tm --graphs-output kpoo1_0_graphs --target proteus --center earth
```

By default, the .dot and corresponding .pdf files will be generated. You can turn off rendering to .pdf by using the --graphs-output-no-render flag.

The following is an example of a graph generated through this method (does not correspond to the previous examples): dot\_graph\_example

#### 12.3.3.4. Notes

- This script has not been extensively tested, use with care. *Hic sunt dracones*
- It is possible that this script shows that a connection is valid, yet SPICE is unable to retrieve it. Barring bugs in the script, this is because there are missing frame definitions:
  - If the desired frame for the state is 'J2000', then the issue is likely that some of the ephemeris specified by the SPK files are given in a frame other than 'J2000' for which no data is available. This error is uncommon for ss bodies, as their ephemeris is usually expressed in 'J2000'.
  - If the desired frame for the state is other than 'J2000', then likely the desired frame is not correctly defined.
- An extension to this tool that can detect frame-related failures might be interesting.

#### 12.4. Software

#### 12.5. Raw documents

## 13. DMesh

### 13.1. Background

#### 13.1.1. Reference & Source material

- [DMesh Doxygen docs](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/DMesh/html/) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/DMesh/html/)
- [Team talks](https://dlabportal.jpl.nasa.gov/resources/darts-lab-talks/) (https://dlabportal.jpl.nasa.gov/resources/darts-lab-talks/)
- [2019, 2020 course slides](https://dlabportal.jpl.nasa.gov/documentation/coursetutorial-links/) (https://dlabportal.jpl.nasa.gov/documentation/coursetutorial-links/)
- [Sphinx docs](https://dartslab.jpl.nasa.gov/dlabdocs/) (https://dartslab.jpl.nasa.gov/dlabdocs/)
- [Jupyter notebooks](https://dlabnotebooks.jpl.nasa.gov/hub/login) (https://dlabnotebooks.jpl.nasa.gov/hub/login)
- [DARTS Lab QA site](https://dartslab.jpl.nasa.gov/qa/) (https://dartslab.jpl.nasa.gov/qa/)

### 13.2. Design

#### 13.2.1. Mesh class design



**TBD:** This section is mostly unfinished and should include information on the design of DMesh as a module, and the design of its various classes and functions.

##### 13.2.1.1. Perturb mesh

The `Mesh` class has a `perturbMesh` method that can be used to perturb the vertices of the mesh. For the C++ interface, this method takes in a `std::function<Eigen::Vector3d(Eigen::Vector3d)>`, while in the Python interface the method takes in a Python function of the form `Callable`. This interface allows users to easily modify the vertices of a mesh, regardless of whether they are in C++ or Python.

Internally, the Python interface version is implemented by wrapping the Python function within a C++ function of the form `std::function<Eigen::Vector3d(Eigen::Vector3d)>` and calling the C++ version of `perturbMesh`.

##### 13.2.1.2. Embree



**TBD:** This section is mostly unfinished and should include information on the design of DMesh as a module, and the design of its various classes and functions.

###### 13.2.1.2.1. Computing closest object to a target point

All of the `computeClosestX` functions (where `X` is a point, face, or vertex) utilize the Embree point query function. This function traverses the BVH and for each primitive (mesh face) that intersects the query domain and calls a user-defined callback. The user-defined callback can perform operations using the primitive, and can modify the size of the query domain.

In general, the `computeClosestX` callbacks follow the following flowchart:

```
PLANTUML
/'
  Define a macro that stores the PlantUML theme.

  To use, in your asciidoc wiki page write:

  include::_auto_global/plantUmlTheme.asciidoc[]

  For example,
  ```plantuml
  include::_auto_global/plantUmlTheme.asciidoc[]
  @startuml
  ...
  @enduml
  ```
  '/

!theme spacelab
skinparam ArrowFontColor #446e9b
@startuml
start
:Receive face from Embree;
:Compute new distance for X for this face;
if (Distance is closer than current closest distance?) then (yes)
  :Store X and associated distance.
  Update Embree callback radius.
  Return updated_radius = true;
else (no)
  :Return updated_radius = false;
endif
end
```

###### 13.2.1.2.1.1. Closest vertex query

The closest vertex query calculates the distance from the user-defined target point to the vertices associated with the face. If the minimum of these distances is lower than the current minimum distance, then the minimum distance is updated, the vertex index is saved, and the Embree callback radius is updated.

###### 13.2.1.2.1.2. Closest point query

The closest point query callback is a little more complex than the vertex callback, since we need to find the closest vertex on a face rather than checking a finite number of points. Therefore, an algorithm is used to find the closest point on a triangle (a face of the mesh) to the user's target point. This algorithm is described [here](#) (https://www.geometrictools.com/Documentation/DistancePoint3Triangle3.pdf) and an implementation of it can be found in the `DistPointTriangle.h` header file of [GTENGINE 6.4](#) (https://www.geometrictools.com/Downloads/Downloads.html).

If the distance of this closest point to the target point is lower than the current minimum distance, then the point location and face ID are saved and the Embree callback radius is updated.

#### 13.2.1.2.1.3. Closest face query

The closest face query uses the same callback as the closest point query: see the closest point query section for details. Since the closest point and closest face do the same computation, there is also a `computeClosestPointAndFace` function that does the computation once and returns the value of both the closest point and the closest face.

Note, the closest face can not be found reliably using the closest vertex callback for the reasons described [here](#)

(<https://darts.gitlab-internal.jpl.nasa.gov/darts-lab/jpl-internal/all-access/development/scene-geometry/dmesh-/issues/55>).

### 13.2.2. Mesh Utilities



**TBD:** This section is mostly unfinished and should include information on the design of DMesh as a module, and the design of its various classes and functions.

#### 13.2.2.1. Construct mesh from function

The mesh utilities includes `constructMeshFromFunction` method that can be used to construct a mesh given a user-defined function. This is done as follows: 1. Create a planar mesh. 2. Perturb that planar mesh using the user-defined function. 3. Return the new mesh

In C++ the user-defined function should be of type `std::function<Eigen::Vector3d(Eigen::Vector3d)>`, while in Python it should be of type `Callable[[float, float, float], Tuple[float, float, float]]`.

#### 13.2.2.2. Simplex noise

The `applySimplexNoise` function can be used to apply simplex noise to a given mesh. The function perturbs the vertices of the mesh using a simplex noise function. The simplex noise itself is controlled user defined parameters to the function: see the function doc string for more details.

#### 13.2.3. DMeshObject

`DMeshObject` is designed to store a `DMesh` and all of its properties, including:

- the faces and vertices, i.e., the geometry of the `DMesh`
- all of the `DMesh` attributes
- the `DMesh` material
- the convex hull file for the `DMesh` if applicable

This data is stored using the following structure

```

/'
  Define a macro that stores the PlantUML theme.

  To use, in your asciidoc wiki page write:

  include::_auto_global/plantUmlTheme.asciidoc[]

  For example,
  ```plantuml
  include::_auto_global/plantUmlTheme.asciidoc[]
  @startuml
  ...
  @enduml
  ```
  */

!theme spacelab
skinparam ArrowFontColor #446e9b
@startuml
map DMeshObject {
  faces => DMesh faces data.
  vertices => DMesh vertices data.
}

map DMeshMaterialObject {
  Double data => All double attributes of the material, e.g., OPACITY defines the opacity of the material.
  String data => All string attributes of the material, e.g., DIFFUSE_MAP defines the map for the diffuse color.
  Color data => All color attributes of the material, e.g., AMBIENT_COLOR defines the ambient color of the material. These are stored as a vector of 4 doubles.
  Integer data => All integer attributes of the material.
}

map DMeshAttributeObject {
  Vector => Stores the actual data of the attribute as a vector or matrix, depending on the data type.
  Stride => Stride of the attribute data.
  Size => Size of the attribute data.
  Interpolation => Defines the interpolation scheme for the attribute.
  Invalid data value => Defines the invalid data value for the attribute.
  Mode => Attribute insert mode.
}

map DMeshMaterialTextureObject {
  Blend => The blend factor for the texture.
  Map mode => Defines how UV coordinates outside the [0..1] range are handled
  UV index => UV channel; index into the parent Mesh object's mTextureCoords[].
  Mapping => Defines how the mapping coords for a texture are generated
  Type => Map type, e.g., Material::DIFFUSE_MAP
  OP => Defines how the Nth texture is combined with previous layer.
}

map FileRefs {
  Texture files => Contain texture data.
  Convex hull files => Contain convex hull data.
}

DMeshObject --> DMeshMaterialObject
DMeshObject --> DMeshAttributeObject
DMeshMaterialObject --> DMeshMaterialTextureObject

DMeshObject ..> FileRefs
DMeshMaterialObject ..> FileRefs
DMeshAttributeObject ..> FileRefs
DMeshMaterialTextureObject ..> FileRefs

@enduml

```

where a solid arrow indicates a parent/child relationship, and a dashed arrow indicates a potential parent/child relationship. The dashed arrows are used because more than one object can point to the same `FileRef`. In that case, one of the objects pointing to the `FileRef` is a parent, and the other objects pointing to the `FileRef` merely retain object references to the file. Note, there can be, and oftentimes are, more than one `DMeshAttributeObject` and more than one `FileRef`. However, there will always only be one `DMeshObject` and one `DMeshMaterial` object per `DMesh` store.

## 13.3. Usage

### 13.3.1. DMeshObject

A `DMeshObject` can be constructed using an `id` (a string), another `DMeshObject`, or `StoreData`. If constructed using another `DMeshObject` or `StoreData`, the associated `DMesh` data will already be set from the incoming `DMeshObject` or `StoreData`. Otherwise, the `DMesh` data will be filled using a dummy mesh with no vertices or faces.

To modify the `DMesh` being used by the `DMeshObject` use the `setMesh` method. This will automatically parse the `DMesh` and extract all of the mesh geometry, material, and mesh attribute data. For example, suppose we had a `cube.obj` file that defines a mesh. Then, we could create a `DMeshObject` for it by using:

```

from DScene import DMesh_Py as DMesh

# Get mesh
importer = DMesh.AssimpImporter()
meshList = importer.importMesh("../test_import/cube.obj", "obj")
mesh = meshList[0]

# Create DMeshObject
mo = DMesh.DMeshObject("mesh")
mo.setMesh(mesh)

```

The above will create a `DMeshObject` called `mo` that holds the `DMesh` data from the `cube.obj` file.

In addition, the `setConvexHulls` method can be used to add an associated `convexhulls` file to the `DMeshObject`.

To retrieve the `DMesh`, use the `getMesh` method. To retrieve the `convexhulls` file, use the `getConvexHullsFile` method.

For more examples, see `DMesh/test/test_dmesh_object`

[13.4. Software](#)

[13.5. Raw documents](#)



## 14. CORE

### 14.1. Background

#### 14.1.1. Reference & Source material

- [CORE Doxygen docs](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/CORE/html/) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/CORE/html/)
- [Team talks](https://dlabportal.jpl.nasa.gov/resources/darts-lab-talks/) (https://dlabportal.jpl.nasa.gov/resources/darts-lab-talks/)
- [2019, 2020 course slides](https://dlabportal.jpl.nasa.gov/documentation/coursetutorial-links/) (https://dlabportal.jpl.nasa.gov/documentation/coursetutorial-links/)
- [Sphinx docs](https://dartslab.jpl.nasa.gov/dlabdocs/) (https://dartslab.jpl.nasa.gov/dlabdocs/)
- [Jupyter notebooks](https://dlabnotebooks.jpl.nasa.gov/hub/login) (https://dlabnotebooks.jpl.nasa.gov/hub/login)
- [DARTS Lab QA site](https://dartslab.jpl.nasa.gov/qa/) (https://dartslab.jpl.nasa.gov/qa/)

### 14.2. Design

#### 14.2.1. CORE objects

##### 14.2.1.1. Object versions

(from Abhi's notes)

- Found a bug in several of the `getStoreData()` method implementations within the SimScape classes. The `_version` member was being used to get the version for that class and that would get stored for the class. This is normally OK, but a problem when the object is actually from a derived class. Thus say the object is a `TopoMesh` (which is at version 10). Thus its `_version` field will have value 10. It is derived from the `TopoCloud` class (which lets say is at version 6). When storing the object, we want to store a the `TopoMesh_Version` tag with value 10, and the `TopoCloud_version` tag with value 6. However if the `getStoreData()` method uses the `_version` field, both tags will be stored with the value 10. When loading back the object, we will have the strange outcome of the `TopoCloud`'s constructor (at version 6) being asked to deal with a `TopoCloud` stored data from version 10 (which does not exist)!
- The `/home/simscape/SimScape/LSOS/LunarSynthesis/Stores/DataSet1_v1:DS1TiledDem` object is one with the `TiledData2D` version number of 10 stored incorrectly as the `Data2DBase` class version.

### 14.3. Usage

### 14.4. Software

### 14.5. Raw documents

# 15. SimScapeBasic

## 15.1. Background

### 15.1.1. Reference & Source material

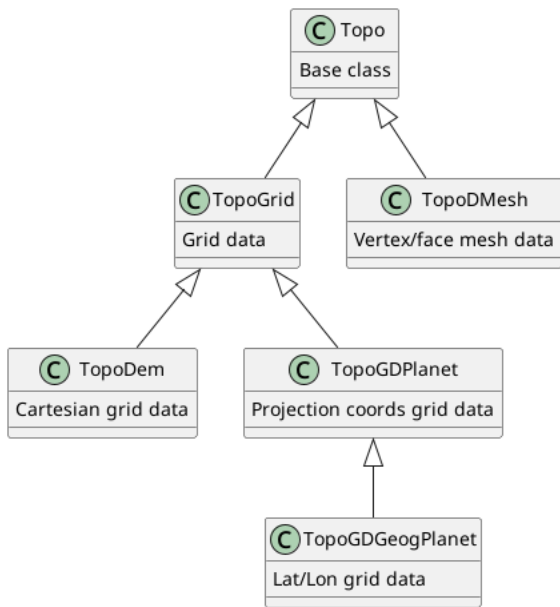
- [SimScapeBasic Doxygen docs](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SimScapeBasic/html/) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SimScapeBasic/html/)
- [Team talks](https://dlabportal.jpl.nasa.gov/resources/darts-lab-talks/) (https://dlabportal.jpl.nasa.gov/resources/darts-lab-talks/)
- [2019, 2020 course slides](https://dlabportal.jpl.nasa.gov/documentation/coursetutorial-links/) (https://dlabportal.jpl.nasa.gov/documentation/coursetutorial-links/)
- [Sphinx docs](https://dartslab.jpl.nasa.gov/dlabdocs/) (https://dartslab.jpl.nasa.gov/dlabdocs/)
- [Jupyter notebooks](https://dlabnotebooks.jpl.nasa.gov/hub/login/) (https://dlabnotebooks.jpl.nasa.gov/hub/login/)
- [DARTS Lab QA site](https://dartslab.jpl.nasa.gov/qa/) (https://dartslab.jpl.nasa.gov/qa/)

## 15.2. Design

## 15.3. Usage

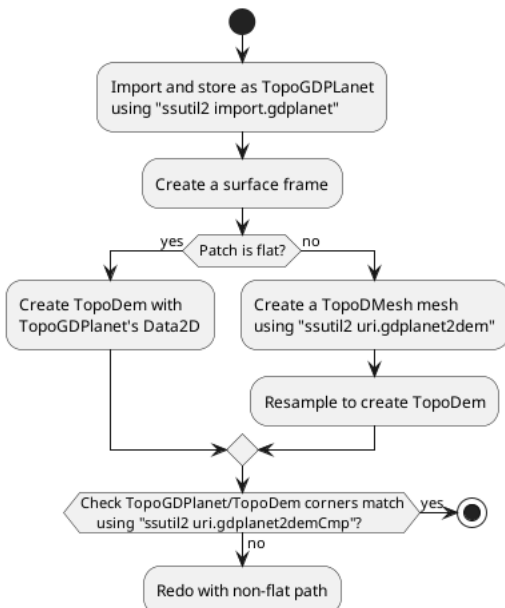
### 15.3.1. Importing GeoTiff files

The following describes the key part of the Topo class hierarchy



The following describes the process for converting a GeoTiff file into a TopoDem in a SimScape store. Also see the [recipe](#)

(https://dartsgitlab-internal.jpl.nasa.gov/darts-lab/jpl-internal/all-access/team/infrastructure/recipes/-wikis/docFiles/recipe\_ImportingTerrains) for examples of the specific commands to use.



15.4. Software

15.5. Raw documents

## 16. DScene

### 16.1. Background

#### 16.1.1. Reference & Source material

- [DScene Doxygen docs](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/DScene/html/) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/DScene/html/)
- [Team talks](https://dlabportal.jpl.nasa.gov/resources/darts-lab-talks/) (https://dlabportal.jpl.nasa.gov/resources/darts-lab-talks/)
- [2019, 2020 course slides](https://dlabportal.jpl.nasa.gov/documentation/coursetutorial-links/) (https://dlabportal.jpl.nasa.gov/documentation/coursetutorial-links/)
- [Sphinx docs](https://dartslab.jpl.nasa.gov/dlabdocs/) (https://dartslab.jpl.nasa.gov/dlabdocs/)
- [Jupyter notebooks](https://dlabnotebooks.jpl.nasa.gov/hub/login) (https://dlabnotebooks.jpl.nasa.gov/hub/login)
- [DARTS Lab QA site](https://dartslab.jpl.nasa.gov/qa/) (https://dartslab.jpl.nasa.gov/qa/)

### 16.2. Design

#### 16.2.1. SceneObject flags

| Flag       | Visualization | Sensors | Collision |
|------------|---------------|---------|-----------|
| VISUAL     | x             | x       |           |
| COLLISION  |               |         | x         |
| PHYSICAL   | x             | x       | x         |
| ORNAMENTAL | x             |         |           |
| STICK      | x             |         |           |

#### 16.2.2. CallbackRegistry

The callback registry was designed to meet the following requirements:

1. Callbacks should be uniquely identified via a string.
2. The callback execution should be ordered, and new callbacks should be able to be inserted at an arbitrary location in that order.
3. Can accept callbacks in C++ or Python.
4. Clean interface, i.e., the user should not need to wrap their callback functions. This should be done for them.
5. No memory leaks.

Here is how each of these requirements was met:

1. The callback registry uses an `std::unordered_map` to associate a name (the key, given as a string) with each callback (the value, given as a callback class that wraps the callback function). Hence, each callback is uniquely identified via a string.
2. An `std::vector` is used to store the names of the callbacks. Hence, each name is associated with an index (its position in the vector). When the callback registry is executed, i.e., the callbacks are executed, the registry loops through this vector in order and then executes the callbacks associated with them. Hence, the callbacks are ordered. Moreover, an index can be specified when adding a new callback to insert it in a particular location.
3. Rather than store the callback functions directly, a Callback wrapper is used. The Callback wrapper is defined using an abstract base class in C++ called `_CallbackBase`, that has a virtual method called `execute`. A concrete C++ class is derived from this, `CallbackCpp`, whose `execute` method is overridden to execute a stored C++ callback. Similarly, using SWIG directors, a Python class is derived from `_CallbackBase`, called `_CallbackPy`, whose `execute` method is overridden to execute a stored Python callback function. The `CallbackRegistry` calls `execute` on each of the `_CallbackBase` shared pointers, and the class inheritance takes care of executing the C++ or Python callback. The Callback wrapper classes are organized as follows:

```

skinparam backgroundColor transparent
@startuml

Class _CallbackBase
{
+execute()
+getCallback()
}

Class CallbackCpp extends _CallbackBase
{
-_f
+execute()
+getCallback()
}
note left of CallbackCpp: _f is type std::function<void(>

Class _CallbackPyBase extends _CallbackBase
{
-_py
+execute()
+getCallback()
+getPyObject()
+setPyObject()
}
note right of _CallbackPyBase: _py is type PyObject*

Class _CallbackPy extends _CallbackPyBase
{
-_f
}
note right of _CallbackPy: _f is a Python function

```

- The `CallbackRegistry` methods accept/return the callback functions directly, and the registry itself takes care of wrapping/unwrapping them in the aforementioned callback wrappers.
- Since the `CallbackRegistry` creates the callback wrappers, it must also take care of ensuring the Python reference counts (for the Python callback wrappers) are incremented and decremented correctly. This is handled via an intermediate C++ class `_CallbackBasePy` that increments the Python `_CallbackPy` object upon creation and holds a reference to it, and decrements the object upon deletion, so the referenced `_CallbackPy` will be destroyed as well. The 3rd party module `objgraph` was used to ensure that there are no memory leaks in Python, and `valgrind` was used to ensure there were no memory leaks in C++. See <https://mg.pov.lt/blog/hunting-python-memleaks.html> blog post for more on `objgraph`.

## 16.3. Usage

### 16.3.1. CallbackRegistry

- `addCallback` - Used to register a callback function. An optional index can be used to place the new callback in a specific position.
- `getCallback` - Returns the callback function associated with the given name.
- `removeCallback` - Removes the callback associated with the given name
- `callbackExists` - Returns true if a callback is registered under name. Returns false otherwise.
- `callbackIndex` - Returns the index of the callback in the registry associated with name.
- `getCallbackNames` - Returns a vector of all names associated with callbacks in the registry.
- `execute` - Executes all callbacks in the registry in order.
- `executeReverse` - Executes all callbacks in the registry in reverse order.
- `dumpCallbacks` - Dumps info about the callbacks in the registry.

In addition, the C++ and Python wrappers use operator overloading so the registry can be accessed in a map-like or dictionary-like fashion, respectively. Python also overloads the functions needed to use the `is` keyword to determine if a name is in the registry and iterate through the names, i.e., it can be used in the same way the `is` keyword is used on dictionaries.

## 16.4. Software

### 16.5. Raw documentation



TBD: Need scrubbing before integration.

#### 16.5.1. DScene: support of Hapke and Principled material through BodyDParam and general improvements



TBD: Needs scrubbing. Notes brought over from [issue](https://dartsgitlab-internal.jpl.nasa.gov/darts-lab/jpl-internal/all-access/development/scene-geometry/dscene/-/issues/23) (https://dartsgitlab-internal.jpl.nasa.gov/darts-lab/jpl-internal/all-access/development/scene-geometry/dscene/-/issues/23).

This issue aims to describe how geometry inputs to `BodyDParam` get translated to scene objects, key take-aways from this process, and a proposed implementation for improving this system and allowing for setting other material types through `BodyDParam`.

The following is a description (to the best of my understanding) of how inputs to `BodyDParam` (specifically the geometry-related ones) are handled and passed from class to class until they reach the C++ objects `FacadePartGeometry` or `FacadeTopoGeometry`, which are attached to the scene.

► Click to expand

From the above explanation, we draw some key points:

- The parameters that are needed/used depend on the geometry type: for example, topos will require different inputs than spheres. See table below for a breakdown.
- The `scene_object_flag` is always used, independently of geometry type.
- `extra_parameters`, `scale`, `material`, `visibility`, `translation`, and `quaternion` are used for all geometry types **except** `TRIANGLE_MESH`.
- There is no way to configure a material other than Phong. The `ambience`, `textureFile`, `diffusivity`, `specularity`, and `emissivity` parameters are used to configure this Phong material.

The following table shows the geometry-specific parameters:

| Geometry Type | Parameters                   |
|---------------|------------------------------|
| TRIANGLE_MESH | topo_url                     |
| FROM_FILE     | filenames, override_material |
| CUBE          | length, width, height        |
| SPHERE        | radius                       |
| CYLINDER      | height, radius               |
| CONE          | height, radius               |

A new way to define geometries in `BodyDParam` is proposed that aims to support all material and geometry types. This new system is designed in such a way that no fields need/can be input if they are not used for that specific material or geometry. This is, if the geometry type is `CUBE`, then `override_material` cannot be used, as this is unique to `FROM_FILE`.

`BodyDParam` would have a `geometry` field which is of the type `Dict[str, Geometry]`. `Geometry` would be an `InputDict` with the following fields: `scene_object_flag`, `geometry`, `material`. The `geometry` field could be populated by instances of the following `InputDict`: `TopoGeometry`, `FileGeometry`, `CubeGeometry`, `SphereGeometry`, `CylinderGeometry`, `ConeGeometry`, which would have the necessary fields to populate the respective geometry types. The `material` field could be populated by instances of the `InputDict` `PhongMaterial`, `PrincipledMaterial`, or `HapkeMaterial` OR by instances of the `DScene_Py` objects `PhongMaterial`, `PrincipledMaterial`, or `HapkeMaterial`. We have separate `InputDict` and `DScene_Py` objects because `InputDict` objects are more user-friendly: they play better with IDEs, the constructor can be used to set the values, allows better validation... This is an example of how creating a `BodyDParam` might look:

```
my_body_dparam = BodyDParam(
    ...,
    geometry = {
        "my_cube" : Geometry(
            geometry = CubeGeometry(length=1, width=2, height=3, translation=[0, 1, 0]),
            material = PhongMaterial(ambient_color=[0.2, 0, 0.05])
        ),
        "file" : Geometry(
            geometry = FileGeometry(file="shape.py", override_material=True),
            material = PhongMaterial(ambient_color=[0.2, 0, 0.05])
        ),
    }
)
```

The data from these objects would then have to be "re-shaped" so that it conforms to the format expected by the `DartsBody::setBodyParams` function. Thus most of the process detailed above would work exactly the same. However, material data represents a significant challenge. `FacadePartGeometrySpec` has no way of storing material data other than Phong-related. At first, I thought a possible way around this would be to change the material after the geometry had been created. However, geometries are not created until the `DartsBody::addPartGeometries` function is called, which is done after the assemblies are created and the params bind. Thus it is inevitable that the C++ `FacadePartGeometrySpec` class has to change to support the new materials.

The cleanest implementation would be removing the Phong-related variables in the `FacadePartGeometrySpec` class (`ambience`, `textureFile`, `diffusivity`...) and instead have three class variables of the types: `std::optional<PhongMaterial>`, `std::optional<PrincipledMaterial>`, `std::optional<HapkeMaterial>`. Then, the `FacadePartGeometry` constructor (or `FacadeTopoGeometry`, if supported) would call `setPhongMaterial`, `setHapkeMaterial`, or `setPrincipledMaterial` as needed (instead of always creating a Phong material and populating it with parameters from `FacadePartGeometrySpec`).

The above detailed implementation would be backward-incompatible, as the Phong-related variables would be removed. A backward-compatible implementation would be possible by keeping these variables (but deprecating them) and also having the `std::optional<PhongMaterial>`, `std::optional<PrincipledMaterial>`, and `std::optional<HapkeMaterial>`. Then, in the `FacadePartGeometry` constructor, these material objects would be given priority to set the material. If none are defined, then we would default to creating a Phong material and populating it with parameters from `FacadePartGeometrySpec`, thus maintaining previous behaviour.

**NOTE:** there is an additional way with which geometry appear to be loaded in DARTS, and that is through the `Dshell++Scripts/python/MbodyDScene.py` module, which handles geometry dictionaries in a different way (for example, this method supports the `attach_topo_to_body_frame` for `topo`). The new way to define geometry in `BodyDParam` **would not** be compatible with `MbodyDScene`. **From Abhi:** *MbodyDScene is obsolete and can be ignored.*

An additional improvement could be made to facilitate defining extra parameters. These would be `InputDict` that contain "groups" of extra parameters and that should be unpacked on the `extra_parameters` field. The following could be defined in `geometry.py`:

```

class DBulletExtraParameters(InputDict):
    """Parameters to configure DBullet

    Parameters
    -----
    max_vertices : Optional[int]
        max number of vertices
    ...
    """
    max_vertices: Optional[int] = None
    min_clusters: Optional[int] = None
    ...

class MayaSceneExtraParameters(InputDict):
    """
    ...
    """
    maya_file: Optional[str] = None
    maya_texture: Optional[str] = None
    maya_bump_map: Optional[str] = None

```

and we could define `BodyDParam` as:

```

my_body_dparam = BodyDParam(
    ...,
    geometry = {
        "my_cube" : Geometry(
            geometry = CubeGeometry(
                length=1, width=2, height=3, translation=[0, 1, 0],
                extra_parameters = {
                    **DBulletExtraParameters(max_vertices=2),
                    **MayaSceneExtraParameters(maya_file="foo"),
                }
            ),
            material = PhongMaterial(ambient_color=[0.2, 0, 0.05])
        ),
    }
)

```

this would help expose extra parameters and add validation to them early on.

### 16.5.1.1. Implementation

Most of the features here described have been implemented in `DScene-R1-15j`, `DshellCommon-R1-56`, `FacadeScene-R1-09d`, and `Ndarts-R1-66g`:

- `FacadePartGeometrySpec` can now hold information about other materials apart from Phong, and this information can be populated in the `DVar::Branch` constructor
- This information is now used in `createPartGeometry` and in `(Ndarts) DartsBody::addPartGeometries` to set the correct material in the `FacadePartGeometry` or `FacadeTopoGeometry` respectively.
- `geometry.py` has been renamed to `shapes.py` and the format used to specify these parts in the python side has changed.

See the following example of how to use `BodyDParam` in the update:

► [Click to expand](#)

### 16.5.1.2. Extra parameters situation

The `XXXExtraParameters` system is interesting, and there have been proposals to extend the shown proposal even further by making using these classes mandatory. This would mean that every time a new set of extra parameters is defined one would need to create its `InputDict` counterpart. Then, we could have a class `ExtraParameters` in `geometry.py`:

```

class ExtraParameters(InputDict):
    """Parent class for all InputDict that define a family of extra parameters"""

```

and have the `extra_parameters` field only accept a list of subclasses of `ExtraParameters`:

```

class _ShapeOrFileGeometry(Geometry):
    ...
    extra_parameters: Iterable[ExtraParameters]

```

Then, the `_ShapeOrFileGeometry` should implement a `populate_geometry_spec` function that unpacks each given `ExtraParameters` into a single dictionary (the expected format by the rest of the code). Subclasses of `_ShapeOrFileGeometry` would need to call the parent `populate_geometry_spec` without their own implementation of `populate_geometry_spec`.

### 16.5.2. DScene: Add better support for client scenes that do not support scene graph constructs



**TBD:** Needs scrubbing. Notes brought over from [issue](https://dartsgitlab-internal.jpl.nasa.gov/darts-lab/jpl-internal/all-access/development/scene-geometry/dscene/-issues/21) (https://dartsgitlab-internal.jpl.nasa.gov/darts-lab/jpl-internal/all-access/development/scene-geometry/dscene/-issues/21).

The goal of this issue is to make it easier to work with client scene's that do not inherently support `scene graph` paradigms where nodes have a parent/child hierarchy. While `Dspace` and `Bullet` have scene graph support, `OptixScene` (nor the proposed `[DMeshScene]`(https://dartsgitlab-internal.jpl.nasa.gov/darts-lab/jpl-internal/all-access/development/scene-geometry/dmesh/-issues/46)) have this. Our original design took scene graph support for granted, and is set up so that geometry world poses are updated by broadcasting the relative poses

for the scene frames in the scene graph backbone. When the client scene does not support scene graphs, the current design is awkward since we need to set the absolute - and not relative - poses of the scene frames. The `OptixScene` design reflects this awkwardness where the scene frames are being organized into a scene graph hierarchy, but with layers added to ensure that the absolute poses are up to date. We need to refactor to clean up this code, and make it easier to bring in non-scene-graph scenes such as the new `DMeshScene`

### 16.5.2.1. Background

Our initial implementation of `DScene` includes the `SceneFrame` class that is used to build up a scene graph hierarchy to which we attach geometries, lights etc. These scene frame instances are proxies for and track the `DFrame::Frame` instances in the simulation world. The `updateSceneFrameTransforms()` method is called by the simulation to push updated relative transform values to sync up the scene frame and frame worlds.

It is worth noting that we already deviate from the frames scene graph hierarchy and structure in a couple of places: - We allow for the use of client scene specific `origin scene frame` to serve as the center of the client scene world. This is to handle single precision accuracy of many client scene implementations. - We allow for `rooted scene frames` where a scene frame is detached from its natural parent and instead attached directly to the `root scene frame`. Once again this is driven by precision considerations.

### 16.5.2.2. Proposed solution:

The main idea is to not use parent/child scene graph constructs for client scenes that do not have scene graph support. Instead, all scene frames in such client scenes will be `rooted`, i.e. they will all be directly attached to the root scene frame. The current design is set up to the broadcast absolute poses for rooted scene frames, and relative poses for non-rooted ones. Making all client scene frames rooted will automatically get the broadcasting process to broadcast the right data. Changes needed:

- Make the `SceneFrame::addChild` and `SceneFrame::removeChild` methods will be empty for non-scene-graph client scenes.
- The concept of a rooted scene frame currently is at the `FacadeSceneFrame` level. This concept instead should be pushed down to the `SceneFrame` level so that it is available to all client scenes.
- When broadcasting poses, a check should be made whether the client scene frame is rooted or not, and the relative or absolute pose info should be sent.
- We should still keep the origin frame concept for client scenes even though all transform updates will be made in double precision. It avoids the issue of z-fighting that come up with a pair of planetary surface assets.
- The client scenes should have a global flag that says if they are scene graph type or not. The manager level `updateSFT` loops can then be cleaned up by acting on this setting instead of making checks for each scene frame to see if it has a parent or not. Also the attach methods can automatically disable attachment based on this flag.
- Need to simplify the manager's `updateSFT` implementation for this more granular rel transform management.

Changes:

- `FacadeSceneFrame` class will have a `_root_transform` and `_stale_root_frame` members
- the existing `rootTransform()` getter method will continue to use the current method of computing the value on the fly. Caching these values requires detecting when they are stale and that can be a headache. For non-SG scene frames there is no nesting so there is not big cost. The root frame value is rarely needed anyway at run-time - mostly for dumps.
- `DartsFacadeScene::updateSceneFrameTransforms` will mostly continue to work as in the past.
- it still updates the `root2origin` transforms for all client scenes first
- it also updates the root transform value for all `MSF`
- then it will loop through all manager scene frames (`MSF`) to set `relTransform` values
- For `MSF` that have a parent `MSF` it will set the `relTransform` as the parent to scene frame transform. Note that the generalization being made here is that just because the `MSF` has a parent scene frame, it is not necessary the the associated client scene frames have parents. The ones in client scenes with no scene graph support will not have parent scene frames.
- For `MSF` with no parent the `relTransform` will be set to the root to scene frame transform
- For ones with a parent, a subtle change made is to explicitly call the `relTransform` method
- `FacadeSceneFrame::relTransform()` will be updated as follows
- For each client scene (`CS`) it will loop through the corresponding client scene frame (`CSF`)
- if the `CSF` has a parent `CSF`, then the `MSF relTransform` will be passed on as is to the `CSF relTransform` call
- if the `CSF` does not have a parent `CSF`, then (both of these options will use the root to `MSF` transform available with the `MSF`
- if the `CS` has no origin scene frame, then the root to `MSF` transform will be passed to the `CSF relTransform` call
- if the `CS` has a origin scene frame, the origin to `MSF` transform will be applied instead.
- For client scenes with no scene graph support
- Set the `_is_scene_graph` member to `false` in the scene constructor
- Disable the scene frame `addChild` and `removeChild` method
- Scene dumps should show the same origin and root transform values, however the relative transform will be different, so regtests may need to be updated
- get rid of the `updateTree0`, cached transforms etc from the client scenes. They should no longer be needed



## 17. FacadeScene

### 17.1. Background

#### 17.1.1. Reference & Source material

- [FacadeScene Doxygen docs](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/FacadeScene/html/) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/FacadeScene/html/)
- [Team talks](https://dlabportal.jpl.nasa.gov/resources/darts-lab-talks/) (https://dlabportal.jpl.nasa.gov/resources/darts-lab-talks/)
- [2019, 2020 course slides](https://dlabportal.jpl.nasa.gov/documentation/coursetutorial-links/) (https://dlabportal.jpl.nasa.gov/documentation/coursetutorial-links/)
- [Sphinx docs](https://dartslab.jpl.nasa.gov/dlabdocs/) (https://dartslab.jpl.nasa.gov/dlabdocs/)
- [Jupyter notebooks](https://dlabnotebooks.jpl.nasa.gov/hub/login) (https://dlabnotebooks.jpl.nasa.gov/hub/login)
- [DARTS Lab QA site](https://dartslab.jpl.nasa.gov/qa/) (https://dartslab.jpl.nasa.gov/qa/)

### 17.2. Design

#### 17.2.1. Registering client scenes

##### 17.2.1.1. Syncing up client scene objects

- A requirement on the `DScene` approach is that client scenes can come and go at any time - before and even after the manager scene has been populated with objects. Also objects can be added and removed from the scene at any time and all client scenes are expected to stay in sync.
  - Indeed we often have the scenario where the manager scene is created, various geometry objects are added to it, and only later are specific client scenes and collision detection registered with the manager. At the early stage before even a client scene has been registered, there is no "user" for the objects, and there is no real reason to populate them. We expect the manager to sync up and populate the client scenes with all the objects it has.
  - This requirement means that the scene manager needs to keep track of all of the heterogeneous objects, and for each of these objects to know enough about its properties so that it can introspect and pass them on to a new client scene so that it can faithfully create a mirror instance of the object.
  - To achieve this, the manager keeps a list of all objects that are created in a list of `deferred objects`. The idea being that the creation of proxy versions of deferred objects can be deferred to when a new client scene has been registered. Every scene frame and scene object at the manager level inherits from the `DeferredObject` class which has the `_realize` and `_initialize` methods. When a new client scene is created the following steps are taken:
    - the `_realize()` method is called for each of the objects in the deferred list with the client scene as an argument. This method calls constructors to create a proxy version in the client scene. Note that some types of objects may be unsupported by some client scenes (eg. `light` object for collision client scenes), and in this case this the `_realize` method does nothing and does not create a proxy method.
    - following this, the `_initialize()` method is called for each of the objects in the deferred list with the client scene as an argument. The job of this method is to pass and set the parameters in the new proxy child to get it ready for use. Nothing is done if a proxy implementation for the client scene was not created. If there is an implementation, then this method walks through the parameters assigned to the manager version of the object and calls the appropriate methods on the proxy implementation with the parameter values.

#### 17.2.2. Various object types

##### 17.2.2.1. Part geometries

Part geometry instances can be added to add geometry to the overall scene.

##### 17.2.2.1.1. Multiple shapes support

Geometry definitions can come in various flavors. They can be parameterized primitive shapes (eg. boxes, spheres, cylinders), procedurally created meshes with faces and vertices, or from CAD format `obj`, `glTF` etc files. All of these `shape` choices are supported.

- We have 2 choices for the implementation:
  - Create an empty part geometry instance via `createPartGeometry()`, and later flesh it out by assigning it a shape and relevant parameters.
  - Or, since each shape type requires different parameters, we would can have multiple `createPartGeometrySphere()`, `createPartGeometryCylinder()` etc methods with each taking custom parameter arguments. We do have the option of having such a single constructor and using a single `std::variant` argument for the various shape parameters to help create the right shape object, but `std::variant` is not friendly for the Python interface generation via `SWIG`.
- So the choice is between a single creation method, and multiple shape setting methods, or multiple constructors. We have chosen the 2-step first approach for part geometries, and the 1-step second approach for topo geometries. So there is a single `createPartGeometry()` method, and multiple method options such as `sphere()`, `cylinder()` etc to assign a shape and set the shape parameters for the object.
- One simplification would be to have a single `shape()` method that takes different parameter value arguments for the different shape types - however this gets ugly too when we multiple shape types take the same number of arguments (eg. `cylinder` and `cone`).

### 17.3. Usage

### 17.4. Software

### 17.5. Raw documents

## 18. OptixScene

### 18.1. Background

#### 18.1.1. Reference & Source material

- [OptixScene Doxygen docs](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/OptixScene/html/) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/OptixScene/html/)
- [Team talks](https://dlabportal.jpl.nasa.gov/resources/darts-lab-talks/) (https://dlabportal.jpl.nasa.gov/resources/darts-lab-talks/)
- [2019, 2020 course slides](https://dlabportal.jpl.nasa.gov/documentation/coursetutorial-links/) (https://dlabportal.jpl.nasa.gov/documentation/coursetutorial-links/)
- [Sphinx docs](https://dartslab.jpl.nasa.gov/dlabdocs/) (https://dartslab.jpl.nasa.gov/dlabdocs/)
- [Jupyter notebooks](https://dlabnotebooks.jpl.nasa.gov/hub/login) (https://dlabnotebooks.jpl.nasa.gov/hub/login)
- [DARTS Lab QA site](https://dartslab.jpl.nasa.gov/qa/) (https://dartslab.jpl.nasa.gov/qa/)

### 18.2. Design

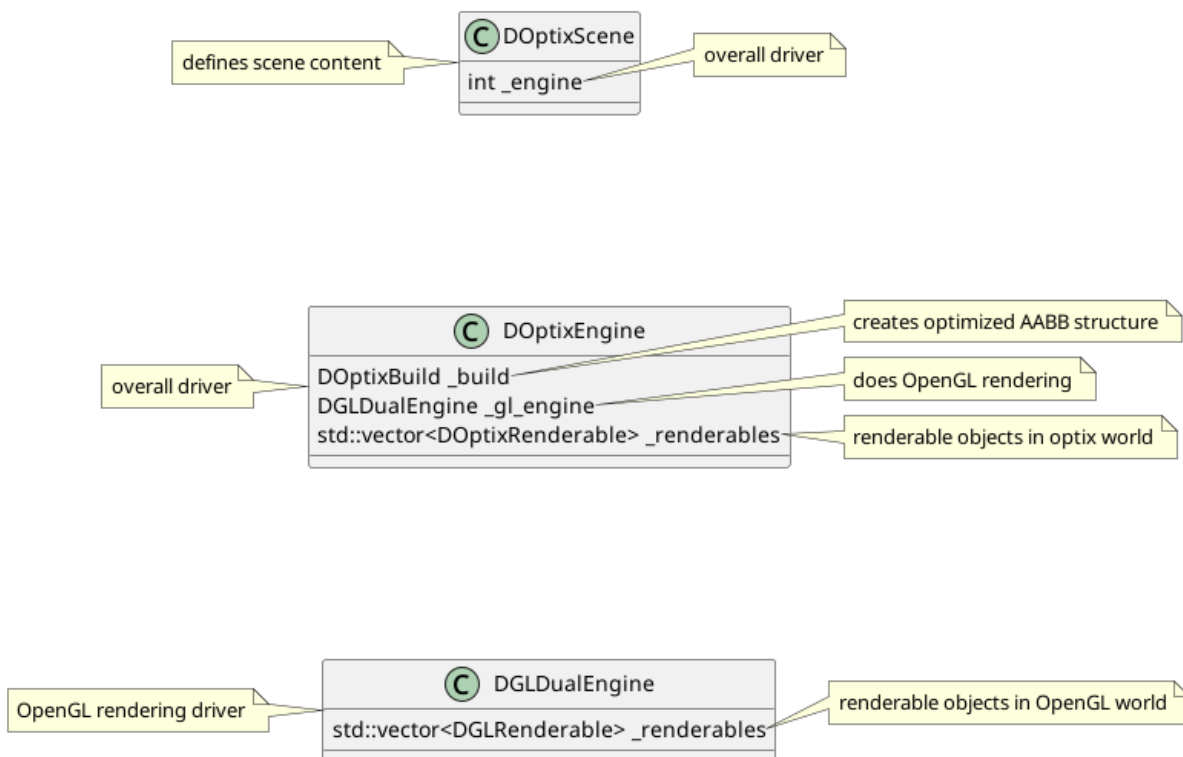
#### 18.2.1. Key components and their relationships

The following provides an overview of the overall structure of the `OptixScene` architecture for sensors and viewports.

- The `Engine` keeps track of all the sensor `Pipelines` (so they can be marked for recreating)
- Need `Builder::update` to rebuild the acceleration structure (IAS) and the `Engine::buildChangedOnQueue` to regenerate all the sensor pipelines, if renderables have been added/deleted
- each viewport has a sensor that is derived from a `SceneObject`. This is different from other client scenes such as `DspaceScene`.
- The `DOptixEngine` instance also keeps a `DGLDualEngine` instance which is used for `OpenGL` rendering. This latter engine keeps a list of `DGLRenderable` objects which are associated with lines, trails etc that are needed.

some PlantUML diagrams to capture the design

#### Key OptixScene components and their relationships



#### 18.2.1.1. Event queue processing

The problem is that the event queue thread alternates between processing all the ordinary events that are added by most optix calls, and poll events with the `glfwWaitEventsTimeout` call, if there are no gui events it will wait the full time even if there are ordinary events from optix calls queued up. So the glfw wait timeout is causing the processing of ordinary events to be delayed. This is made worse if the ordinary event is one that requires the sim thread to sync, such as one that returns a value, because then the sim thread also has to wait for the glfw timeout. This may be why ROAMS is taking a long time to start up. I think the proper fix is really to spawn another thread for each GUI viewport and call `glfwWaitEvents` in a loop on that thread. This will cause the thread to block until the moment a glfw GUI event becomes available. The GUI event handler would be running on a different thread from our existing event queue thread, so it couldn't safely make low level optix calls. To be thread safe, the handler for these glfw events could add an event to our existing event queue. This would remove the need for any explicit sleeps or busy waiting.

The **current solution** that seems to work well is to switch the main loop to use `glfwPollEvents` when there are events in the queue, and to fallback to `glfwWaitEventsTimeout` with a `10ms` timeout when idling. This seems to keep the system responsive, while also keeping the CPU from maxing out. Note that the ROAMS gui is by itself maxing out the CPU - which is a separate issue that needs to be fixed.

### 18.2.1.2. Sensors and viewpoints

The split between sensors and viewpoints is introduced in `OptixScene` and is differentiated from `DspaceScene`'s use of viewpoints.

Sensors are the data generation classes and inherit from `DOptixSensor`, for example `DOptixCameraSensor` and `DOptixLidarSensor`. These classes define appropriate ray generation capabilities and sensor configurations. These classes can generate data in any manner they deem necessary and do not rely on GL functionality to perform their respective data generation.

Viewports are data display classes which can allow sensors to draw their data to a screen (including drawing frames, trails, or text over the sensor data), or allow interaction with a sensor. These classes inherit from `DOptixViewport`, and include `DOptixSensorViewport` and `DOptixInteractiveViewport`.

The `DOptixCameraSensor` and `DOptiLidarSensor` classes do the actual work. We can associate viewpoints with them to display or interact with the sensor data, or wrap the sensor for backward compatibility with old-style viewport uses.

A viewport-sensor combination can be created together via the scene. For example: `* scene.createViewport()` and `scene.createOffscreenViewport()` create a perspective camera sensor with a `DOptixInteractiveViewport`.

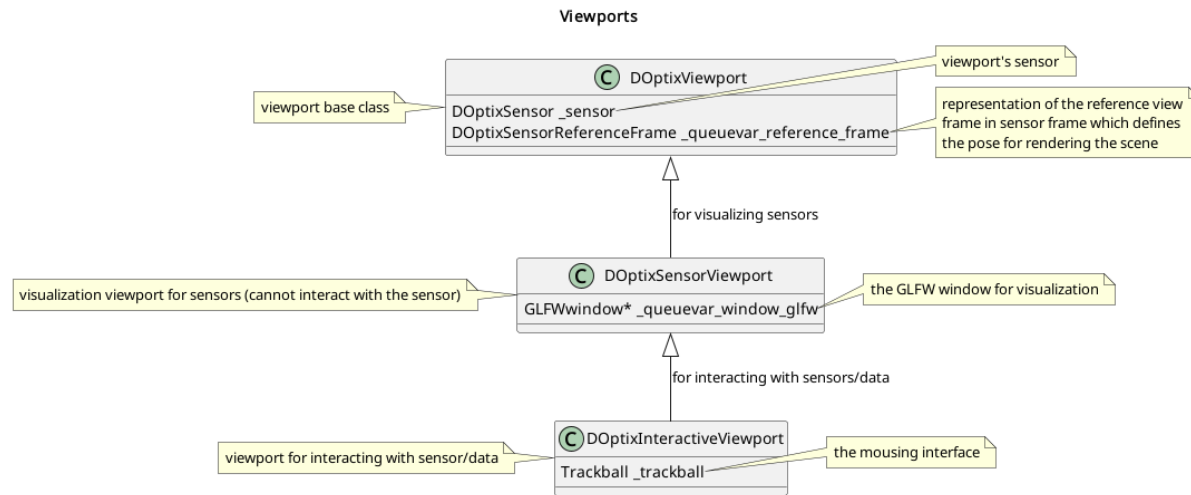
Here is a description of how **viewports** are implemented. As shown in the figure below, a viewport can be a

- An empty wrapper around a sensor: `DOptixViewport`
- A window where sensor data is displayed without interaction: `DOptixSensorViewport`
- A window where the user can interact with the sensor or sensor data: `DOptixInteractiveViewport`

`DOptixInteractiveViewport` extends `DOptixSensorViewport`.

Also, as shown in the figure, each viewport has a registered **sensor** which does the real work of doing the appropriate sensor specific rendering. The currently supported sensors are

- camera sensors
- lidar sensors



Each viewport/sensor has an independent rendering pipeline independent of others. So multiple viewpoints and sensors can co-exist.

#### 18.2.1.2.1. 3D graphics visualization viewpoints

On-screen viewpoints are either `DOptixSensorViewport` or `DOptixInteractiveViewport` type. These viewpoints use `GLFW` to create a window to which sensor data and GL overlays can be drawn. The sensor data along with scene ornaments are drawn to the viewport by `DOptixSensor::_GLDraw()` such that each sensor can define how its own data should be drawn. A 2D window overlay can be added with `ImGui` to add text such as current `FPS` or "DARTS Lab".

For camera sensors, this is simply the rendered buffer as an image, while for Lidar's, a color map is used to visually display the depth map output. To save the sensor's information directly to a file, you can call `viewport.writeContentsToFile()`. For a camera, this will save the image, without any overlays. When a window is used, and it is necessary to save the contents for the window to a file (e.g. to save an image of a lidar's point cloud drawn over an image of terrain), you can call `viewport.writeWindowedContentsToFile()`. This writes out the pixels being displayed in the viewport's window.

Furthermore `DOptixInteractiveViewport` has a mousing interface via a `Trackball` model that supports zooming, panning, rotating and resizing features. The trackball model's mousing callback updates the viewport's reference view frame based on mouse motion events. Additionally, only a `DOptixInteractiveViewport` supports `pointCameraAt(eye, lookat, up)` functionality, as this is considered interacting with the sensor through the viewport.

In certain cases, included tests, it may be beneficial to have interaction (e.g. `pointCameraAt(eye, lookat, up)`) without a window, or it may be beneficial to hide the `glfw` window so that a developer can keep working while tests run. In these cases, viewport can be created in `headless` or `hidden` mode. These modes are equivalent to the graphics concept of `headless` and `offscreen` rendering.

`headless` : the viewport's window and all GL-related calls are disabled. This means no window will be shown and no graphics context is required (i.e. no X context for linux machines). Additionally, all GL-draws are disabled, resulting in disabling of draw frames, trails, text, etc. Finally, `viewport.writeWindowedContentsToFile()` is disabled, as this saves the GL-window to file. `viewport.writeContentsToFile()` is still available as it writes the underlying sensor's data to file without any GL additions.

`hidden` : the viewport's window will be used, but will remain invisible to the user. Since a `hidden glfw` window is not a well-defined rendering target, a `gl framebuffer` object is used as the render target regardless of `glfw` window visibility. When the window is visible, this framebuffer is `blitted` to the `glfw` window (copied with interpolation). This allows hiding and showing the `glfw` window without losing functionality. For a `DOptixSensorViewport`, this also allows the sensor's GL render target to be independent of the size of the `glfw` window. For a `hidden` window, `viewport.writeWindowedContentsToFile()` and `viewport.writeContentsToFile()` are both available.

`headless` and `hidden` can be passed as booleans through `extra_parameters` to `DOptixSensor::createViewport()`, `DOptixScene::createViewport()` or `DOptixScene::createOffscreenViewport` :

```
virtual DOptixViewport* createViewport(DScene::ViewportType vw_type = DScene::VIEWPORT_TYPE_SENSOR,
    unsigned int width = 0,
    unsigned int height = 0,
    unsigned int antialiasing_factor = 0,
    const std::map<std::string, boost::any>& extra_parameters = {});
```

```
virtual DOptixInteractiveViewport* createViewport(const std::string& name,
    unsigned int width,
    unsigned int height,
    bool interactive = true,
    bool keyboard_interaction = false,
    unsigned int antialiasing_factor = 0,
    const std::map<std::string, boost::any>& extra_parameters
    = std::map<std::string, boost::any>()) override;
```

```
virtual DOptixViewport* createOffscreenViewport(const std::string& name,
    unsigned int width,
    unsigned int height,
    unsigned int antialiasing_factor = 0,
    const std::map<std::string, boost::any>& extra_parameters = {}) override;
```

C++ example:

```
DOptixViewport* viewport = sensor->createViewport(DScene::VIEWPORT_TYPE_INTERACTIVE,
    0u,
    0u,
    0u,
    { { "hidden", true }, {"headless", false} });
```

Python example:

```
viewport = scene.createViewport("viewport", 1280, 720, extra_parameters={"headless": True, "hidden": False})
```

#### 18.2.1.2.1.1. Visualizing multiple lidar sensor outputs

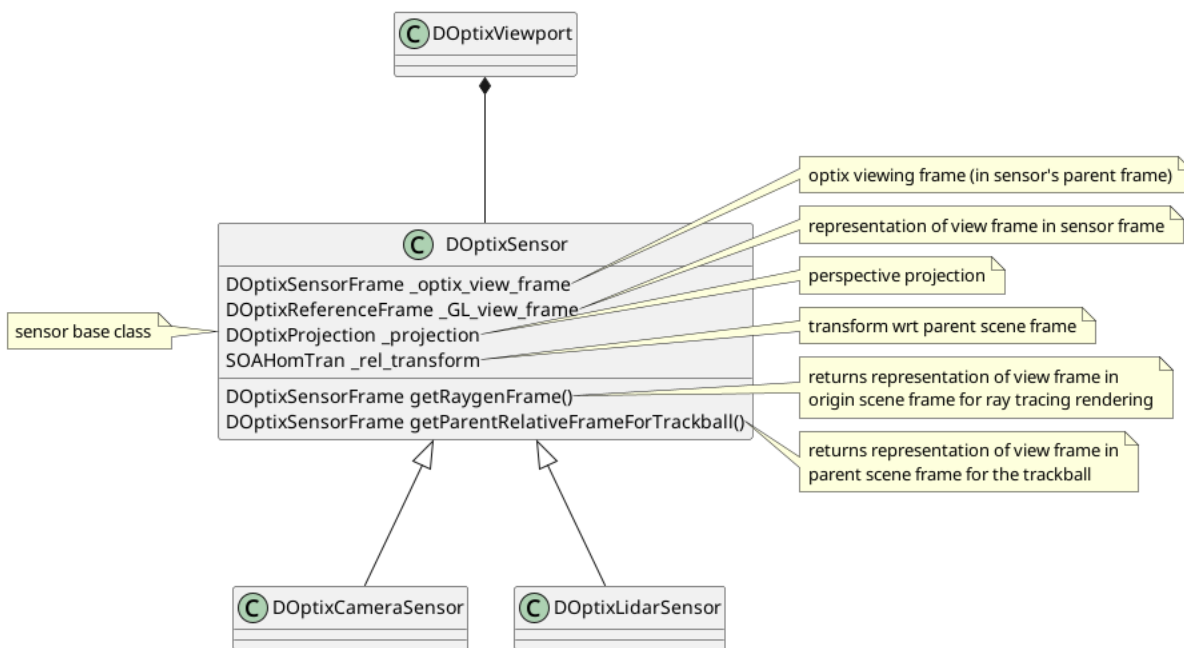
Having this feature requires us to combine the render results of multiple separate pipelines - one for the plan view camera, and one or more lidar sensors - into a single viewport display. An approach to doing this would be to

- ensure that the plan view pipeline gets rendered last after all the lidar ones have been rendered
- register the lidar sensors with the plan view viewport
- update the `GLDraw` part of the viewport code to call `GLDraw` on each of the registered lidar sensors as well to add in their contributions to the plan view on-screen display.

#### 18.2.1.2.2. Sensors

Sensors are derived from `DScene::SceneObject` class, and can thus be attached to scene frames (for chase view) or be free standing. Each sensor has a full rendering pipeline containing a `raygen action` and multiple `filters` specific to the sensor type.

## Viewport Sensors structure



Each sensor has a view frame specified in different frame representations as follows:

- **GL view frame** : which specifies the **rendering pose** in the sensor's frame. This view frame is identical is kept in sync with the parent viewport's reference view frame. This view frame is used as follows.
  - Each sensor of `camera` type has a `DGLDualRenderFilter` filter as the last filter in its pipeline which contains a list of `GL` renderable objects for frame axes, bounding boxes, trails, text labels. This filter uses `OpenGL` to render and add such content to the the optix ray-tracing based rendering buffer in the on-screen viewport display.
  - This view frame is used to visually display the point cloud range output of sensors of `lidar` type via the `DOptixGuiViewport::_redrawOnQueue()` method and merge it into the on-screen viewport display.
- **optix view frame** : this representation is in the scene's `origin frame` and is used by `optix` for the ray tracing rendering.
- **trackball view frame** : this representation is in the parent scene frame. It is used for re-centering the trackball's internal frame to avoid gimbal lock singularities.

### 18.2.1.2.2.1. Viewport/sensor poses and mousing

As described above, sensors have a viewing frame (or pose) used for the optix rendering. The `pointCameraAt()` and `setCameraTransform()` viewport methods can be used to change the sensor's pose.

For **perception** sensor simulations, the parent viewport is used to visualize, write to file, and output the rendered data. In this case the viewport has the same pose as the sensor; and mousing interface is disabled to keep them in lock.

For visualization the `DOptixGuiViewport` provides additional support for mousing interface. The mouse can be used to change how we view the sensor data for interactive visualization. For this case, the viewport pose should be independent of the sensor pose.

- We need this decoupling to visualize lidar point cloud data from different vantage points. For this case, the mouse motion callback gets the current desired pose and updates just the `queuevar_reference_frame` value to set the viewport's viewing frame, and does not change anything in the sensor.
- For simple 3D visualization (instead of perception sensor models) we want to in fact lock the sensor pose to the viewport pose and control them via the mouse. For this case, the mouse motion callback gets the current desired pose and updates just the `queuevar_reference_frame` value to set the viewport's viewing frame, but also simultaneously calls the sensor's `setReferenceFrame` method to sync up the sensor's pose to match that of the viewport. Also, for this case, when the `pointCameraAt` etc methods are called to directly set the sensor's pose, the new values are also used to sync up the viewport's pose so they stay in sync.

### 18.2.1.2.3. Sensors and the event queue

Most of the viewport methods do their work by posting events on the event queue. Sensors keep track of both `sim` and `event queue` versions of data. The former are updated and used on the main thread, while the latter are used by the events on the event queue. Since the event queue will normally lag the sim thread in time, it is very important that each thread only use the data mean for it to avoid data race issues. Each sensor instance manages has 2 member instances of `SensorParameters` named `_sim_params` and `_queued_params`. The helper `_sensorParameters(bool on_queue)` method can be used to retrieve, with `true` argument returning the queued version and `false` the sim version. Whenever, the sim parameters are updated from the main thread, it is usually necessary to register an event on the event thread to set the same values in the queued parameters. Event queue methods should always be working with the queued paramters, and main thread ones with the sim parameters.

From Aaron Gaut's comments from <https://dartsgitlab-internal.jpl.nasa.gov/darts-lab/jpl-internal/all-access/development/rendering/optixscene/-/issues/14>

I wasn't able to get to this but basically the approach should be to first identify which thread each function is called on. A variable that is accessed from a function that is called on the sim thread should belong to the sim and same story for the event queue thread. These should be mutually exclusive but they most likely aren't fully. In cases where a variable is accessed by both threads, care needs to be taken to fix that. First determine which thread the variable should really belong to, then work out how to get rid of the uses of the variable from the other thread. It's really a case-by-case basis, but some things that might work are:

- Create two versions of the variable, one for each thread. When the sim thread sets the variable, immediately update the one owned by the sim thread, and create an event to update the one owned by the event queue thread.#
- If the sim doesn't need its own copy but is directly accessing a variable that should be owned by the event queue, make the sim thread use events instead of directly accessing the variable

Most likely the worst offender is in the sensors. It is using mutexes to allow access from any thread instead of requiring the sim to put events on the queue. This is problematic if the sim and event queue are both updating these variables, even with a mutex, because the values they are setting reflect the state at different times, since the sim is generally a bit ahead of the event queue. This can lead to a jitter effect which fluctuates depending on how far behind the event queue thread is at any time.

Notes from follow up discussion with @gaut

- the event queue thread activity is only from events on the queue, which are all `lambda` functions. so we need to go through these lambda functions, see what members and methods they access and give them event queue names.
- the event queue thread has a `mutex` to protect its state from access from multiple threads. Generally there is just one mutex to support a class instance's state, but we can have multiple mutexes dedicated to different parts of the state. Name them accordingly.
- The optix sensors also have a `mutex` to protect parts from the sim and the event queue. We may want to see if there is a way to get rid of the mutex, by having variables dedicated to each thread, and events to sync them up. Currently both threads can touch the same variable, and this can be a source of jitter since one thread lags the other.
- Need to think about the lifetime of objects, especially if they are being accessed by both threads. Use `sync_` to avoid lingering objects.\_

#### 18.2.1.2.3.1. Sensor pipeline

Each sensor has an `ActionGraph` and a `FilterGraph` which are customizable. The pipeline structure invoked by the `DOptixSensor::update()` method is as follows:

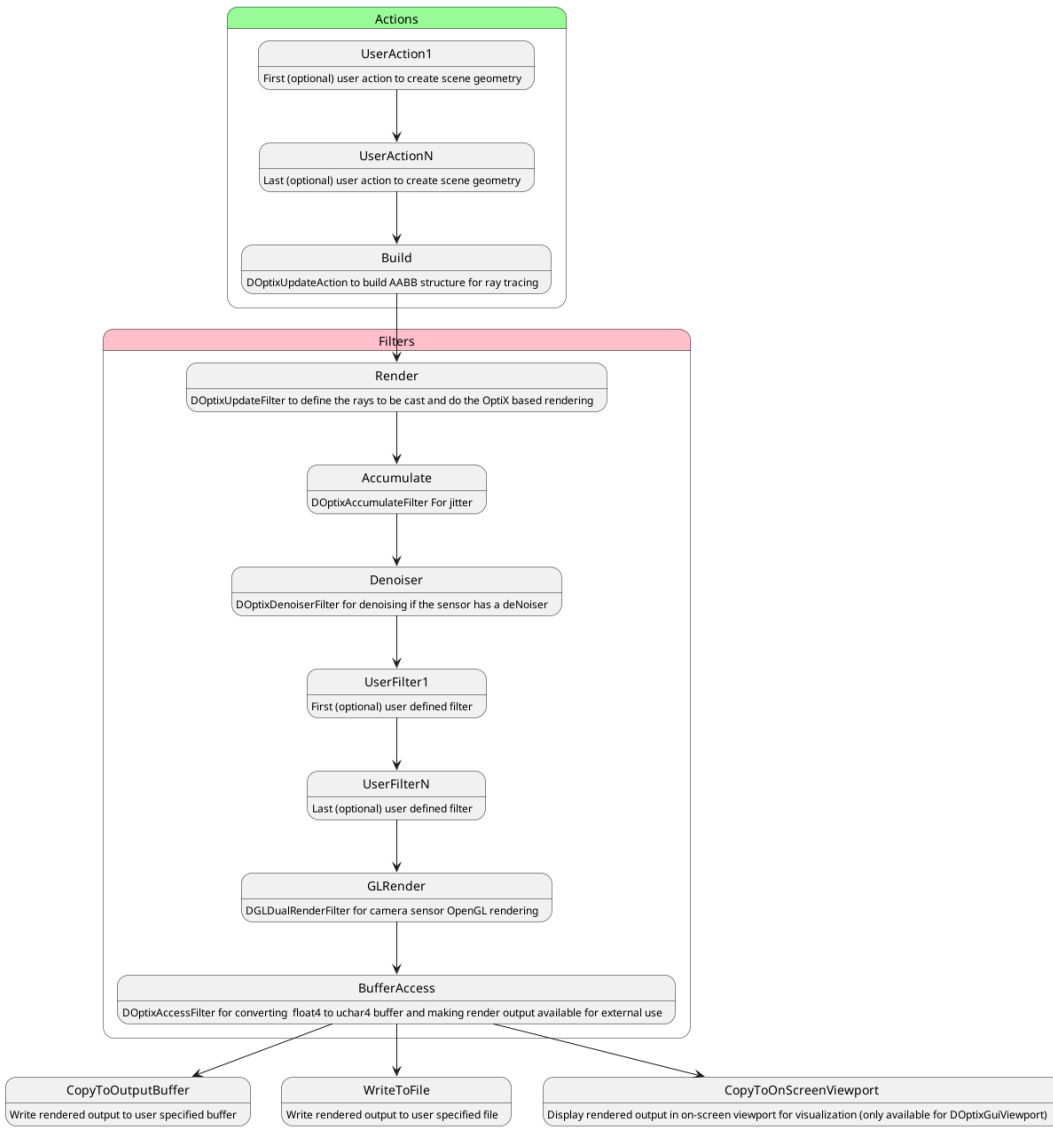
- user defined sensor specific `actions` that update the scene geometries, eg. clipmapping, band-mode materials, ornamental geometry etc
- a `DOptixUpdateAction` (last action, before any filters) which goes with the Builder. This step creates the optimized AABB data for use during ray tracing based rendering.
- a `DOptixUpdateFilter` which is the first filter which goes with a Pipeline and is responsible for the `RayGen` step to define the sensor specific rays to be cast, and the subsequent invocation of OptiX rendering to generate output buffers. This step uses CUDA shaders (eg. phong materials, CAHVORE lens) defined in `cuda/shaders` to define the rays and their interaction with the geometry. Its output is a rendered buffer.
- a `DOptixAccumulateFilter` in case of jitter
- a `DOptixDenoiserFilter` after the accumulate filter
- user defined sensor specific `filters` that transform the rendered buffer in some way. Examples including adding noise, blur, vignetting etc. These filters also make use of CUDA shader kernels defined in `cuda/kernels/pixel_operations.cu` to transform the buffers in some way.
- for `camera` sensors only, a `DGLDualRenderFilter` for lines etc (camera here is being used presumably for visual viewport as well as camera sensor).
- a final `DOptixAccessFilter` for converting the buffer's `float4` values into `uchar4` values and providing access to the rendered buffer output.

For an on-screen viewport, the final buffer values are written into an on-screen `ImGui` widget for 3D visualization.

Pipeline structure for a single sensor

[OptixScene-old\_sensor\_viewports] [OptixScene-new\_sensor\_viewports]

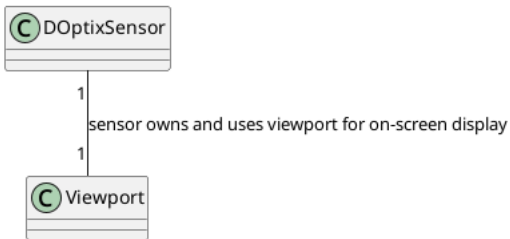
### Single Sensor Pipeline Flow



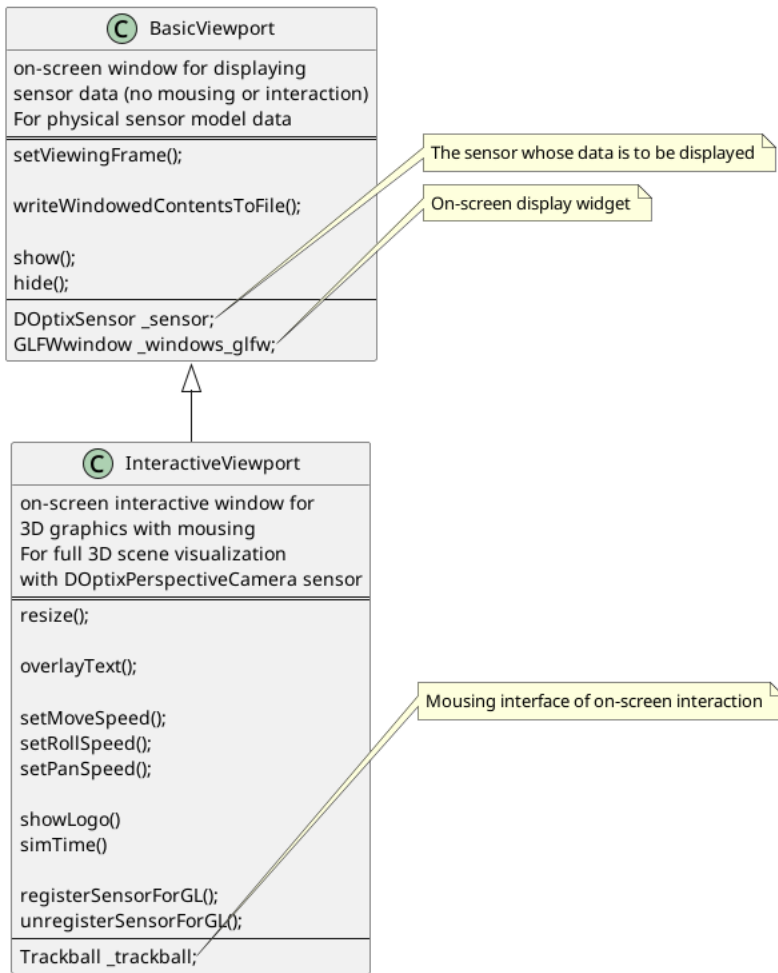
#### 18.2.2. Proposed restructuring of sensor/viewport classes

The following figure describes the proposed reorganization of the on-screen display classes.

#### Relationship between sensor and viewport classes



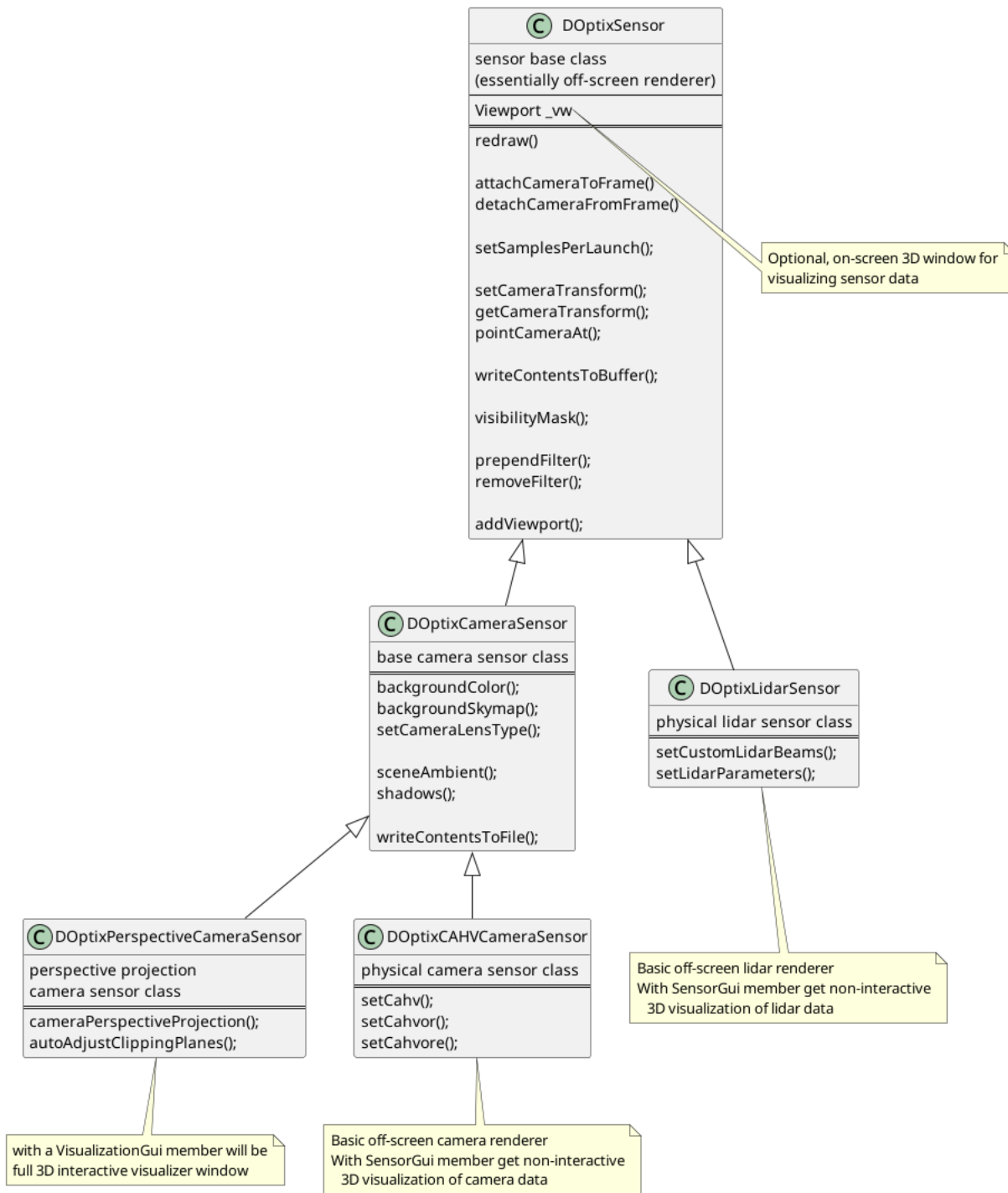
### Proposed On-Screen Viewport classes



The following figure describes the proposed reorganization of the sensor classes.



## Proposed Sensor classes



### 18.2.3. Renderables

#### 18.2.3.1. Current Design



TBD: Add diagram

#### 18.2.3.1.1. Primary choices

- One massive DOptixRenderable class
- Geometry and materials are handled as class variables

#### 18.2.3.1.2. Cleanup

Information that needs to be cleaned up may included data accessible only as a device pointer from another device pointer: For example, a `TextureLayer*` may have the only pointer to a `cudaTextureObject_t`. In these cases, the convention is to copy the `TextureLayer*` back to the host, free the `cudaTextureObject_t`, then free the host `TextureLayer*`, then free the device `TextureLayer*`



TBD: Add diagram showing copy from device to host and freeing operations

### 18.2.3.1.3. Changing Materials

When materials need to be modified, the old materials are first cleaned up (deleted). The new material is then used. However, mesh geometry holds a material id, which can be greater than 0, meaning a mesh can have multiple materials. When a mult-material mesh has its material programmatically replaced, this replaces multiple materials with a single material. In this case, the mesh is reloaded without materials (to force all material ids to be 0 and to prevent material from being loaded again), then the new material is set.



TBD: Add diagram showing the relationship between geometry and materials

### 18.2.3.2. Proposed Design Changes

#### Intended Design Principles

- Class that holds scene transform (pos, rotation, scale) can share geometry
- Instance meshes where possible
- Instance materials where possible
- Use inheritance to reduce complexity of `DOptixRenderable`

### 18.2.4. Sensor Frames

Here we cover the various aspects related to rendering poses associated with `sensors` and `viewports` in `OptixScene`.

#### Glossary:

- `PS` - physical sensor
- `VP` - viewing pose
- `SV` - static viewport
- `IV` - interactive viewport
- `VIV` - visualization `IV`

#### 18.2.4.1. Rendering frame types

There are two distinct renderings associated with each sensor/viewport pair combo. The first is for **raytracing** using `optix`, and the second is for **OpenGL** based painting on a displayed viewport for the sensor. The renderings require pose info which have to be mutually consistent.

- the **raytracing** render is carried out by `optix` and is determined by the camera/lidar characteristics. The pose used for this render is the pose of the `physical sensor (PS)`.
- the **OpenGL** render is for a viewport. Such viewports using a `viewing pose (VP)` to carry out the visualization render.
  - If the viewport does not have a mouse or trackball for user interaction, then it is a `static viewport (SV)` whose `VP` pose is locked to the `PS` pose. Use of `SV` is recommended - and the default - for visualizing `PS` output data.
  - When the viewport does have mousing support, then it is an `interactive viewport (IV)` whose `VP` pose can also be manipulated by the user via the mouse.

We can have the following three situations regarding `PS` and `SVCS`:

1. The poses of the `PS` and `VP` are often the same for consistency - but need not be. For instance, for lidars, we generate the point cloud using `optix` and the `PS` pose. However, we may want to visualize the point cloud from different vantage point in the sensor's `IV` viewport, and hence may want to be able to mouse around and change the pose of the viewport's `VP`. So the two poses can diverge from each other.
2. However, for physical **camera** sensor models that have viewports, the render from the ray tracing pass is also used for the viewport display, and so there is no separate `VP`. For this case the `PS` and `VP` are the same, and as such the same pose is used for both the raytracing and OpenGL renders. At some level `IV` interactive viewports are **incompatible** with physical camera `PS` since we have two separate and conflicting sources determining the `VP` pose for this case.
3. While the above discussion has focused on physical sensors that may or may not create visualization viewports, for **3D graphics visualization** the situation is the reverse and the viewport instance is in the driver's seat. For this case, the users can directly create one or more `visualization interactive viewports (VIV)` that have surrogate `perspective camera sensors` dedicated to 3D visualization. Such `VIV` use their surrogate perspective camera sensors for all rendering. For this case, the `VP/PS` pose is defined by user interaction via the mouse, or by direct calls to methods such as `viewAroundFrame`, `setCameraTransform` or `pointCameraAt` that can programmatically set convenient viewing poses or by setting up **chase** views where the viewing follows specified frames. Unlike situation (2) above, in this case there is no ambiguity or conflict about the correct value of the `VP` pose.

#### 18.2.4.2. Sensor pose determination

Sensors are scene objects, which may be attached to a scene frame, or be **rooted** and not have a parent scene frame (in which case their pose is defined wrt to the origin scene frame). Their full pose wrt to the origin frame is thus a composition of the the parent scene frame's pose wrt to the origin scene frame (if it exists) and the `relTransform` of the sensor itself.

- For a physical sensor, the `PS` is attached to the mount node's scene frame, and its `relTransform` is often constant (eg. defined by a camera CAHVORE file), or more generally managed by the simulation. The simulation also manages the parent scene frame's pose (eg. from the motion of the vehicle). This applies to all physical camera and lidar sensors. The `PS` pose for a physical sensor is thus determined and set by the simulation.
- A viewport's `VP` viewing pose is identical to the `PS` pose for a `SV` static viewport for a physical viewport.
- An `IV` interactive viewport's `VP` viewing pose can be set via a mouse or programmatically using `viewAroundFrame` etc methods. \*This can lead to a conflicting situation when the physical sensor is a camera.\*
- Visualization `VIV` viewports make use of surrogate camera sensors whose `VP` pose can be set interactively by `mousing`, or by calling methods or by enabling chase views by attaching them to scene frames.

#### 18.2.4.3. Sensor default frames

Each sensor will define its default orientation. For example, if a camera is attached to a frame with no orientation, which direction is it facing?

- Cameras: The OptixScene convention is to have cameras pointing toward Z, with Y-up and X-left to complete the right-handed frame. However, CAVHORE's assumption is that -z will be forward when considering the parameters, so we flip the CAVHORE model's z axis to fit with the other camera model conventions.
- Lidar:
- SRP: does not fit into frame convention. It will always point from the sun light to the object origin, regardless of frame orientation.

#### 18.2.4.4. Actions

- `setCameraPose`, `pointCameraAt` methods are only part of the viewport classes, and cannot be set on a sensor.
- Viewing frame is part of the sensor class, as the sensor can optionally use this as an offset for rendering or for visualizing its own data.
- `getCameraTransform()` for a camera now returns its full transform including the view offset for completeness.
- The `setCameraTransform`, `pointCameraAt` etc methods from the sensor class to the interactive viewport class. The sensor's pose should come entirely from the `relTransform` values, and these methods should only be used to set the `VP` pose for viewing in a viewport.
- mousing interactions set the view frame for the sensor, and the sensor decides how best to use these. For example, camera will include these in the render position, while a lidar will ignore these for data generation, and will instead use it only for displaying the position of the points so that the user can move around the point and understand what is seen.

## 18.3. Usage

### 18.3.1. Assigning materials to partGeometries

After creating a `partGeometry` (in the example `partGeom`) there are several ways to change its appearance. If we are creating a primitive or importing a file with the flag `override_material` set to True, a basic Phong material with white diffuse color will be attached to the object. To override it we can create a new one and attach it to the object:

```
material = PhongMaterial()
material.diffuse_color = Color(0.800000, 0.000000, 0.300000)
material.ambient_color = Color(0.200000, 0.000000, 0.050000)
material.specular_color = Color(0.000000, 1.000000, 0.000000)
material.emissive_color = Color(0.000000, 0.000000, 0.400000)
material.shininess = 1.2
partGeom.setPhongMaterial(material)
```

or retrieve the existing one, modifying the field we want to change, and re attach it:

```
material = partGeom.getPhongMaterial()
material.emissive_color = Color(0.000000, 0.000000, 0.400000)
partGeom.setPhongMaterial(material)
```

`PhongMaterial` is a standard lighting model (not PBR) that is supported by all our client scenes.

However, there are other two lighting models, `HapkeMaterial` and `PrincipledMaterials`, that are more complex and supported only by `BlenderScene` and `OptixScene`. The Hapke lighting model is used to describe a regolith body, while the Principled lighting model is a shader created by Walt Disney which can describe many different types of materials by just changing its parameters.

To assign those two materials to a part geometry:

```
material1 = HapkeMaterial()
material1.albedo = Color(0.8, 0.3, 0.5)
material1.normal_map_texture = "../assets/foil_n.jpg"
partGeom1.setHapkeMaterial(material1)

material2 = PrincipledMaterial()
material2.base_color = Color(0.8, 0.3, 0.5)
material2.normal_map_texture = "../assets/foil_n.jpg"
partGeom2.setPrincipledMaterial(material2)
```

For more details on the `PhongMaterial`, `HapkeMaterial`, and `PrincipledMaterials` look at `DScene/Material.h`.

### 18.3.2. Assigning materials to partGeometries

`TopoGeometries` have also `setPhongMaterial` and `getPhongMaterials` available for all the client scenes, and `setHapkeMaterial` and `getHapkeMaterial` available in `BlenderScene` and `OptixScene`, while there has been no need for `PrincipledMaterial`.

### 18.3.3. Advanced topics

#### Texture tiling

`OptixScene` supports passing extra parameters to `topoGeometries` to tile the texture, and in the future also `partGeometries` will support this.

*Texture tiling* is the process of placing multiple copies of a high resolution texture image over the terrain surface in a way that there are no seams between neighboring tiles. The advantage of tiling textures is that a relatively small texture can be applied to a large surface by repeating (or "tiling") that texture over the surface. In this way, we can achieve higher resolution for the terrain texture, without the need to store expensive high-resolution textures.

The extra parameters are two integer values, `texture_tile_x` and `texture_tile_y`, to specify how many textures needs to be aligned along the X (width) and Y (height) directions. To achieve better results and to avoid stretched textures `texture_tile_x` and `texture_tile_y` should be the equal.

Example:

```

extras = {
  "use_clipmaps": 0,
  "max_coarse_levels": 14,
  "clipmap_exp": 6,
  "texture": "../../assets/texture_test.png",
  "texture_tile_x": 3,
  "texture_tile_y": 3,
}
topo = scene.createTopoGeometry(leveler_dem, SceneObject.PHYSICAL | SceneObject.ORNAMENTAL, extras, "topo")

```

### 18.3.3.1. Blending two textures together

OptixScene supports a really common feature in 3D software which consists in blending two textures.

This is based on the lerp function, which is basically a linear interpolation between the two textures based on a factor (FAC, a float between 0 and 1 where 0 means 100% texture A and 1 means 100% texture B). This FAC factor can also be a texture, (usually grayscale) so we can blend two textures adding variety across the mesh/topo.

This feature is available only when using PhongMaterial:

```

material = PhongMaterial()
material.diffuse_map = "../../assets/grass_3K.jpg"
material.diffuse_map2 = "../../assets/rock_3K.jpg"
material.FAC_map = "../../assets/perlin_noise.png"
# OR:
# material.FAC = 0.5
topo.setPhongMaterial(material)

```


If you specify only `diffuse_map`, it will apply that texture map as usual, but if you specify also `diffuse_map2` it will linearly interpolate between the two based on the FAC or the FAC\_map.



similar to what happens with `diffuse_map` and `diffuse_color`, if `FAC_map` has been specified `FAC` will be ignored.

Output example:

Given this texture: 

and this one: 

we can blend them using FAC = 0.5 (for example), which results in: 

Or using this texture as FAC\_map: 

which results in: 

## 18.4. Software

### 18.4.1. Event queue implementation

TBD: taken from Aaron Gaut's email of 11/6/2021. Needs to be cleaned up.

- why are we storing the future instance in the event queue along with the callback when we can always get the future from the packaged event via `get_future0`?



I'm not sure exactly, but if I remember right, were are storing a `shared_future`, not just a future. The c++ docs say a `shared_future` can be copied (the same way a `shared_ptr` can be, while an ordinary future can only be moved like a `unique_ptr`). Also a `shared_future` can be waited on from multiple threads at once. If you're going to use a `shared_future` I think you would want to do it early, because if you're going to use the `shared_future` from multiple places, they need to all be using the same `shared_future` or copies of the same `shared_future`. I'm not sure where we need the capabilities of a `shared_future` in the design though. I'd look for places we are copying it or potentially waiting on it from multiple threads. Or maybe we aren't using those capabilities right now but the designer wanted to allow for that possibility in the future.

- `addEvent0` currently does not return anything? why does it not return the future instance? Do we never have a need to get back the returned value?



I believe that's what `addReturnEvent` is for. Generally in the sim thread if we are getting something back from the event queue we don't want a future - we want the value that will be stored in the future, because the sim thread isn't really set up to go do other work while it's waiting for the future to complete. So the event queue doesn't really expose futures to the sim thread. Instead, there is an `addReturnEvent` call that will block the sim thread until the future is completed. But the sim thread only needs to block if it needs to get a value back from the event queue thread. So blocking until the future is complete for every single event would take way longer than it needs to. That's why `addEvent` is a separate method that doesn't care about the return value - it prevents the sim thread from needing to block on events where there is no return value.

- why does the packaged event signature such as `to` not return anything instead of at least a status flag or something?



I think it returns a `std::any, no?`

- why does the dispatch loop not move all pending events to a temp list and unlock the queue right away and then process the events on the temp queue? Currently we are unlocking, processing one event, locking it, popping the processed event and going back to the top of the loop. The former would seem to be more efficient. **THIS DOES NOT WORK WELL - mousing becomes choppy and sluggish if we try and do this.** Note that since events can themselves post new events on the queue, so it is necessary that the mutex for the event queue not be locked while events are being processed.



Sounds like a reasonable optimization to me

- why does the event queue allow for multiple threads? Is it for the case when the order of processing items is not critical and the event processing can be parallelized?



I think you're right that it would require items that don't need to be processed in order. I'm not sure what the plan was with allowing a different number of threads.

## 18.4.2. Upgrading Optix and CUDA

### 18.4.2.1. Upgraded to OptiX 7.7 with CUDA 12.1

Date: July, 2023 OS: Fedora 38

CUDA: 12.1

OptiX: 7.7

Issues, fixes, and notes:

- Removed `-allow-unsupported-compiler` and forced use of gcc 12 since this flag relies on undefined behavior
- Change `optixModuleCreateFromPTX()` to the generic `optixModuleCreate()`
- explicitly set `OPTIX_DENOISER_ALPHA_MODE_COPY` for the denoiser
- pass the optix pipeline into `optixProgramGroupGetStackSize()`

### 18.4.2.2. Upgraded to OptiX 7.4 with CUDA 11.5

Date: October 23, 2021 OS: Fedora 34

CUDA: 11.5

OptiX: 7.4

Issues, fixes, and notes:

- Had to add `-allow-unsupported-compiler` to get nvcc to play with gcc 11
- Also had to add `-compiler-options -std=c++14` option to nvcc to build properly
- Eventually had to do a source build of gcc 10.2 to give to nvcc for things to work fully
- Eventually updating to gcc 11.2 fixed the bug introduced in gcc 10.4 so we could go back to nvcc using the default gcc installation.
- Also needed to use a source build of glfw since the dnf package install seems to be specific to Wayland

## 18.5. Raw documentation



TBD: Need scrubbing before integration.

### 18.5.1. OptixScene: Stop using scene graph approach for updating transforms



TBD: Needs scrubbing. Notes brought over from [issue](https://dartsgitlab-internal.jpl.nasa.gov/darts-lab/jpl-internal/all-access/development/rendering/optixscene/-/issues/93) (https://dartsgitlab-internal.jpl.nasa.gov/darts-lab/jpl-internal/all-access/development/rendering/optixscene/-/issues/93).

We have been carrying over the Dspace approach - based on a scene graph paradigm - of setting relative transform values, and leaving it to the client scene to update the absolute poses of all the objects. This works fine for Ogre and other OpenGL approaches where the accumulation of transforms is done fast on the GPU. However, `optix` does not do this - and does not have the notion of a scene graph. Thus, all objects have to be a position absolutely before invoking ray tracing functions.

This is being done currently via lots of `originTransform()` calls. This is however very inefficient, since there is no caching being done, and each of these calls triggers a full recursion across the rel transform along the path from the origin scene frame. We need to shift away from the scene graph paradigm and to one more suited to the way optix works. Some options are

- Make all objects rooted. This way the `relTransform` is the `originTransform` for every object, and no additional recursions are needed. Very simple, but the rootedness will have to be done at the `FacadeScene` level - and would need to be done for any new objects that are added.
- Disable the `originTransform` method in `DOptixSceneFrame`. Instead populate the `DOptixScene::updateSceneTransforms` method to compute all the origin frame transforms in a single sweep and store them in each `DOptixSceneFrame` instance and also push the value on a queued version. The `DOptixSceneObject` and scene frames then simply look up this value and use it instead of recomputing it.

These options would be a big improvement over the current approach.

#### 18.5.1.1. Shader implementation



TBD: Overview of shaders

We are modeling the lighting equation:

$$L_o(x, \omega_o) = \int_{S^2} f(x, \omega_o, \omega_i) L_i(x, \omega_i) |\cos\theta_i| d\omega_i$$



For path tracing, sampling is done based on the roughness of the surface, following GGX sampling.

For Whitted ray tracing, the "sampled" direction is the mirror direction. The pdf is chosen to be  $\frac{1}{2\sqrt{1-\rho^2}}$  so that a brdf associated with the following parameters produces half a mirror reflection for a viewing direction  $(.707, 0.0, -.707)$ .

```
albedo = (1,1,1)
metallic = .0
subsurface = 0.0
specular = 0.5
roughness = 0.0
specular_tint = 0.0
anisotropic = 0.0
sheen = 0.0
sheen_tint = 0.5
clearcoat = 0.0
clearcoat_roughness = 0.03
```

#### 18.5.1.1.2.1.2. Phong BRDF

```
void __direct_callable__sample_brdf_phong(const MaterialParameter& material,
                                         const Ray& ray,
                                         const PrimitiveData& primitive_data,
                                         const float2& uv,
                                         const PackedLightData& light_data,
                                         const GeneralPRD& wrapped_prd,
                                         const bool& tracing_lights)
{
}
```

The default Phong model is defined as

The default Phong model is not a valid brdf as a diffuse color of (1,1,1) and specular color of (1,1,1) would not be energy conserving. This introduces a problem for path tracing, and creates an inconsistency between Phong and Principled materials, with Phong object appearing brighter in a given scene than Principled objects. To alleviate this, we modify the phong model to be a valid brdf. To do this, we normalize the diffuse model by dividing by PI. When path tracing, we also normalize the specular model following [https://graphics.geometrian.com/research/normalize\\_phong.html](https://graphics.geometrian.com/research/normalize_phong.html). We could normalize by default and then use a similar reflection heuristic for Whitted ray tacing. However, to simplify, we normalize only for path tracing and leave the specular model as is for Whitted ray tracing, as it already produces values [0-1] which is what the heuristic would do anyway.

For Whitted ray tracing, we additionally ignore the diffuse component, as the mirror reflection model is by definition a model only of the specular reflection. When determining the specular reflection contribution for the mirror reflection, we use the specular model with  $\text{adjusted\_Ks} = \text{phong.Kr} * \text{phong.Ns} * \text{phong.Ks}$ ; . This ensures the user can still control the reflection with Kr. However, it should be Ks (the specular term) which determines reflection as it does with the highlights of the lights. Ns is the shininess of the material and when it is zero, path tracing produces a perfectly diffuse reflection. Therefore, we use Ns in part to determine, as a heuristic, how much mirror reflection should be produced.

Lastly, to prevent the issues caused by Kd and Ks and Ks, we normalize Kd and Ks by the sum of Kd and Ks means:

```
float Ks_weight = mean(phong.Ks) / mean(Kd + phong.Ks);
float Kd_weight = mean(Kd) / mean(Kd + phong.Ks);
```

This makes it such that when Kd and Ks = (1,1,1), we say the material is equally diffuse and specular; absorbing no light (half is diffusely reflected, have is specularly reflected). When Kd = (1,1,1) and Ks = (0,0,0), the object is perfectly diffuse. When Kd = (0,0,0), Ks = (1,1,1), Kr = (1,1,1), and Ns = 1.0, the object will be perfectly reflective for both path tracing and Whitted ray tracing.

**Note:** this makes the Dspace and OptixScene implementations inconsistent, as the OptixScene environment will be darker than Dspace and will need higher intensity lights to achieve the same scene brightness.

#### 18.5.1.1.2.1.3. Hapke BRDF

```
void __direct_callable__sample_brdf_hapke(const MaterialParameter& material,
                                         const Ray& ray,
                                         const PrimitiveData& primitive_data,
                                         const float2& uv,
                                         const PackedLightData& light_data,
                                         const GeneralPRD& wrapped_prd,
                                         const bool& tracing_lights)
{
}
```



TBD: missing Hapke implementation

#### 18.5.1.1.2.2. Light callables

Light callables are responsible for calculating the amount of light that would hit a given location. These callables follow the signature:

```
void __direct_callable__<light name>(const LightParameters& light, const float3& hit_point, LightSample& light_sample)
{
    // float3 light_contribution;
    // float dist_to_light;
    // float3 dir_to_light;
}
```

The light function is required to populate the following struct and, if necessary, should take into account attenuation due to distance, the intensity and color of the light, and the angular falloff. The lights are not responsible for account for shadow attenuation/transmission or shading based on viewing and normal vectors.

```
struct LightSample
{
    float3 light_contribution; //how much light would hit the tested point
    float dist_to_light;      // distance to the light
    float3 dir_to_light;      // the direction to the light
};
```

Point light contribution:

$$L_i(x, \omega_i) = \frac{L_0}{a + bd + cd^2} d; \text{ distance from hit point } (x) \text{ to the light location } L_0: \text{ product of the intensity and the light color (separate only due to convention in Dscene)} a, b, c: \text{ falloff coefficients}$$

Spot light contribution:

$$L_i(x, \omega_i) = \begin{cases} \frac{L_0}{a + bd + cd^2} & \cos^{-1}(\omega_i \cdot \omega_L) < \alpha_0 \\ \frac{L_0}{a + bd + cd^2} \frac{\omega_i \cdot \omega_L}{\cos(\alpha_0) - \cos(\alpha_1)} & \alpha_0 \leq \cos^{-1}(\omega_i \cdot \omega_L) < \alpha_1 \\ 0 & \alpha_1 < \cos^{-1}(\omega_i \cdot \omega_L) \end{cases} d: \text{ distance from hit point } (x) \text{ to the light location } L_0: \text{ product of the intensity and the light color (separate only due to convention in Dscene)} a, b, c: \text{ falloff coefficients}$$

Directional light contribution:

$$L_i(x, \omega_i) = L_0 L_0: \text{ product of the intensity and the light color (separate only due to convention in Dscene)}$$

#### 18.5.1.1.2.3. Sampling strategies



TBD: cosine hemisphere sampling



TBD: phong sampling



TBD: ggx sampling

#### 18.5.1.1.3. Normalization of Phong model for use with PBR and path tracing



TBD: missing

Not currently implemented as it would break consistency with DspaceScene

#### 18.5.1.2. Adding a new sensor



TBD: What is takes to make a new sensor

#### 18.5.1.2.1. DScene API



TBD: what needs to be exposed in the DScene API?

#### 18.5.1.2.2. OptixScene API



TBD: what needs to be exposed in the DOptixScene API?

#### 18.5.1.2.3. Sensor and viewport classes



TBD: what needs to be added in terms of sensor classes/viewport changes?

#### 18.5.1.2.4. Filter Graph Implementation

The filter graph passes a `DOptixSensorBuffer` between filters. This is where data produced/consumed by the filters is stored. The filter graph is sensor agnostic, but individual filters may be sensor specific. To allow for this, `DOptixSensorBuffer` holds the dimensions of the buffer along with a `std::variant`` data field. Individual sensors are responsible for defining a set of buffer types that need to be added to this data variant. Currently, the data variants are:



```
std::variant<CameraDeviceBuffers,
            CameraHostBuffers,
            CameraDeviceOutputBuffers,
            LidarDeviceBuffer,
            LidarHostBuffer,
            SRPDeviceBuffers,
            SRPHostBuffers>
data;
```

Any filters that can be applied to the new sensor but be modified to conditionally get the new sensor's relevant data variant. For example, `DOptixUpdateFilter` switches on `_sensor.getSensorFamily()` to define an update for each sensor class. When adding a new sensor, add the switch condition needed throughout `DOptixUpdateFilter.cc`. Similarly, add the new sensor's case to `DOptixFilterGraph.cc` and `DOptixAccumulateFilter.cc` (which can be applied to any/all sensor classes).



TBD: what needs to be added and modified in the filter graph?

#### 18.5.1.2.5. Material classes

In order to interact with SRP, materials must define an SRP direct callable index. Along with an index, a direct callable for SRP material shading also needs to be defined. It is **CRITICAL** that the function index matches the order of direct callable functions created in `DOptixEngine::_create_DC_buffersOnQueue()`



TBD: what needs to be added in terms of materials/renderables?

#### 18.5.1.2.6. GPU implementation



TBD: what needs to be added and modified in the shader code?

### 18.5.2. Cleanup and Redesign of OptixScene 2023

#### 18.5.2.1. Primary Design Principles

##### 18.5.2.1.1. Simulation time

The simulation should interface with the rendering scene in a way that allows viewports/sensor to use sim-time to complete a redraw, even if that data is required by a specific time in simulation. For visualization viewports, these should be given the frame period before syncing the render. For sensors, they should be parameterized by an allowable sim-lag-time which defines how much sim time can elapse before the sim should wait and sync the sensor's data. In some cases this could model the real sensor lag. In other cases, this would be for computational efficiency only when the processing of the sensor data would allow this. The sensor can still stamp the data/image with the actual sim time, even when returning the data at the lag time.

##### 18.5.2.1.2. Sensor/scene data access should be enforced at the language level

Calls to change information in the scene or sensor should not be able to modify data that is in-use. Additionally, even when those calls are allowed to modify such data, the data should automatically be marked dirty rather than relying on all developers to know what information is sensitive.



TBD: still need a way to make this happen

##### 18.5.2.1.3. CPU/GPU threading

All GPU functions should be called on a non-default thread, which allows async memcopy and the use of pinned memory for faster data movement. GPU functions that belong to a set of functions that have sequential dependence on the same data should be on the same thread and same CUDA stream. For example, the ray launch, accumulate, RGBA access, and GL draw should be on the same CPU thread, using the same non-default CUDA stream with pinned memory for the buffer that is copied back for OpenGL.

#### 18.5.2.2. Proposed Design

##### 18.5.2.2.1. Sim-render interface and syncing

##### 18.5.2.2.2. Queuing and thread/data access

##### 18.5.2.2.3. Optix rebuilding and launching

## 19. DMeshScene

### 19.1. Background

#### 19.1.1. Reference & Source material

- [DMeshScene Doxygen docs](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/DMeshScene/html/) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/DMeshScene/html/)
- [Team talks](https://dlabportal.jpl.nasa.gov/resources/darts-lab-talks/) (https://dlabportal.jpl.nasa.gov/resources/darts-lab-talks/)
- [2019, 2020 course slides](https://dlabportal.jpl.nasa.gov/documentation/coursetutorial-links/) (https://dlabportal.jpl.nasa.gov/documentation/coursetutorial-links/)
- [Sphinx docs](https://dartslab.jpl.nasa.gov/dlabdocs/) (https://dartslab.jpl.nasa.gov/dlabdocs/)
- [Jupyter notebooks](https://dlabnotebooks.jpl.nasa.gov/hub/login) (https://dlabnotebooks.jpl.nasa.gov/hub/login)
- [DARTS Lab QA site](https://dartslab.jpl.nasa.gov/qa/) (https://dartslab.jpl.nasa.gov/qa/)

### 19.2. Design

### 19.3. Usage

### 19.4. Software

### 19.5. Raw documentation



TBD: Need scrubbing before integration.

#### 19.5.1. DMeshScene: DMesh collision detection capability using embree's rtcCollide method



TBD: Needs scrubbing. Notes brought over from [issue](https://dartsgitlab-internal.jpl.nasa.gov/darts-lab/jpl-internal/all-access/development/scene-geometry/dmesh/-issues/36) (https://dartsgitlab-internal.jpl.nasa.gov/darts-lab/jpl-internal/all-access/development/scene-geometry/dmesh/-issues/36).

It is seemingly possible to use embree's [rtcCollide](https://spec.oneapi.io/oneart/latest/embree-spec.html#rtccollide) method to do broad phase collision detection. It also has option to register a callback for fine grain work that the user has to provide. We should explore using this for implementing our own collision detection method. There is an example of this in the PDF documentation at <https://raw.githubusercontent.com/embree/embree/master/readme.pdf> (look for the Collision Detection section at the end and the source code at [https://github.com/embree/embree/blob/master/tutorials/collide/collide\\_device.cpp](https://github.com/embree/embree/blob/master/tutorials/collide/collide_device.cpp)).

Outstanding issues:

- Ensure that convex decomposition is used when creating the mesh geometries for a `DMeshSceneObject` since the near `libccd` phase depends on the meshes being convex. This would require the `DMeshPartGeometry::load` method to carry out inline convex-decomposition when loading from an `obj` file. (see [issue](https://dartsgitlab-internal.jpl.nasa.gov/darts-lab/jpl-internal/all-access/development/scene-geometry/dmeshscene/-issues/11) (https://dartsgitlab-internal.jpl.nasa.gov/darts-lab/jpl-internal/all-access/development/scene-geometry/dmeshscene/-issues/11)).
  - filtering support at the mesh level? there may be something at the `embree` level to enable filtering. Need to research. Alternatively, cull after the results of the broad phase results. Should be able to use `detachGeometry` - but it is not working yet.
  - Expand map to find parent scene object.
- Assuming we go this route, we may also want to add support for doing the convex decomposition off-line, and passing in the results to the part geometry to use and skip the convex decomposition step. (see #52)
  - Maybe use `DMeshObject` route to load stored convex decomposition files. Possibly store as a `obj` file.
- Do we need to change `DMeshSceneObject` to therefore have a list of convex `Mesh` instances instead of a single member like we do currently? If we do this, we need to make sure to add filtering so we are not doing collision detection between meshes belonging to the same scene object.
- Implement collision filtering in `MeshGroup`. This will be necessary for convex decompositions of meshes; we don't want to detect collisions between meshes belonging to the same scene object.
  - `SubMesh` instances should be acceptable.
- Assuming small penetrations so far. Do we need to add a check for this?
  - `libccd` will detect penetration for large values, but results may not be reliable. However, we expect to be dealing with small penetrations virtually all the time. Its only during initialization that we may have a bad state with large penetrations. At this point we only want to know whether there are collisions or not (since we want a collision free initial state), and do not really care about accuracy.
- Does `libccd` always return a legal vertex? Does `libccd` return a position of a vertex index? if position, then do we need to have a fast way to look up the vertex index?
  - `libccd` returns the mid point between the contact points on the two bodies. currently we go half way on both sides to get the contact point on the geometries. then we do a `pointQuery` (`computeClosestVertex` call) to get the closest vertex. Should we also be checking for faces? `libccd` seems to work with vertices - not faces.
- Can we skip search for contact points early by checking normals to detect if we have a potential line or face contact? See issue #53 for details
  - if we can find faces involved with the contact vertex then it can be used to possibly skip plane/plane contact point detection using normals and separation direction checks. However, still need to do something about edge/plane intersections.
- Add support for analytical collisions between basic shapes (spheres, cubes, etc.). This will greatly improve speed and accuracy of collisions between these objects.
  - Will need to create different `narrow-phase` colliders for sphere-sphere, sphere-mesh, sphere-box, box-box, box-mesh, etc. collisions. See [issue](https://dartsgitlab-internal.jpl.nasa.gov/darts-lab/jpl-internal/all-access/development/scene-geometry/dmeshscene/-issues/2) (https://dartsgitlab-internal.jpl.nasa.gov/darts-lab/jpl-internal/all-access/development/scene-geometry/dmeshscene/-issues/2)
  - Add enum type for known geometries. Look through the pairs and invoke the specialized colliders as needed before falling `libccd` when we need to.

- Replace `DMesh/ContactPoint` class with `DScene/CollisionInfo` class. All we need to do is to work with `DMeshSceneObject` instances instead of `Mesh` instances. Should the methods that use it be moved to `DMeshScene`? See [issue](https://dartsgitlab-internal.jpl.nasa.gov/darts-lab/jpl-internal/all-access/development/scene-geometry/dmeshscene/-issues/5) (https://dartsgitlab-internal.jpl.nasa.gov/darts-lab/jpl-internal/all-access/development/scene-geometry/dmeshscene/-issues/5)
  - perhaps derive a `MeshCollisionInfo` struct from `DScene/CollisionInfo` and have it hold mesh instances, so the only update needed it to look up the parent scene object ...
- Optimize `computeClosestVertex` per discussion below: (see issue #49)
  - just using floats seems to be fast as well
- Figure and fix Sam's implementation and use of `Mesh::_cur_transform`. See #54
- Move `Mesh::_cur_transform` member to `DMeshSceneObject` class
  - there is a benefit to keeping this in the mesh. We do not want to change the embree wrapper within each individual mesh. However we do want to update the collision scene where you can do things at the higher level.
  - we keep the internal embree world stale, since bounding box does not depend on the embree part
  - because of this, we should make the mesh objects be private to the collision world and not shared with others who may trip over the stale embree worlds.
- Move collision detection specific methods in `MeshUtilities` into static or protected methods of `MeshGroup` class.
- Add hooks to view frames at the contact points (eg. from `ContactGui`)
- Fix the mesh removal issue (`rtcDetachGeometry`) mentioned below at the end of the [comment](https://dartsgitlab-internal.jpl.nasa.gov/darts-lab/jpl-internal/all-access/development/scene-geometry/dmesh/-issues/36#note_8684) (https://dartsgitlab-internal.jpl.nasa.gov/darts-lab/jpl-internal/all-access/development/scene-geometry/dmesh/-issues/36#note\_8684).
- Fix handling of edge/edge intersections as discussed below at the end of the [comment](https://dartsgitlab-internal.jpl.nasa.gov/darts-lab/jpl-internal/all-access/development/scene-geometry/dmesh/-issues/36#note_8658) (https://dartsgitlab-internal.jpl.nasa.gov/darts-lab/jpl-internal/all-access/development/scene-geometry/dmesh/-issues/36#note\_8658)
- Change some of the static callback functions into `lambda` functions if possible. See [issue](https://dartsgitlab-internal.jpl.nasa.gov/darts-lab/jpl-internal/all-access/development/scene-geometry/dmeshscene/-issues/6) (https://dartsgitlab-internal.jpl.nasa.gov/darts-lab/jpl-internal/all-access/development/scene-geometry/dmeshscene/-issues/6)
- Add `DMeshTopoGeometry` class and use to support collision detection with terrains. Also, this class should be able to break up the overall geometry into a set of `DMesh` tiles (possibly a `MeshGroup`), so that the broad-phase BVH (and the later near phase) processing can work with the smaller number of faces and vertices on tiles rather than for the whole terrain. However, we need to ensure that each of these tiles is `convex`.
- Update `DMeshScene` to work with multi-link mechanisms. For this we need to address handling of non-scene-graph clients as discussed in this [issue](https://dartsgitlab-internal.jpl.nasa.gov/darts-lab/jpl-internal/all-access/development/scene-geometry/dscene/-issues/21) (https://dartsgitlab-internal.jpl.nasa.gov/darts-lab/jpl-internal/all-access/development/scene-geometry/dscene/-issues/21)
- See if we can use `openmp` to parallelize some of the loops.
  - process vertices in contact point generation for parallelization. Check if the number of vertices is large enough to be worth it.
  - In the support function, we might be able to parallelize over all the vertices.
- Do we still need to worry about duplicate vertices when determining contact points. Has this been taken care of the `mesh_only` argument to `AssimpImporter`?
  - Add a checker using the `igl::all_pairs_distances` method. Use this method and check for 0 distances.
- Convert `DMeshScene` requests to create a scene manager so that we can also create 3D visualization to go along with the collision detection. See [issue](https://dartsgitlab-internal.jpl.nasa.gov/darts-lab/jpl-internal/all-access/development/scene-geometry/dmeshscene/-issues/9) (https://dartsgitlab-internal.jpl.nasa.gov/darts-lab/jpl-internal/all-access/development/scene-geometry/dmeshscene/-issues/9)
  - Also make them generic do that they can be run via `dmesh`, `bullet` or `fcl` using a command line option. This will help with cross-validation
- Refactor the `std::vector<std::set<size_t>>` `Mesh::getAdjacentVertices(bool duplicate_verts)` method body since its current implementation requires `Eigen 3.4.0` due to the way it uses `auto/rowwise` in a for loop. We should add a version dependent alternate version so that the code compiles with `Eigen 3.3` as well.
- Switch the adjacent vertices computation method to use `libigl's adjacency_list` method
- Create more test cases to check out the various surface/surface, surface/edge, surface/point, edge/edge, edge/point intersection cases. See [issue](https://dartsgitlab-internal.jpl.nasa.gov/darts-lab/jpl-internal/all-access/development/scene-geometry/dmeshscene/-issues/8) (https://dartsgitlab-internal.jpl.nasa.gov/darts-lab/jpl-internal/all-access/development/scene-geometry/dmeshscene/-issues/8)
- Would there be any benefit to creating a single precision version of `Mesh` (i.e one where vertices are stored as floats instead of doubles)? **[Sam - just a note: embree currently uses single precision for ray tracing, and I don't believe there is a double precision build out there. I believe libccd also uses single precision.]**
  - probably does not matter. Just make sure callback functions work with floats.
  - perhaps add methods that work with floats instead of `SOARay` objects and use them in the callbacks (eg. in the support function)
- In contrast to a sub-item above, it looks like `libccd` is automatically built in double-precision mode. We will need to compile it in single-precision mode to avoid unnecessary calculations. See [here](https://github.com/danfis/libccd) (https://github.com/danfis/libccd)
  - In fact, remove all use of `SOA` objects in the collision detection algorithms. see [issue](https://dartsgitlab-internal.jpl.nasa.gov/darts-lab/jpl-internal/all-access/development/scene-geometry/dmeshscene/-issues/7) (https://dartsgitlab-internal.jpl.nasa.gov/darts-lab/jpl-internal/all-access/development/scene-geometry/dmeshscene/-issues/7)
- With the `assimp` importer now creating a `MeshGroup` for each part geometry (see [issue](https://dartsgitlab-internal.jpl.nasa.gov/darts-lab/jpl-internal/all-access/development/dynamics/hdart/-issues/5) (https://dartsgitlab-internal.jpl.nasa.gov/darts-lab/jpl-internal/all-access/development/dynamics/hdart/-issues/5)), we need to implement the group level methods to add filters between these meshes, implement visibility, and removal methods for these as a group.
- Add `libccd` related make variables to the `site-config-*` files
- Do further optimizations to get closer to `Bullet` performance.

Added benchmark to track how these affect performance [here](https://docs.google.com/spreadsheets/d/1cq1kPWtXv2sEakUSy8jUmBxSkPuiW8j/edit#gid=595681158) (https://docs.google.com/spreadsheets/d/1cq1kPWtXv2sEakUSy8jUmBxSkPuiW8j/edit#gid=595681158).

### 19.5.1.1. Minimal working example with `rtcCollide`

Before immediately trying to integrate `DMesh` with `rtcCollide`, I wanted to start with something more basic. I created basic structs to compose a bare-bones mesh, `Vertex` and `Triangle`. `Vertex` merely contains 3 floats (`x`, `y`, `z`), and `Triangle` contains 3 unsigned integer indices (`v0`, `v1`, `v2`). The struct `CustomMesh` stores two vectors, one for `Vertex` and one for `Triangle`.

#### 19.5.1.1.1. Adding a mesh to BVH

Currently `rtcCollide` only supports **user geometry type** (`RTC_GEOMETRY_TYPE_USER`) (see [here](#)

(<https://spec.oneapi.io/oneart/latest/embree-spec.html#rtcCollide>:-:text=Currently%2C%20the%20only%20supported%20type%20is%20the%20user%20geometry%20type%20(see%20RTC\_GEOMETRY\_TYPE\_USER)), so there are some extra steps you have to do to write your own intersection and bounding functions. I tried for a while to get `rtcCollide` working using a triangle geometry type (`RTC_GEOMETRY_TYPE_TRIANGLE`), but kept getting `segfault`.

Adding the mesh to the `embree` scene BVH is done in a series of steps: (these steps can be seen in the Github example [here](#)

([https://github.com/embree/embree/blob/master/tutorials/collide/collide\\_device.cpp#L20](https://github.com/embree/embree/blob/master/tutorials/collide/collide_device.cpp#L20):-:text=/%20iP%20(use\_user\_geometry)%20%7B,RTCGeometry%20geom%20%3D%20rtcNewGeometry%20(g\_device%2C%20RTC\_GEOMETRY\_TYPE\_USER)%3B,rtcReleaseGeometry(geom)%3B,meshes.push\_back%20(std%3A%3Amove%20(sphere))

).

- Create the new geometry - `RTCGeometry geom = rtcNewGeometry(g_device, RTC_GEOMETRY_TYPE_USER);`
- Attach the new geometry, which returns a unique ID - `unsigned int geomID = rtcAttachGeometry(g_scene, geom);`
- Set the number of primitives in geometry, i.e. for 1 triangle/primitive - `rtcSetGeometryUserPrimitiveCount(geom, 1);`
- Set geometry user data - this attaches a pointer to the "data" of the user geometry. When the `embree` callbacks are called, a `userptr` is often passed along with it. This `userptr` points to whatever you set the geometry user data to point to. What the Github example does is have a global variable that stores the meshes, and the geometry user data is nothing but an index in that global vector. Then, in the `rtcCollide` callbacks, they take that `userptr` and use it to index the global vector of meshes. When adding the geometry to the scene, they use the returned `geomID` variable from earlier to index the vector. This is how I did it too, since I basically copied the structure of the Github example. `rtcSetGeometryUserData(geom, (void*)(size_t)geomID);`
- Set the geometry bounds function - this function returns the upper and lower coordinates in all 3 dimensions of an arbitrary mesh element - `rtcSetGeometryBoundsFunction(geom, triangle_bounds_func, nullptr);`
- Set the geometry intersect function - right now I have this function unimplemented, as I believe `rtcCollide` does not use it. Basically the function you specify should determine whether a given ray intersects the mesh element. `rtcSetGeometryIntersectFunction(geom, triangle_intersect_func);`
- Commit the geometry to the scene - `rtcCommitGeometry(geom);`
- Release the geometry - `rtcReleaseGeometry(geom);`
- Commit the scene - `rtcCommitScene(g_scene);`

#### 19.5.1.1.2. How I created my CustomMesh to test rtcCollide

Sometime before you commit the scene, you make the actual mesh. For example, the way I have set up my basic `CustomMesh` type:

```
std::unique_ptr<CustomMesh> sample(new CustomMesh());
sample->vertices.resize(3);
sample->tris.resize(1);

sample->vertices[0] = Vertex(0,0,1);
sample->vertices[1] = Vertex(0,1,0);
sample->vertices[2] = Vertex(1,0,0);

sample->tris[0] = Triangle(0,1,2);

meshes.push_back(std::move(sample));
```

C++

Where `meshes` is a global vector of the `CustomMesh` type. I created two meshes, each with one triangle each, that intersect each other; in order to test out `rtcCollide`.

#### 19.5.1.1.3. User geometry bounds function

The user geometry bounds function (set with `rtcSetGeometryBoundsFunction`) is pretty straightforward. The callback has a number of parameters passed into it, namely the `userptr`, the `primID` (primitive ID), and the `bounds` struct. Essentially, using the `userptr` to locate the mesh in question and the `primID` to locate the specific primitive in question, the function must update the `bounds` struct to have the upper and lower limits of the mesh element in all 3 dimensions. When the geometry is committed to the scene, `embree` constructs the BVH using this function. Here is how I implemented it for my basic mesh:

```
void triangle_bounds_func(const RTCBoundsFunctionArguments* args) {
    void* ptr = args->geometryUserPtr;
    unsigned geomID = (unsigned) (size_t) ptr;
    auto const &mesh = *meshes[geomID];

    const Vertex* tri_verts[3];
    tri_verts[0] = &mesh.vertices[mesh.tris[args->primID].v0];
    tri_verts[1] = &mesh.vertices[mesh.tris[args->primID].v1];
    tri_verts[2] = &mesh.vertices[mesh.tris[args->primID].v2];

    args->bounds_o->lower_x = std::min({tri_verts[0]->x, tri_verts[1]->x, tri_verts[2]->x});
    args->bounds_o->upper_x = std::max({tri_verts[0]->x, tri_verts[1]->x, tri_verts[2]->x});
    args->bounds_o->lower_y = std::min({tri_verts[0]->y, tri_verts[1]->y, tri_verts[2]->y});
    args->bounds_o->upper_y = std::max({tri_verts[0]->y, tri_verts[1]->y, tri_verts[2]->y});
    args->bounds_o->lower_z = std::min({tri_verts[0]->z, tri_verts[1]->z, tri_verts[2]->z});
    args->bounds_o->upper_z = std::max({tri_verts[0]->z, tri_verts[1]->z, tri_verts[2]->z});
}
```

C++

#### 19.5.1.1.4. Using rtcCollide

The `rtcCollide` function call is pretty simple when everything is set up properly. It is merely:

```
rtcCollide(g_scene, g_scene, &collideFunc, &sim_collisions);
```

C++

Where `collideFunc` is the collision callback and `sim_collisions` stores output collision information. What gets tricky is making sure that `collideFunc` does what it is supposed to do. `rtcCollide` implements "broad phase" collision detection, and `collideFunc` is supposed to take the pairs of mesh elements that are thought to be colliding and verify precisely that they are indeed intersecting. For a triangle mesh, you take the 3 vertices from the first mesh element and the 3 vertices from the second mesh element, and determine if they are intersecting. In the Github example, they wrote a bunch of helper functions to determine if two arbitrary triangles are indeed intersecting. These functions are contained in a class [here](https://github.com/embree/embree/blob/ae029e2ff83bebbbe8742c88aba5b0521aba1a23/kernels/geometry/triangle_triangle_intersector.h) ([https://github.com/embree/embree/blob/ae029e2ff83bebbbe8742c88aba5b0521aba1a23/kernels/geometry/triangle\\_triangle\\_intersector.h](https://github.com/embree/embree/blob/ae029e2ff83bebbbe8742c88aba5b0521aba1a23/kernels/geometry/triangle_triangle_intersector.h)). I adapted their method to work with `SOAVector3` rather than their vector type, `Vec3fa`.

When I use `rtcCollide` with my `collideFunc`, and add two meshes with single triangles that intersect each other, I record a collision!

#### 19.5.1.1.5. Next Steps

I will test a variety of different test cases with minimal amounts of triangles to verify that my `collideFunc` works properly. Then I will adapt my code to integrate with `DMesh`, which shouldn't be too cumbersome since `DMesh` already has a lot of the necessary tools.

#### 19.5.1.2. Tailoring example to work with `DMesh`

Editing the minimal working example above to integrate with `DMesh` was fairly straightforward. `DMesh` has helper methods to access a certain vertex or triangle number with `getVertex(size_t vertex_index)` and with `getFace(size_t face_index)`, so really all I had to do was change the code to work with `SOAVector3` rather than my own simple `Vertex` struct.

Another thing that became apparent is that triangles near each other in the same mesh are counted as "collisions" by `rtcCollide`. In case we want to be able to handle self-intersecting meshes, we can check for triangle adjacency in the mesh (by seeing if the two "colliding" triangles share at least one vertex). However, the ground plane mesh that I am working with for some reason has duplicate vertices (i.e. vertices that have the exact same position but different indexes), so in order to truly check for adjacency, you have to compare the positions of each vertex and not just the indices of each vertex in the vertex array. All in all, this might be computationally expensive to do for meshes that we know are not going to self-intersect (such as a ground plane), so it might be better to just ignore any collision between triangles of the same mesh.

Here is a screenshot of a result of colliding a 1 m diameter sphere mesh that is 0.5 m above a ground plane mesh. Seems to be a reasonable result.

```
[stobin@leibniz NdartsTest]$ srun test_rtcCollide
Number of triangles in ground: 7708
Number of triangles in sphere: 1280
Total number of collisions: 20
```

#### 19.5.1.3. First approach to finding contact points

Note I (knowingly) make the following assumptions:

- small penetration distance
- meshes are convex polyhedra

##### 19.5.1.3.1. High level overview

The way I determine the contact points, penetration distance, and collision normal is as follows:

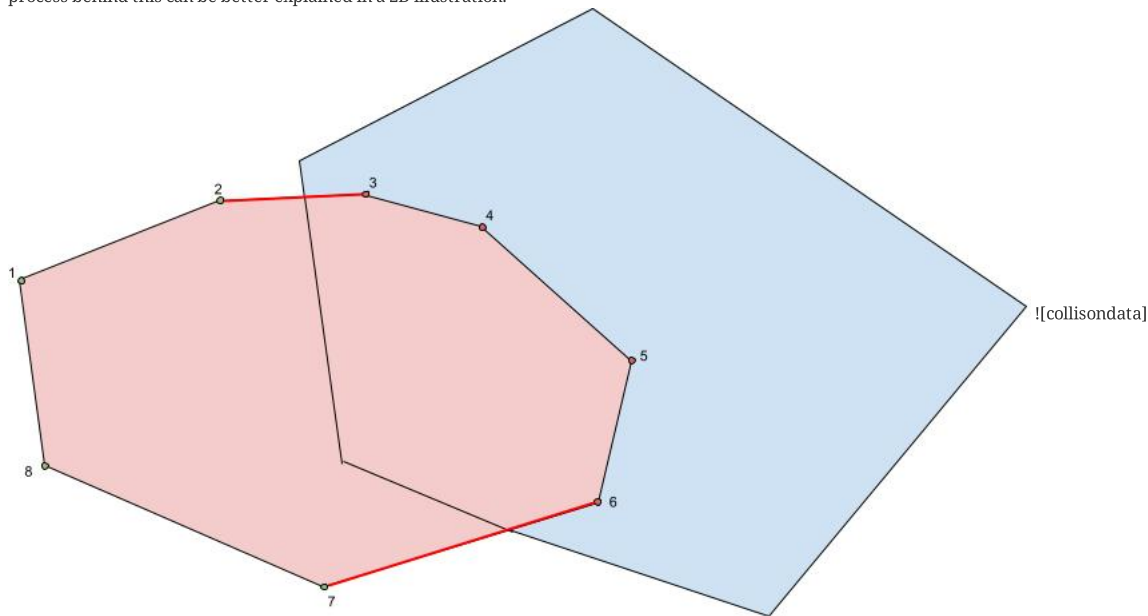
- Perform initial collision detection with `rtcCollide`, which tells us the faces of each mesh that are intersecting
- If there is a collision, get all vertices from one mesh that are penetrating (inside of) the other mesh
- Use `libccd` to run the GJK algorithm on the two colliding meshes. The `libccd` implementation can return the minimum separating axis, which I use as the collision normal.
- Use embree's `rtcIntersect` to cast rays from all the penetrating vertices in the collision normal direction. The distance until the ray collides with the other mesh is the penetration distance for that vertex.
- Use the penetration distances to determine the contact points - i.e. if there is only one vertex with the max penetration distance, then there was a point collision, so there is just one contact point.

##### 19.5.1.3.2. Initial collision detection with `rtcCollide`

See above points for some more specific details on how I set up and used `rtcCollide`.

##### 19.5.1.3.3. Find all penetrating vertices

I try to be clever about this; rather than checking every single point to see if it's inside the other mesh, I only check the points of the faces that intersect with the other mesh. The thought process behind this can be better explained in a 2D illustration.



(uploads/22362f5fc3594a030d97858898cbe02b/collisiondata.png)

The edges in red are the ones that intersect with the blue shape, and are analogous to the faces that are returned by `rtcCollide`. So, we know that one vertex on the red edges are inside the blue shape, and one vertex is outside. If we first check vertices 2, 3, 6 and 7 for being inside of the blue shape, we will find that vertices 3 and 6 are penetrating the blue shape, and vertices 2 and 7 are not. Then, because both shapes are convex, any vertices connected to vertices 3 and 6 must also be inside the blue shape as well. So, without having to check for penetration, we can determine that vertices 4 and 5 are also penetrating the blue shape, and all other vertices are not.

This same logic carries over to 3D convex polyhedra, and potentially saves a lot of time if there are a lot of vertices that are penetrating the other mesh. It also fits nicely with what `rtcCollide` gives us.

#### 19.5.1.3.4. Use `libccd` to perform GJK algorithm

Using `libccd` is pretty simple, most of the work is setting up what is called the "support function". The support function is a function that, given a direction, returns the vertex in the mesh that is furthest away along that direction. How this is computed is actually pretty simple: it is merely the vertex that has the largest dot product with the direction. `libccd` specifies how the function should be implemented [here](https://libccd.readthedocs.io/en/master/examples.html) (https://libccd.readthedocs.io/en/master/examples.html). `obj` is a void pointer that refers to the user geometry. I decided to just pass the index in the meshes vector; like what is done with the embree callbacks. Then, I do a simple loop to determine the vertex with the maximum dot product with the direction `dir`, and save that vertex in the `vec` parameter. The specific implementation can be seen below.

```
void gjk_support_func(const void *obj, const ccd_vec3_t *dir, ccd_vec3_t *vec) {
    unsigned geomID = *(unsigned*) obj;
    auto const& mesh = *meshes[geomID];
    auto const& vertices = mesh.getPositions();
    const SOAVector3 SOAdir(ccdVec3X(dir), ccdVec3Y(dir), ccdVec3Z(dir));

    double max_dot = std::numeric_limits<double>::lowest();
    double dot;
    const SOAVector3* best_vert;

    for (auto& v : vertices) {
        dot = v.dot(SOAdir);
        if (dot > max_dot) {
            best_vert = &v;
            max_dot = dot;
        }
    }

    ccdVec3Set(vec, (*best_vert)[0], (*best_vert)[1], (*best_vert)[2]);
}
```

To get the separation vector (a.k.a. the collision normal), I use the `libccd` call defined [here](https://github.com/danfis/libccd/blob/master/src/ccd/ccd.h#:~:text=CCD_EXPORT%20int,ccdGJKSeparate,-(const%20void))) (https://github.com/danfis/libccd/blob/master/src/ccd/ccd.h#:~:text=CCD\_EXPORT%20int,ccdGJKSeparate,-(const%20void)). I set it up like so:

```
ccd_t ccd;
CCD_INIT(&ccd);

ccd.support1 = &gjk_support_func;
ccd.support2 = &gjk_support_func;
ccd.max_iterations = 100;
ccd.epa_tolerance = 0.0001;

ccd_vec3_t dir;
int intersect = ccdGJKSeparate(&geomID1, &geomID0, &ccd, &dir);
```

`dir` now stores the separation vector that I can use in the next step, casting rays to determine penetration distance.

### 19.5.1.3.5. Use embree rays to find penetration distance of each penetrating point

With the penetrating points and separation vector found, we can use embree to cast rays and find the intersection with the other mesh. This requires the use of a [geometry intersection function](https://spec.oneapi.io/oneapi/latest/embree-spec.html#rtcsetgeometryintersectfunction) (https://spec.oneapi.io/oneapi/latest/embree-spec.html#rtcsetgeometryintersectfunction:-:text=RTC\_GEOMETRY\_TYPE\_USER,-rtcSetGeometryIntersectFunction,-%C2%B6), which checks for the intersection of a ray with user defined geometry primitives. Luckily, the embree collision detection GitHub example has this function implemented [here](https://github.com/embree/embree/blob/master/tutorials/collide/collide.cpp#:~:text=triangle_intersect_func) (https://github.com/embree/embree/blob/master/tutorials/collide/collide.cpp#:~:text=triangle\_intersect\_func), which I adapted for use with DMesh and SOA. In order to ensure that the ray is colliding with the particular mesh we are interested in, I pass the mesh ID we would like to collide with in the [RTCRay.flags](https://spec.oneapi.io/oneapi/latest/embree-spec.html#rtcsetgeometryintersectfunction:-:text=/%20ray%20ID,-unsigned%20int%20flags,-%3B%20/%20ray%20flags) property (https://spec.oneapi.io/oneapi/latest/embree-spec.html#rtcsetgeometryintersectfunction:-:text=/%20ray%20ID,-unsigned%20int%20flags,-%3B%20/%20ray%20flags). Then, if in the geometry intersection callback the mesh ID of the face in question is not the one we care about, there is no need to compute the potential intersection and we can return without doing anything. This not only saves time but makes sure we are getting the correct penetration distances for the collision at hand. My implementation of the intersection function can be seen below:

```
void triangle_intersect_func(const RTCIntersectFunctionNArguments* args) {
    // unpack args
    void* ptr = args->geometryUserPtr;
    RTCRayHit* rayhit = (RTCRayHit*)args->rayhit;
    RTCRay* ray = &rayhit->ray;
    RTCHit* hit = &rayhit->hit;

    // ID of primitive
    unsigned int primID = args->primID;
    // mesh ID of current primitive
    unsigned geomID = (unsigned) (size_t) ptr;

    // the mesh ID we are interested in intersecting with is stored in the RTCRay flags
    unsigned hit_geomID = ray->flags;

    // return if the mesh ID of the primitive is not the one we're interested in intersecting with
    if (geomID != hit_geomID)
        return;

    // calculate potential intersection
    auto const & mesh = *meshes[geomID];
    auto const & tri = mesh.getFace(primID);

    const SOAVector3& v0 = mesh.getVertex(tri.v0);
    const SOAVector3& v1 = mesh.getVertex(tri.v1);
    const SOAVector3& v2 = mesh.getVertex(tri.v2);

    const SOAVector3 e1 = v0-v1;
    const SOAVector3 e2 = v2-v0;

    const SOAVector3 Ng = e1.cross(e2);

    const SOAVector3 O(ray->org_x, ray->org_y, ray->org_z);
    const SOAVector3 D(ray->dir_x, ray->dir_y, ray->dir_z);
    const SOAVector3 C = v0 - O;
    const SOAVector3 R = D.cross(C);

    double den = 1 / (Ng.dot(D));

    double u = R.dot(e2) * den;
    double v = R.dot(e1) * den;

    bool valid = (den != 0.0) & (u >= 0.0) & (v >= 0.0) & (u+v<=1.0);
    if (!valid) return;

    // calculate distance of intersection
    double t = Ng.dot(C) * den;
    valid &= (t > ray->tnear) & (t < ray->tfar);
    if (!valid) return;

    // set hit parameters to register an intersection
    ray->tfar = t;
    hit->u = u;
    hit->v = v;
    hit->geomID = geomID;
    hit->primID = primID;
    hit->Ng_x = Ng[0];
    hit->Ng_y = Ng[1];
    hit->Ng_z = Ng[2];
}
```

### 19.5.1.3.6. Determining contact points from penetration points

I classify contacts as a couple different types:

#### Point Contact

These occur when one of the colliding meshes has just one penetrating point with the deepest penetration distance. One contact point is generated at this deepest penetrating point. This type of contact includes Point-Point, Point-Edge, and Point-Face collisions.

#### Edge Contact

These occur when one of the colliding meshes has exactly two penetrating points with the deepest penetration distance, and none of the meshes have a singular deepest penetrating point (this would be a point contact). One contact point is generated at the midpoint of the deepest penetrating line segment. This type of contact includes Edge-Edge and Edge-Face collisions.

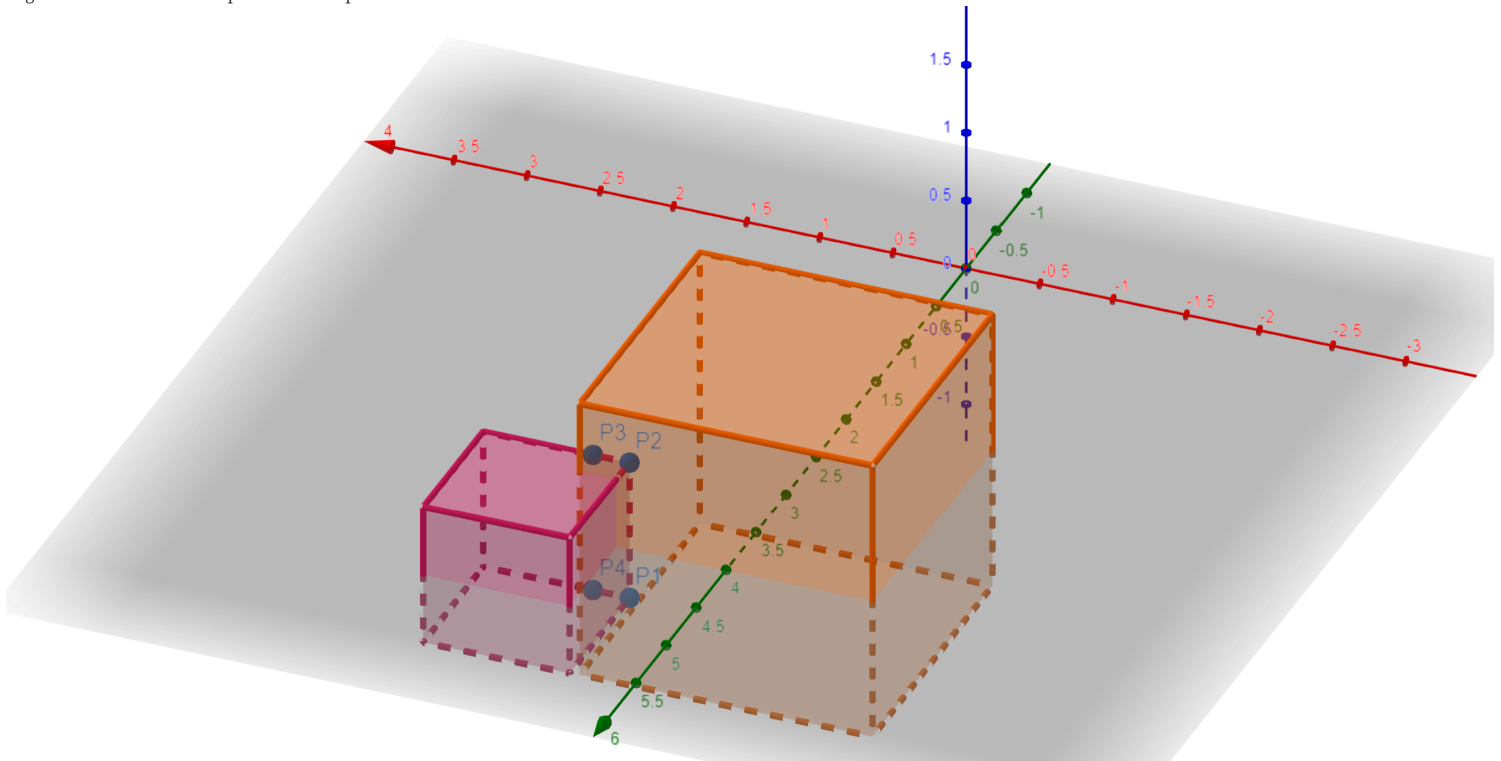
#### Face Contact

These occur when both of the colliding meshes have more than two penetrating points with the deepest penetration distance. Multiple contact points are generated; one at each vertex of the penetrating face. This type of contact only includes Face-Face collisions.

#### 19.5.1.4. Examining DBulletTest examples

##### 19.5.1.4.1. Plane-Plane example

The categorization above seems to align reasonably well with what Bullet does for contact point determination. For example, take the Face-Face contact case shown below, where two axis-aligned cubes collide with a penetration depth of 0.22.



```

Number of contacts: 4
Broad Penetration Depth: 0.22
Broad Collision Normal: (-0, -1, -0)
Broad Collision Position: (0.75, 3.78, -0.5)
Moving Cube Position: (1.25, 4.28, 0)



---


Broad Penetration Depth: 0.22
Broad Collision Normal: (-0, -1, -0)
Broad Collision Position: (0.75, 3.78, 0.5)
Moving Cube Position: (1.25, 4.28, 0)



---


Broad Penetration Depth: 0.22
Broad Collision Normal: (-0, -1, -0)
Broad Collision Position: (1, 3.78, 0.5)
Moving Cube Position: (1.25, 4.28, 0)



---

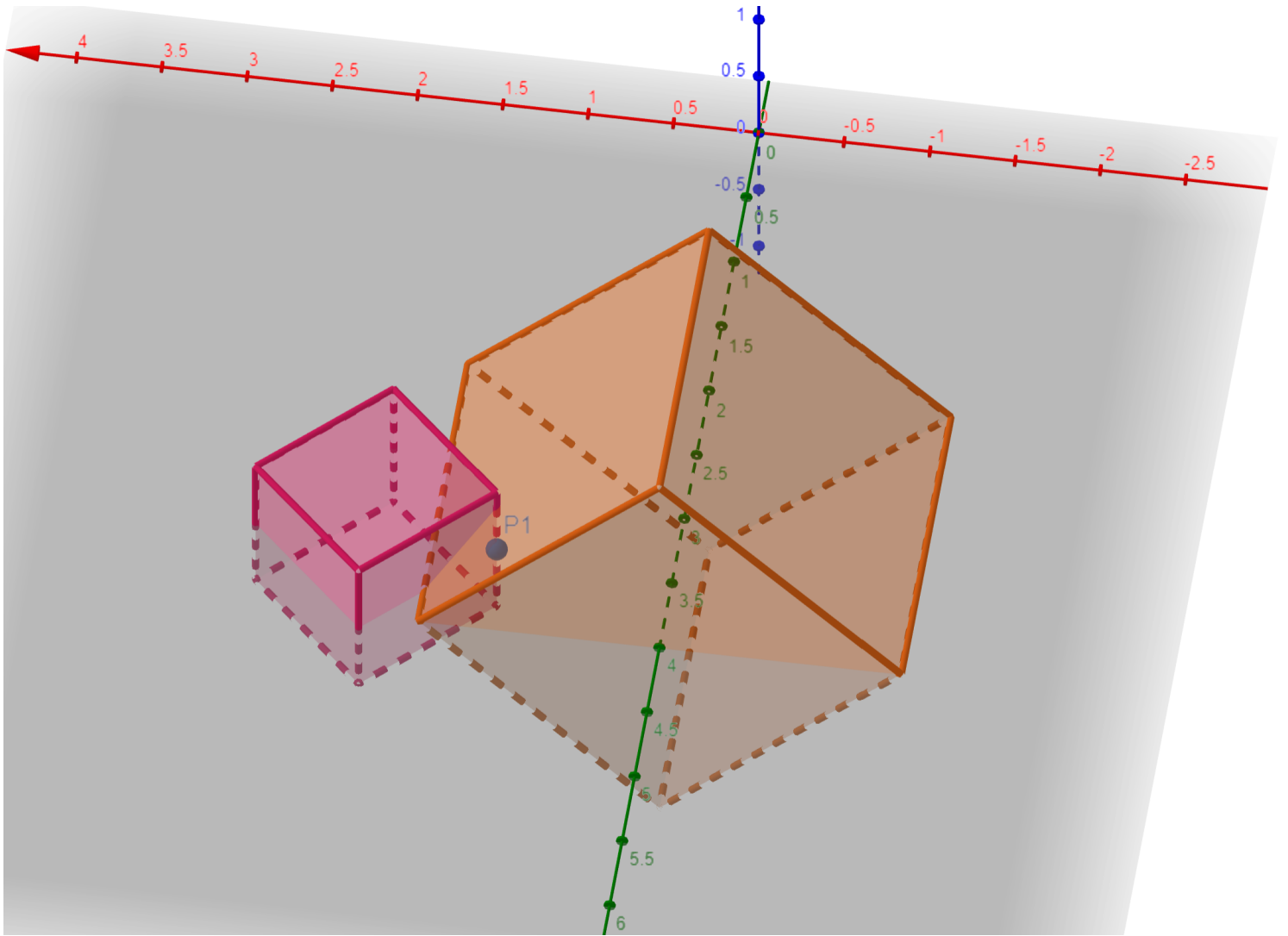

Broad Penetration Depth: 0.22
Broad Collision Normal: (-0, -1, -0)
Broad Collision Position: (1, 3.78, -0.5)
Moving Cube Position: (1.25, 4.28, 0)
    
```

The points P1, P2, P3, and P4 are the contact points on the red cube determined by Bullet - they are the vertices of the colliding face. And, as we can see, the collision normal is the minimum separating vector; which in this case is the negative Y-axis (the green axis in the picture). The visualization was done with GeoGebra and can be viewed and tinkered with [here](https://www.geogebra.org/3d/ysvwj5bb) (https://www.geogebra.org/3d/ysvwj5bb).

##### 19.5.1.4.2. Edge-Edge example

Another interesting example is the Edge-Edge contact case shown below, where two rotated cubes collide perpendicularly through their edges.



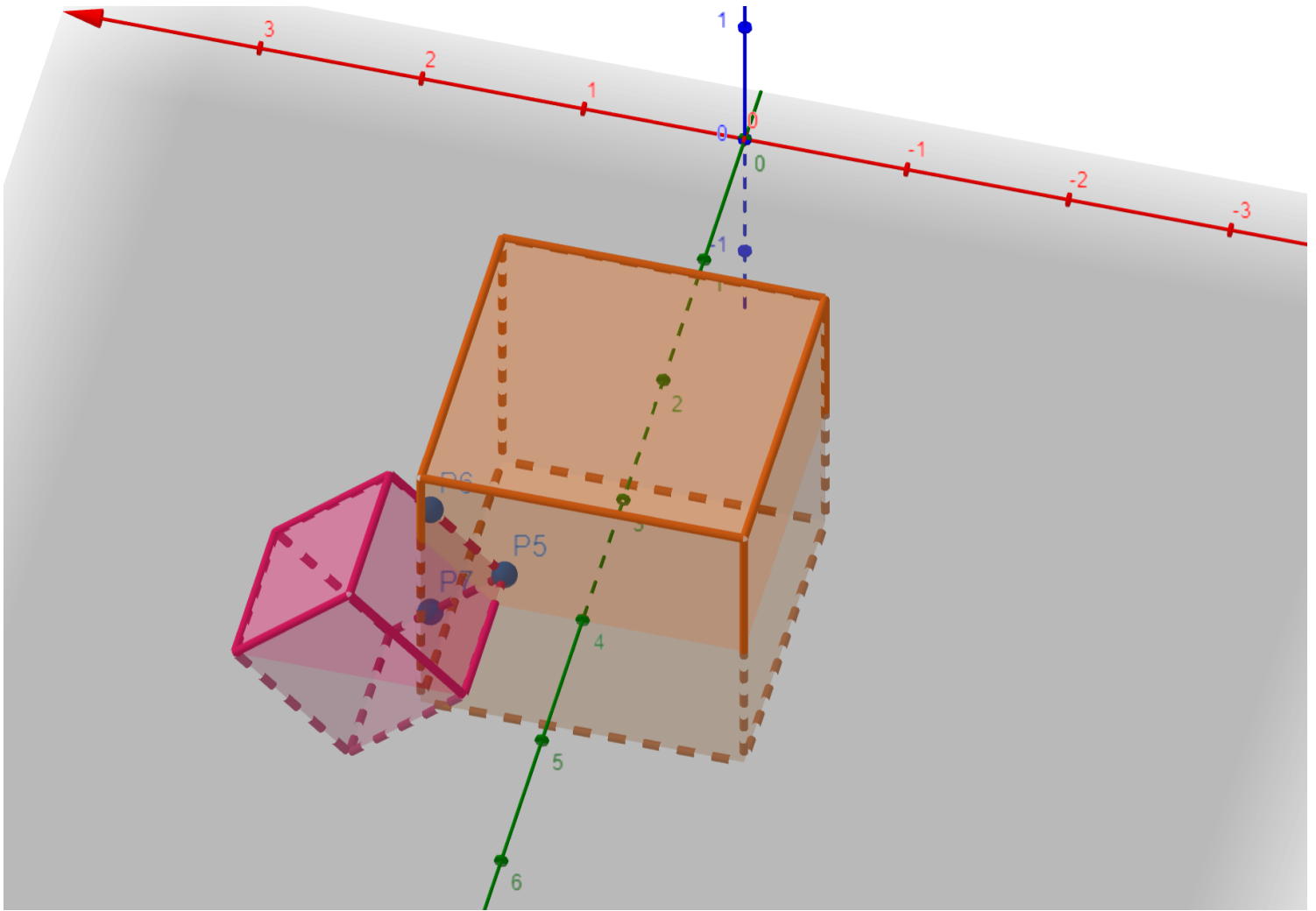


The point P1 is the contact point on the red cube determined by Bullet. This is simply the point of deepest penetration. GeoGebra link: [here](https://www.geogebra.org/3d/dmw9xnur) (https://www.geogebra.org/3d/dmw9xnur)

#### 19.5.1.4.3. Edge-Plane example

In the Edge-Plane example, a rotated cube collides with an axis-aligned cube such that the rotated cube's edge collides with one of the faces of the axis-aligned cube. This example involves two types of collisions, Face-Face and Edge-Face, and they come about depending on the collision normal (the minimum separating vector).

Face-Face:



**Number of contact points: 3**

Broad Penetration Depth: 0.24

Broad Collision Normal: (-0,-1,-0)

Broad Collision Position: (0.542893,3.76,-1.63797e-05)

Moving Cube Position: (1.25,4.26,0)

**Broad Penetration Depth: 0.24**

**Broad Collision Normal: (-0,-1,-0)**

**Broad Collision Position: (1,3.76,0.457112)**

**Moving Cube Position: (1.25,4.26,0)**

**Broad Penetration Depth: 0.24**

**Broad Collision Normal: (-0,-1,-0)**

**Broad Collision Position: (1,3.76,-0.457102)**

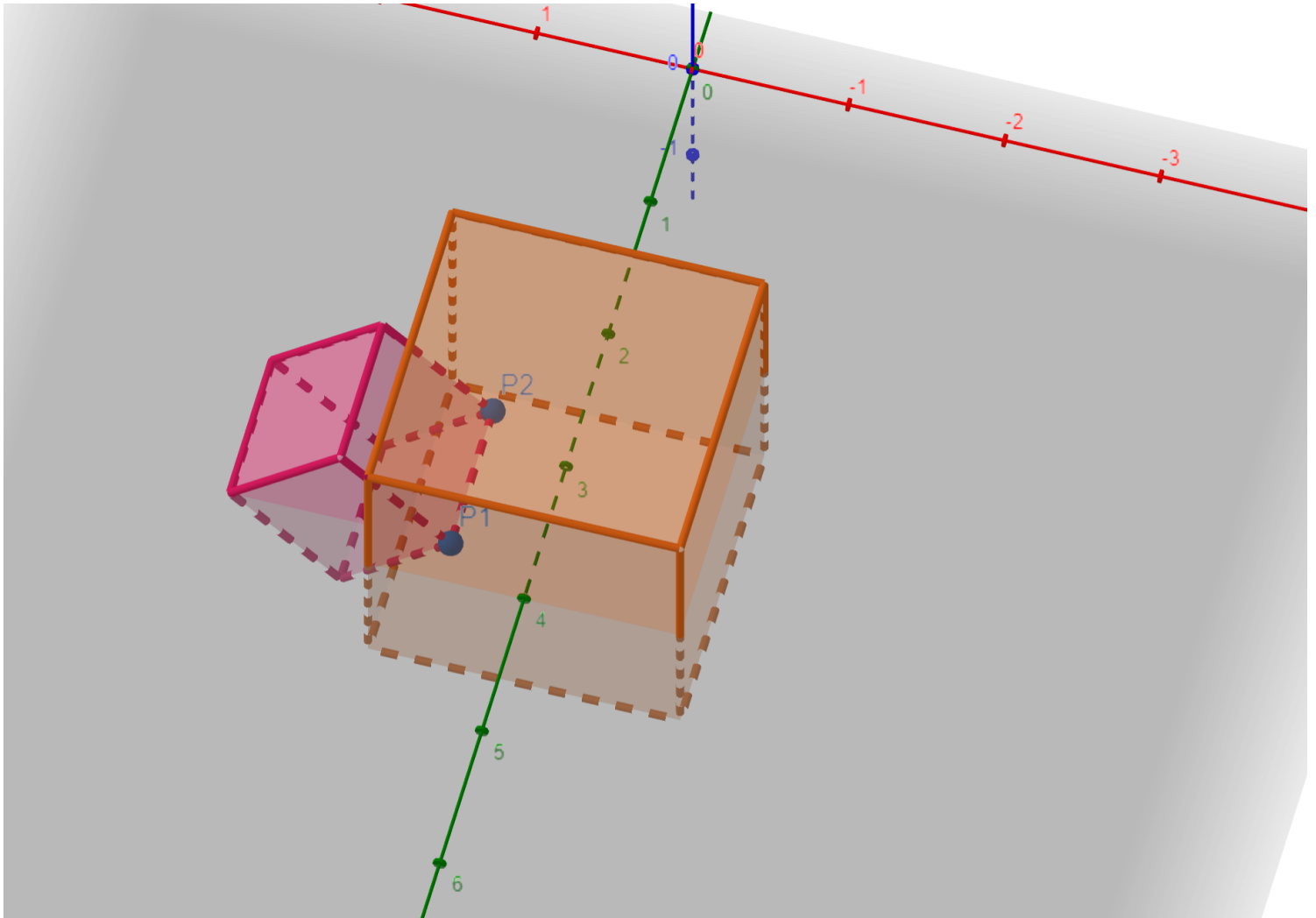
**Moving Cube Position: (1.25,4.26,0)**

In the Face-Face collision, the collision normal is along the Y (green) axis, so there are 3 points with the deepest penetration into the larger cube. As such, these 3 points (P1, P2, P3) are the contact points determined by Bullet.

Edge-Face:

Number of contact points: 2  
Broad Penetration Depth: 0.457107  
Broad Collision Normal: (-1,-0,-0)  
Broad Collision Position: (0.542893,2.73,-1.63797e-05)  
Moving Cube Position: (1.25,3.23,0)

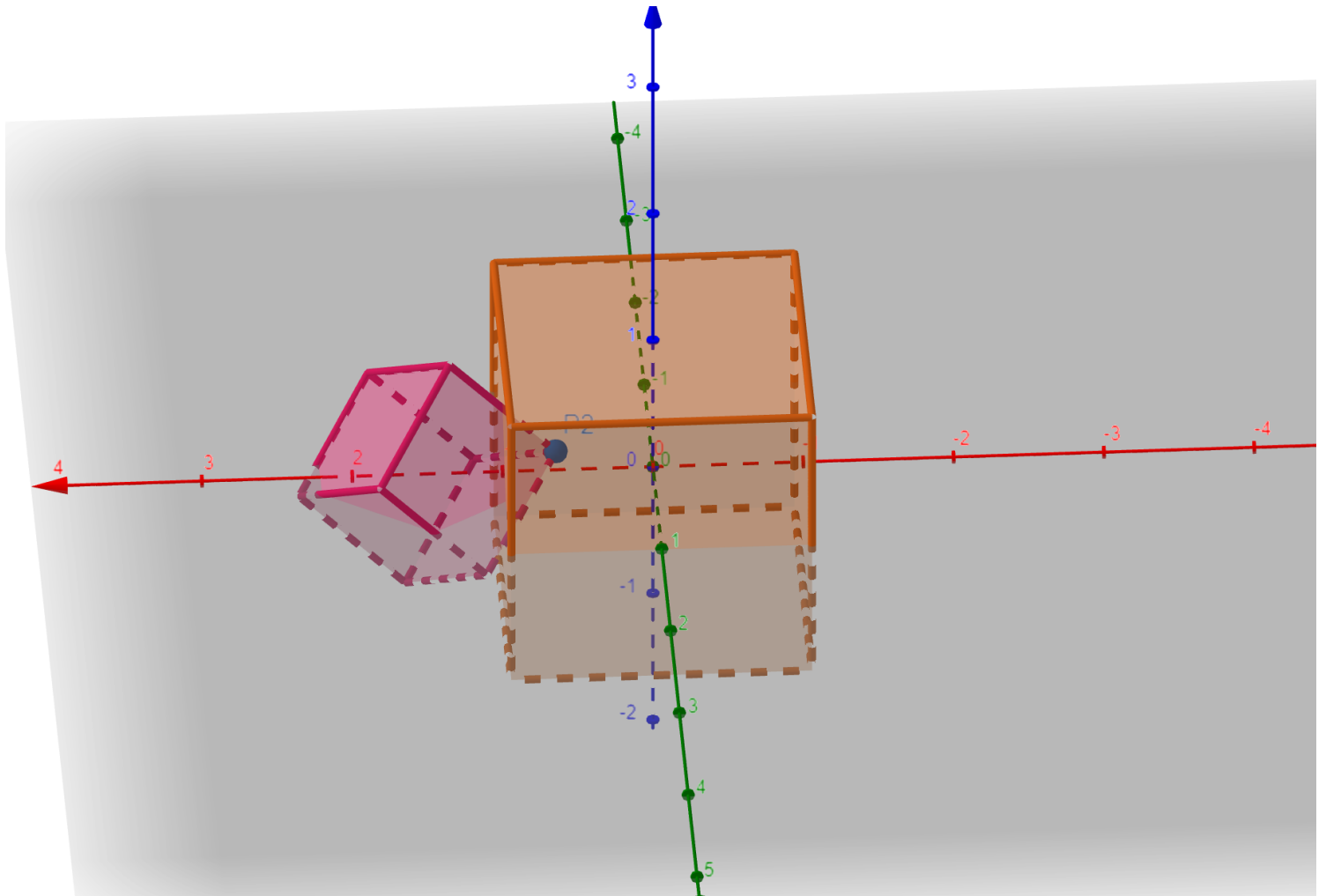
Broad Penetration Depth: 0.457107  
Broad Collision Normal: (-1,-0,-0)  
Broad Collision Position: (0.542893,3.73,-1.63797e-05)  
Moving Cube Position: (1.25,3.23,0)



In Edge-Face collision, the collision normal is along the X (red) axis, so there are only 2 points with the deepest penetration into the larger cube. As such, these 2 points (P1, P2) are the contact points determined by Bullet. GeoGebra link: [here](https://www.geogebra.org/3d/vh7y2vpq) (https://www.geogebra.org/3d/vh7y2vpq)

#### 19.5.1.4.4. Vertex-Plane example

In the Vertex-Plane example, a cube rotated once about the Z-axis and once about the Y-axis collides with an axis-aligned cube such that a rotated cube's vertex collides with one of the faces of the axis-aligned cube. This example should just involve one type of collision (by my logic), a Vertex-Face collision, which only generates one contact point. For small penetrations, Bullet agrees with this. However, when more of the rotated cube's vertices penetrate the larger cube's face, Bullet produces multiple contact points. This can be seen below:



**Number of contacts: 1**

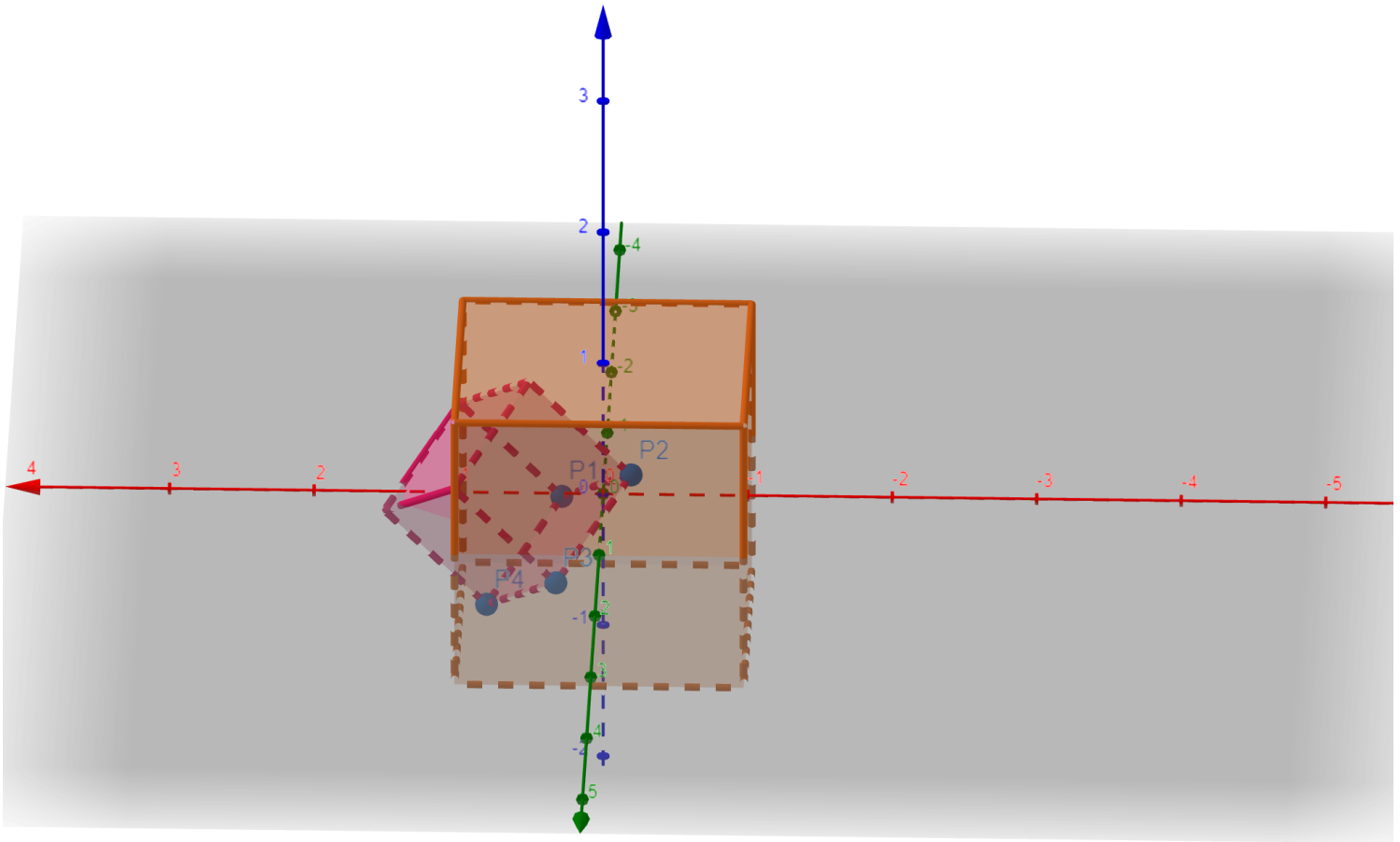
Broad Penetration Depth: 0.353557

Broad Collision Normal: (-1,-0,-0)

Broad Collision Position: (0.646443,1.63797e-05,0.146427)

Moving Cube Position: (1.5,0,0)

Deeper penetration (4 contact points):



**Number of contacts: 4**

**Broad Penetration Depth: 0.693557**

**Broad Collision Normal: (-1,-0,-0)**

**Broad Collision Position: (0.306443,-0.707107,-0.35355)**

**Moving Cube Position: (0.66,0,0)**

**Broad Penetration Depth: 1.19356**

**Broad Collision Normal: (-1,-0,-0)**

**Broad Collision Position: (-0.193557,1.63797e-05,0.146427)**

**Moving Cube Position: (0.66,0,0)**

**Broad Penetration Depth: 0.693534**

**Broad Collision Normal: (-1,-0,-0)**

**Broad Collision Position: (0.306466,0.707107,-0.353573)**

**Moving Cube Position: (0.66,0,0)**

**Broad Penetration Depth: 0.193534**

**Broad Collision Normal: (-1,-0,-0)**

**Broad Collision Position: (0.806466,-1.63797e-05,-0.85355)**

**Moving Cube Position: (0.66,0,0)**

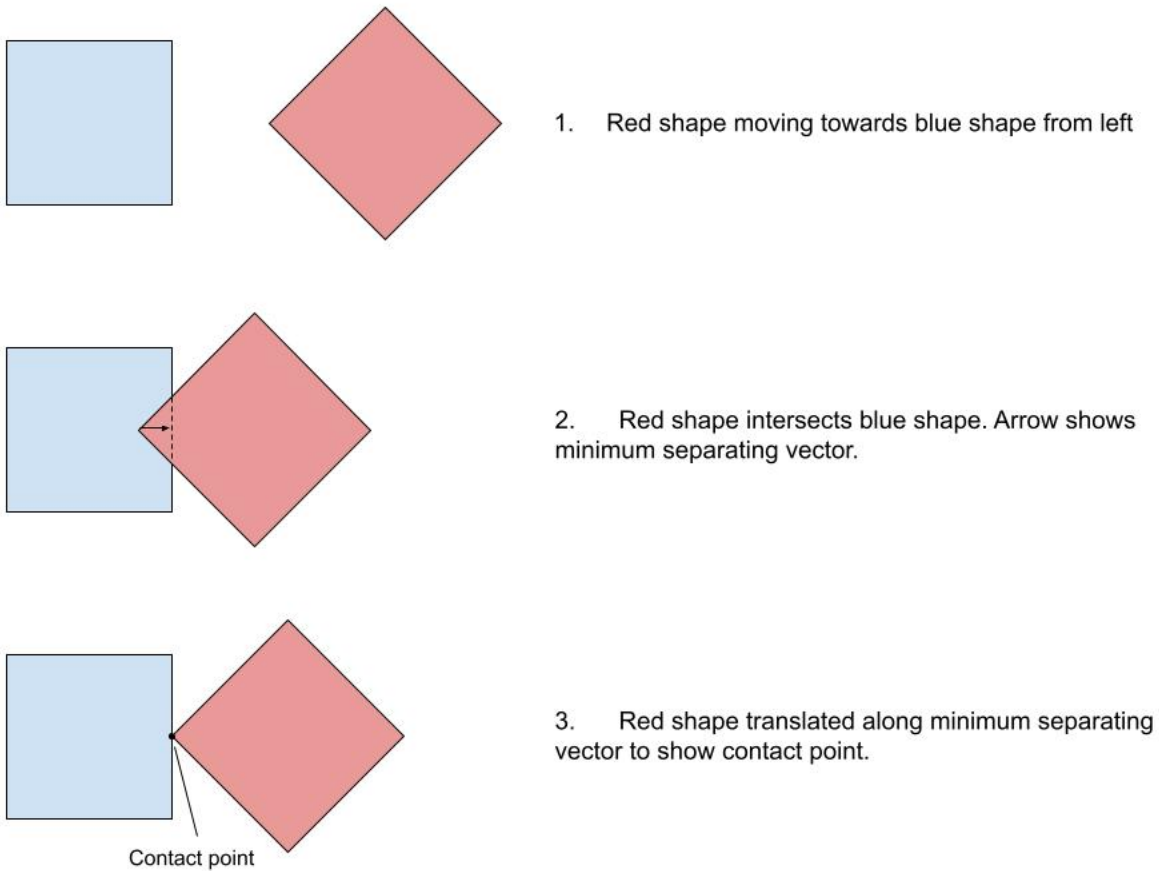
In my opinion, this doesn't make a whole lot of sense, since there is only one point with the deepest penetration, so really the collision is only happening between that point and the face of the larger cube. That being said, Bullet probably has a good reason for doing it this way. In any case, it is something interesting to note and be aware of. GeoGebra link: [here](https://www.geogebra.org/3d/suftxcgg)  
(<https://www.geogebra.org/3d/suftxcgg>)

### 19.5.1.5. Some intuition behind how contact points are determined

From doing a little research online, it seems there is no standard way to determine/define what contact points should be. I've defined them as what I believe to be most intuitive: **given two colliding shapes that penetrate one another, if you rewind time to the time at which they first touch, the points where they touch are the contact points for the collision.**

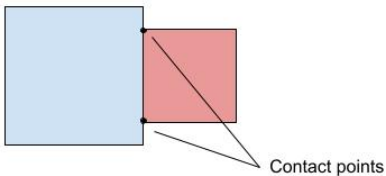
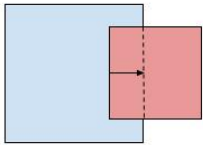
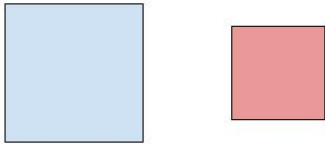
The intuition behind this makes sense if we consider a simulation scenario where we have an infinitesimal time step. Then, we would be able to detect collisions at the exact instant that they happen. The collision contact points would be very well defined; they would simply be the points at which the two meshes touch.

But, simulations don't use an infinitesimal time step, and we detect collisions with nonzero penetrations. So how do we rewind to a time in between time steps to get the time where the two meshes first touched? In short, we don't. Instead, what we can do as an approximation is use the minimum separating direction and penetration distance of the collision. (Penetration distance = minimum translation distance required to fully separate the two meshes). We can move one of the meshes the penetration distance along the minimum separating direction, resulting in a contact that mimics what we would see if we rewound time. Assuming small penetration distances, this approximation will work well. This kind of logic can be seen in the simple 2D case below.

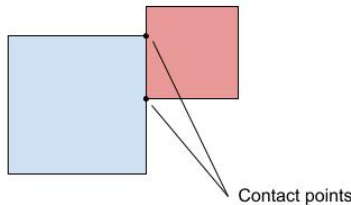
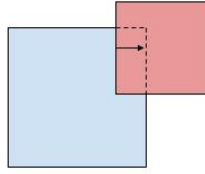
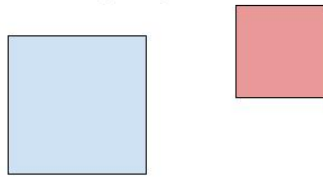


If you notice, the contact point in this case is actually the vertex on the shape with the deepest penetration in step 2! So, this provides us with an initial intuition for how to find contact points for a collision: **look for vertices with the deepest penetration.** There can also be more than one contact point, like in 2D when there is an edge-edge intersection. Two examples of this can be seen the 2D example below.

### Centered edge-edge collision



### Offset edge-edge collision



1. Red shape moving towards blue shape from left

2. Red shape intersects blue shape. Arrow shows minimum separating vector.

3. Red shape translated along minimum separating vector to show contact points.

Here, we can see that the edges intersect squarely, creating an infinite number of points with the deepest penetration. Obviously, we cannot have an infinite number of contact points, so we select the endpoints of the line segment. Notice how in the case on the left, the centered edge-edge collision, the endpoints of this intersecting line are vertices of the red shape. However, for the case on the right, the offset edge-edge collision, only one endpoint is a vertex on the red shape. The other endpoint is where edges from the blue and red shapes intersect, a so-called **intersection point** (this is how I'll be referring to points such as this one from now on). This is a crucial feature to capture, since this will drastically affect the applied forces and moments to the collision, so we have to not only consider just vertices but also intersection points as potential contact points. Thus, we can modify our initial intuition to the following: **look for vertices or intersection points with the deepest penetration**. This will ultimately be the main guiding criterion for determining contact points.

#### 19.5.1.6. Updated contact point determination for two colliding meshes

Above, we developed some intuition on what a contact point should be: a point (vertex or intersection point) with the deepest penetration. So, how do we put this into practice efficiently?

##### 19.5.1.6.1. High level overview

- **Inputs:** two meshes that are potentially colliding
- **Outputs:** contact points for the collision, if there is one

##### 19.5.1.6.1.1. Algorithmic steps

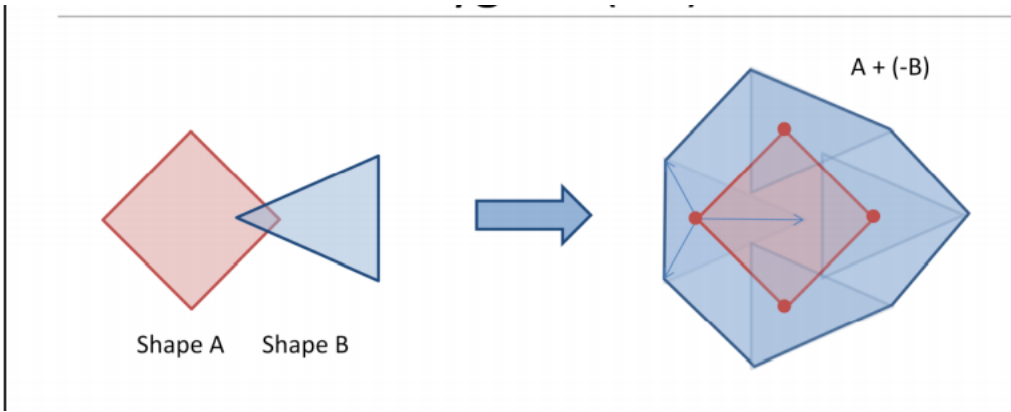
- Run `libccd`'s `ccdGJKPenetration` function to determine the minimum separating vector, penetration distance, and "position" of the collision, if there is one.
- Starting at the mesh vertex closest to the `libccd` "position" (which should at least approximately be the deepest vertex), traverse adjacent connected vertices until there are no more vertices with the deepest penetration depth. Add all vertices with the deepest penetration depth to be examined further as potential contact points. Do this for each mesh.
- Take the potential contact points and find the convex hull of them. This step also removes collinear and duplicate points. Do this for each mesh.
- Check the number of potential contact points for each mesh and use that to determine the type of collision. Add the relevant contact points to a final vector to be returned.

##### 19.5.1.6.2. `libccd ccdGJKPenetration` function call

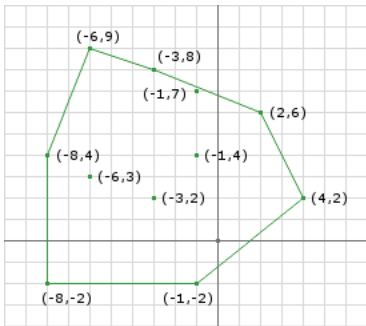
The `ccdGJKPenetration` runs the GJK algorithm augmented with the EPA (Expanding Polytope Algorithm) algorithm. The GJK algorithm alone only is able to detect *if* there is a collision between two shapes, not anything more. The EPA algorithm is an augmentation to GJK, and is able to determine the minimum separating vector, penetration depth, and position of the collision, which is ultimately what we care about when determining contact points.

##### 19.5.1.6.2.1. Brief summary of GJK and EPA (for some context)

GJK is a pretty elegant solution for determining if two polygons (or polyhedrons in 3D) intersect. It is primarily based around something called the Minkowski difference. Something similar, called the Minkowski sum, can be explained a little easier. The Minkowski sum is defined as the polygon that contains every point in one shape added to every point in the other. The Minkowski difference, in contrast, is the polygon that contains every point in one shape added to the negative of every point in the other shape. A way of thinking about it is sweeping the inverted polygon B around the edges of polygon A, and taking the resulting polygon as the Minkowski difference. This sort of logic can be shown below:



The Minkowski difference can be used to determine if two polygons are intersecting; if the Minkowski difference contains the origin, then the polygons are intersecting! (Since the Minkowski difference can be thought of as subtracting every point contained in polygon B from every point contained in polygon A, then if the origin is in the Minkowski difference, there's at least one pair of points in polygon A and B such that they subtract to  $0 = \text{intersection!}$ ). This can be visualized better in the illustration below.



As we can see, the Minkowski difference (in green) contains the origin, and thus the two polygons collide. Once the Minkowski difference is found, the core of the GJK algorithm is to try different combinations of points on the boundary of the Minkowski difference in order to find a triangle that contains the origin. If one is found, then the shapes intersect!

EPA builds on this by using the final triangle that GJK produces and expanding it by adding vertices, looking for the edge (or face in 3D) that is closest to the origin. The vector from this edge to the origin is the penetration vector: These implementation details aren't crucial for us to understand; `libccd` handles them for us.

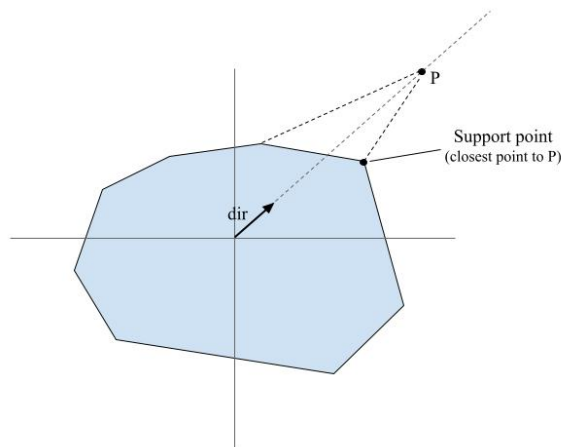
#### 19.5.1.6.2.2. GJK support function

The reason we have to understand a bit about the Minkowski difference is because `libccd` requires us to specify a **support function**, which is used to compute the Minkowski difference. The GJK support function is defined for a polygon as a function that, for a given direction, gives the vertex on the polygon that is furthest in that direction.

- insert visualization here \*\*

A simple way to calculate the support function would be to, for a given direction, take the dot product with every vertex in a mesh, as done [here](https://stackoverflow.com/questions/1969922/support-function-in-the-gjk-algorithm) (<https://stackoverflow.com/questions/1969922/support-function-in-the-gjk-algorithm>). The vertex with the highest dot product is the support point in that direction. However, this becomes very slow for meshes with a high number of vertices.

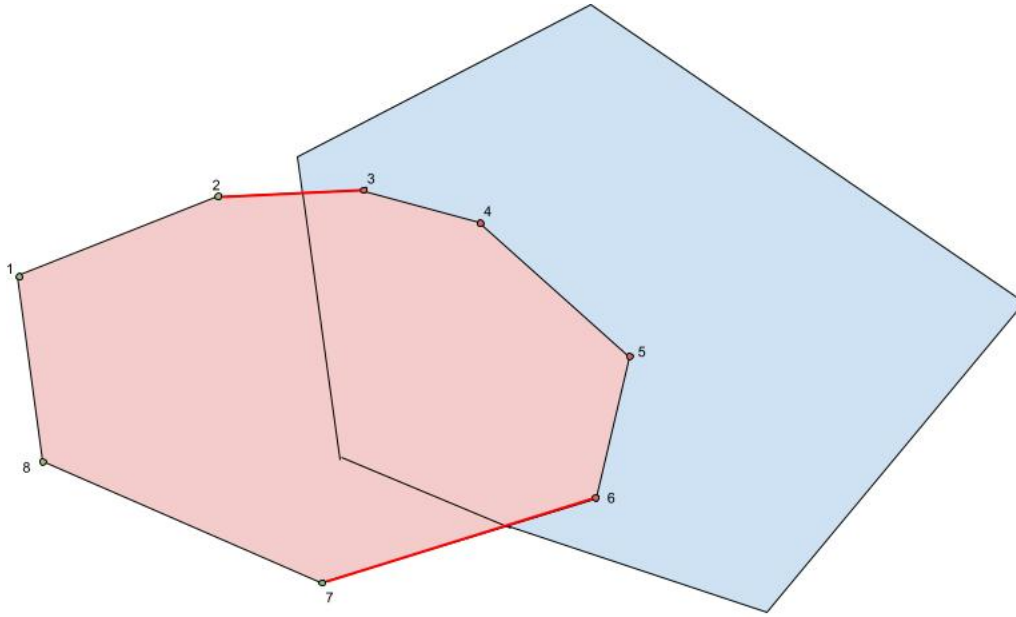
Another, faster way to calculate the support function is to perform an Embree point query. Say that for some given support direction `dir` we have some point `P` extremely far away from the origin, on the vector `dir`. The closest vertex to point `P` on the mesh will be the support point for that direction. This is a direct result of the shapes being convex. We can use the already existing `computeClosestVertex` function in `DMesh` to do this, and it's a clever way of avoiding iterating through the vertices. A 2D visualization of this can be seen below.



#### 19.5.1.6.3. Traversing vertices for contact points



As mentioned in a post above, our criterion for contact points are vertices or intersection points that have the maximum penetration distance. Well, how do we define penetration distance for a point? A natural way to define penetration distance is the distance along the minimum separation vector needed for the point to be outside the other shape. A 2D illustration of this definition is below.



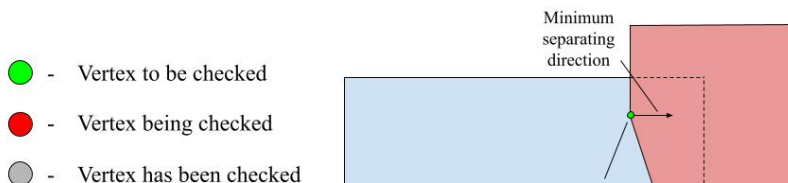
/Penetration\_Distance.jpg[Penetration\_Distance]

The penetration distance for a point can be found rather simply using Embree; we can cast a ray with the point as the origin in the minimum separating direction, and the distance until this ray hits the other shape is the penetration distance. So, to see if a point is at the maximum penetration distance, we need 2 things: the minimum separating direction and the maximum penetration distance. Luckily for us, both of these things can be found with the prior step, the call to `ccdGJKPenetration`. `ccdGJKPenetration` also provides us with the point at which the penetration is the deepest! We can use this point (or rather, the closest vertex on the shape to this point) as a starting point for our traversal algorithm.

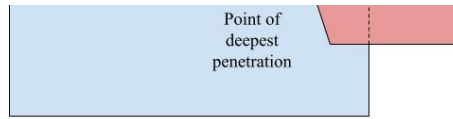
The traversal algorithm is relatively simple and should be pretty intuitive. We perform the following steps:

1. Define a list of contact points, a set of vertex indices that we've checked, and a queue of vertices that we need to check.
2. We start at the deepest vertex determined by `libccd`. Add it to the set of checked vertices.
3. We cast an initial ray from this point along the minimum separating direction, and see if it intersects with the other shape.
  - a. If it doesn't, then there are no penetrating points (meaning there are no contact points), so return.
  - b. If it does intersect, check the penetration distance. If it is the maximum penetration distance, we have a maximum penetration point, so add it to the list of contact points.
4. Add adjacent vertices, with the penetration distance of this vertex, to the queue of vertices to check later. Add these vertices to the set of checked vertices.
5. While queue of vertices to check isn't empty:
  - a. Cast a ray from vertex at front of queue, along the minimum separating direction.
  - b. If it intersects:
    - i. Find the penetration distance. If it is the maximum penetration distance, we have a maximum penetration point, so add it to the list of contact points.
    - ii. Compare the penetration distance to the penetration distance of the previous adjacent vertex. If the current penetration distance is  $\geq$  the previous penetration distance:
      - A. Add this vertex's adjacent vertices that are not already checked, along with the penetration distance of this vertex, to the queue of vertices to check later. Add these adjacent vertices to the set of checked vertices.
  - c. If it doesn't intersect:
    - i. This means the current point is outside the other mesh. So, there is an intersection point somewhere between this vertex and its adjacent vertices.
    - ii. For each adjacent vertex:
      - A. Cast a ray from the adjacent vertex to the current vertex.
      - B. If it intersects the other shape, find the intersection point. Otherwise, continue onto the next adjacent vertex.
      - C. Cast a ray from this intersection point along the minimum separating direction.
      - D. Find the penetration distance. If it is the maximum penetration distance, we have a maximum penetration point, so add it to the list of contact points.

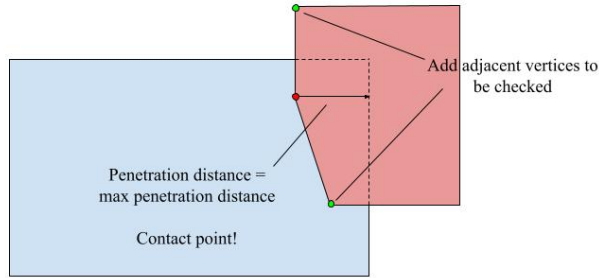
Perhaps it is better explained with a visual example.



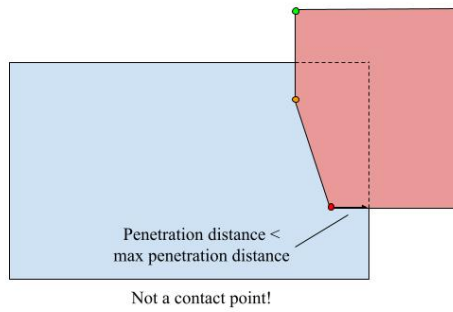
● - Contact point



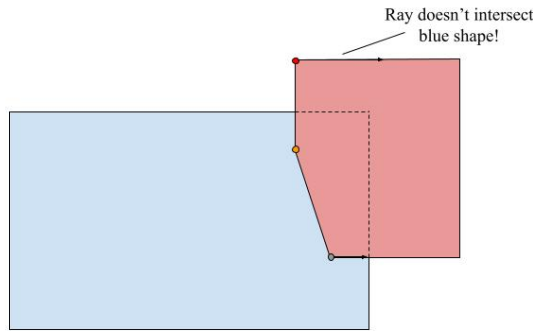
1. RESULTS OF HOCU



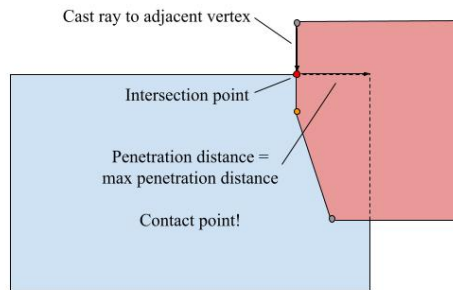
2. Cast a ray from the deepest vertex. It intersects the blue shape, so add its adjacent vertices to be checked. The penetration distance is the max, so add it to the list of contact points.



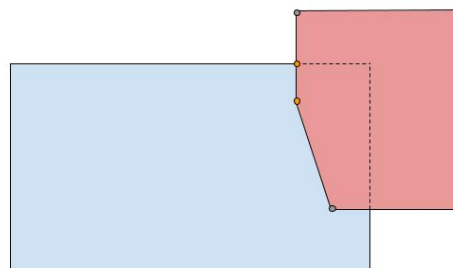
3. Cast a ray from the next vertex in the queue. The penetration is less than the previous vertex, so don't add its adjacent vertices to be checked. The penetration distance is less than max, so it is not a contact point



4. Cast a ray from the next vertex in the queue. The ray doesn't intersect the blue shape, so don't add its adjacent vertices to be checked.



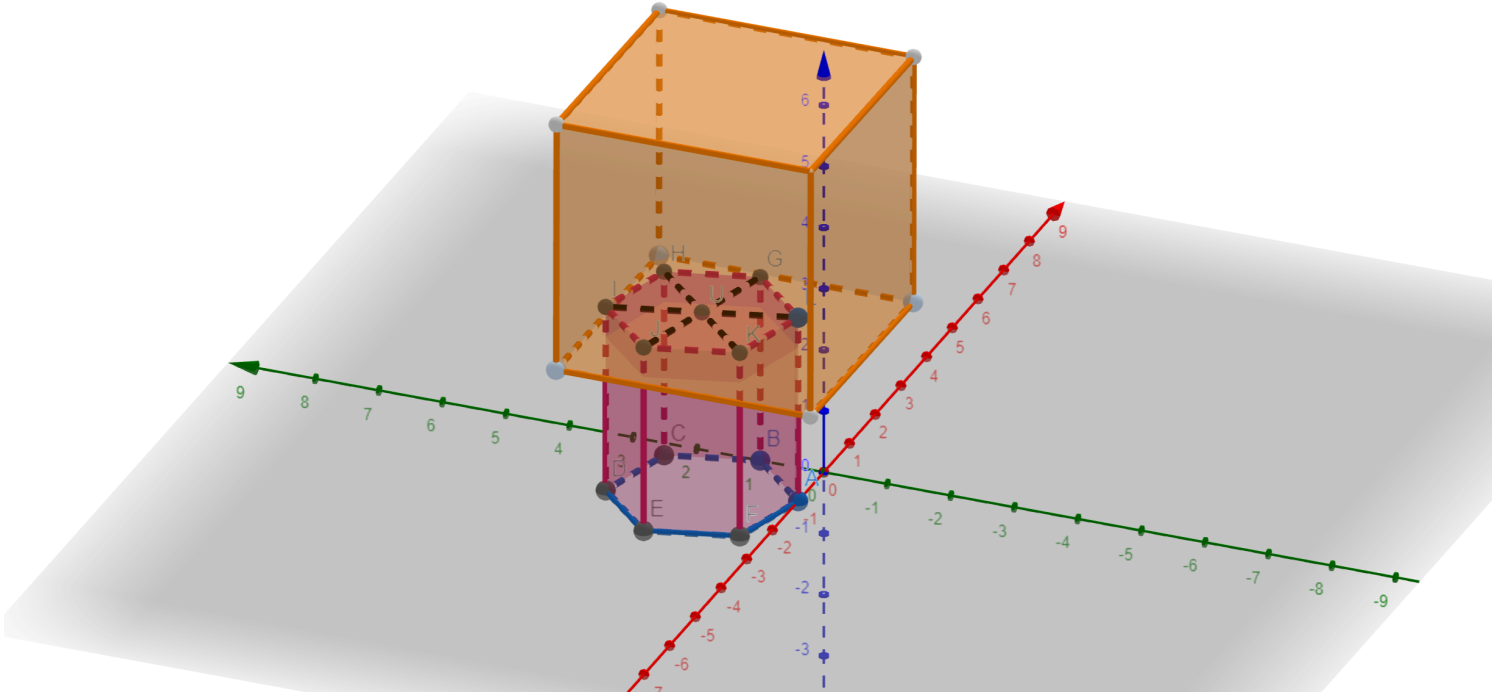
5. Cast a ray from the vertex to its adjacent vertex, and find the intersection point. Cast a ray from the intersection point to find the penetration distance. The penetration distance is the max, so it is a contact point



6. Final result.

#### 19.5.1.6.4. Find convex hull of contact points

Ultimately, we only want to keep the contact points that are absolutely necessary. In certain types of collisions, there might be extraneous contact points (possibly due to the number of subdivisions in the mesh), that don't actually add anything to the shape of the contact manifold. For this reason, it's necessary that we filter out these extraneous points by computing the convex hull. An example of this can be seen below, where the point in the center of the hexagon is extra and does not need to be a contact point.



The algorithm we choose to do this is the simple but effective **Graham Scan** algorithm, implemented in code [here](https://github.com/MiguelVieira/ConvexHull2D/blob/master/ConvexHull.cpp) (https://github.com/MiguelVieira/ConvexHull2D/blob/master/ConvexHull.cpp). This algorithm operates in 2D, however, so first we must project the contact points from 3D to 2D. There is no information loss here since all the points already lie in the same plane by the nature of a contact manifold. After the Graham Scan algorithm is completed, some post-processing steps also occur to remove duplicates and collinear points, to ensure that only the essential contact points are kept.

#### 19.5.1.6.5. Use contact points for each mesh to determine final contact points

We perform the above steps for both meshes involved in the collision, and arrive at two sets of potential contact points. Now we have to make a decision: which set of contact points to go with? Whichever set of contact points we choose, we can project the points from one mesh along the minimum separating direction to the surface of the other mesh, to get the contact position for the other mesh.

In all cases, we select the mesh with more contact points, and project those onto the other mesh. Each contact point in one mesh usually has a corresponding contact point counterpart in the other mesh. However, when there is a contact point without a counterpart (i.e. one mesh has more contact points than the other), then we need to use the one with more contact points to capture the collision information accurately. If two meshes have the same number of contact points, we arbitrarily pick one meshes set to go with.

##### 19.5.1.6.5.1. Dealing with edge-edge collisions

If neither mesh has contact points, then a special type of collision has occurred: an edge-edge collision. An illustration of this can be seen a few posts up. In these types of collisions, there should be a single contact point, but unfortunately it is in the middle of an edge, so the algorithm that we use to determine contact points fails. `libccd` struggles with this case as well, as (at least I believe) EPA yields indeterminate results when this case occurs.

So, what we do is find the "deepest" edge for each mesh. I define the deepest edge as the edge in the mesh that is furthest into the other mesh. More rigorously, the deepest edge is the edge whose midpoint has the deepest penetration. We can then move one of these deepest edges along the minimum separation vector. Where these edges intersect is where the contact point should be.

It should be noted that this "deepest edge" technique is not necessarily the best way to do this. The problem is a bit tricky to solve efficiently, because the edge on which the contact point should lie may not penetrate the shape at all (which is what we see in the edge-edge `DBulletTest` test case). Ideally, the contact point determination process would handle this natively, but I was unable to figure out a good way to do that.

#### 19.5.1.7. Setting up `rtcCollide` to do broadphase collision detection

The broadphase collision detection is done using Embree's `rtcCollide`. Each `MeshGroup` object has an Embree scene where this collision detection is run. An important note is that `MeshGroup`'s scene only contains the overall mesh bounding boxes, not the primitives that make up the mesh (this was what I was doing initially). Doing this drastically speeds up the broad phase, and makes committing geometries to the scene a lot faster as well. As before, `rtcCollide` requires us to specify geometry of the type `RTC_GEOMETRY_TYPE_USER` for it to work.

##### 19.5.1.7.1. Adding a mesh to the scene

Adding a mesh is fairly straightforward. First, we create and attach a new `RTCGeometry` object which returns its ID in the scene. We store both the `RTCGeometry` object and the geometry ID, as well as the pointer to the `DMesh` object in a map in `MeshGroup`. Then, we specify the user primitive count using `rtcSetGeometryUserPrimitiveCount`. This is just 1, since we only have one "user primitive": the entire mesh. Then, we specify the pointer to the user data using `rtcSetGeometryUserData`, which is just the raw pointer to the mesh. Then, we specify the

geometry bounds function using `rtcSetGeometryBoundsFunction`. The bounds function we pass in is super simple, it just sets the Embree bounding box of the mesh. Finally, we can commit and release the geometry with `rtcCommitGeometry` and `rtcReleaseGeometry`.

#### 19.5.1.7.2. Updating the meshes in the scene

Before every call to `rtcCollide`, we must tell Embree which meshes have moved before it rebuilds the internal BVH. This is done using `rtcCommitGeometry`. All we have to do is pass each geometry object (or at least just the ones that have had their meshes move in the scene) to `rtcCommitGeometry`, and it will tell Embree that the geometry has been modified.

Then, to actually rebuild the BVH, we call `rtcCommitScene`, which will rebuild the BVH. Now we are ready to call `rtcCollide`.

#### 19.5.1.7.3. Calling `rtcCollide`

Calling `rtcCollide` is very simple, but because we call it within a class there are some important technicalities. We intersect the `MeshGroup` scene with itself to do the broad phase collision detection, that is nothing different from before. A more interesting thing, however, is that the collide function callback that we pass in needs to be a static function to work properly (there might be some contrived way to make it a member function using `std::bind`, but it's easier just to make it a static function). Because it is a static function, we need some way of getting back to the specific `MeshGroup` in question. Hence, we pass a pointer to the caller `MeshGroup` (using the `this` keyword), as the user pointer, so in the collide function callback, we know which `MeshGroup` called `rtcCollide`.

The collide function callback is simple; all we do is add each potentially intersecting pair of geometry IDs to the caller `MeshGroup` using the public method `MeshGroup::addCollision`. Then, the `MeshGroup` object will go through these pairs of potentially intersecting pairs and run the narrow phase collision detection (described in a post above) to see if they actually intersect, and if they do find the contact points associated with the collision.

#### 19.5.1.7.4. Removing a mesh from the scene

A mesh is removed from the scene using a call to `rtcDetachGeometry`, and then recommitting the scene. This should be all you need to do. However, for whatever reason **this does not seem to work, and Embree still detects collision between geometries that have been detached**. I experimented briefly with trying to figure it out, but was unable to determine the cause. For now I just check if a colliding geometry ID added by `rtcCollide` currently exists in the mesh-geometry map in `MeshGroup`. If it doesn't, then I know that the mesh and geometry has been deleted from the scene, and to skip it.

#### 19.5.1.8. DMeshScene ecosystem

I thought it might be useful to spend a little time documenting how things fit together at the application level. When creating and colliding geometries in `DMeshScene`, there is a lot of moving parts, and a lot that is going on in multiple classes/files. Here is the test script that we will be breaking down, which collides two cubes in a plane-plane intersection:

```
1  from Math.SOA_Py import SOAVector3, SOAQuaternion, SOAHomTran, SOARotationMatrix
2  from DScene import DScene_Py
3  from DScene import DMeshScene_Py
4
5  cscene = DMeshScene_Py.DMeshScene()
6
7  sf = cscene.createSceneFrame("root")
8
9  little_cube = cscene.createPartGeometry(DScene_Py.SceneObject.PHYSICAL, "little_cube")
10 little_cube.cube(1.0, 1.0, 1.0)
11 little_cube.relTransform(SOAHomTran(SOARotationMatrix(0, 0, 0), SOAVector3(1.25, 4.28, 0)))
12 sf.attachSceneObject(little_cube)
13
14 big_cube = cscene.createPartGeometry(DScene_Py.SceneObject.PHYSICAL, "big_cube")
15 big_cube.cube(2.0, 2.0, 2.0)
16 big_cube.relTransform(SOAHomTran(SOARotationMatrix(0, 0, 0), SOAVector3(0, 3, 0)))
17 sf.attachSceneObject(big_cube)
18
19 cscene.detectCollision()
20 nc = cscene.getNumContacts()
21
22 contactInfo = cscene.getContactInfo()
```

##### 19.5.1.8.1. Creating scene and scene frame

**Line 5:** `cscene = DMeshScene_Py.DMeshScene()`

This creates the `DMeshScene` object. The constructor of `DMeshScene` initializes protected member `_mesh_group`, among other things related to setting up the `DMeshScene` (like calling base constructors). The initialization of `_mesh_group` makes a call to the `MeshGroup` constructor, which sets up the Embree scene and the `libccd` object.

**Line 7:** `sf = cscene.createSceneFrame("root")`

This creates the root `DMeshSceneFrame` that geometries will be added to. This class is mostly just a wrapper around `SceneFrame`. The constructor initializes the `_rel_transform` and `_abs_transform` private members to a 0 transform, and then calls the parent constructor for `SceneFrame`.

##### 19.5.1.8.2. Creating part geometry

**Line 9:** `little_cube = cscene.createPartGeometry(DScene_Py.SceneObject.PHYSICAL, "little_cube")`

This creates the `DMeshPartGeometry` object for the smaller of the two cubes that we will be colliding. The mesh is not created yet, just the scene object. The call to `createPartGeometry` calls the `DMeshPartGeometry` constructor, which sets up the scene object by calling its parent constructors. Most notably, `DMeshPartGeometry` inherits from `DMeshSceneObject`, which actually holds a pointer to the `DMesh` object in the protected member `_mesh`.

**Line 10:** `little_cube.cube(1.0, 1.0, 1.0)`

This actually creates the mesh for the little cube. `DMeshPartGeometry` contains mesh construction methods for numerous primitives that are implemented by `DMesh`, such as a sphere, cylinder, cone, and cube. The cube construction method calls the `DMesh::constructBox` method, which returns a shared\_ptr to a newly created box mesh, and sets the `_mesh` field to it. Then, it calls `DMesh::computeAdjacentVertices`, which will compute vertex adjacency information for the new mesh. This is an important (but often costly) pre-processing step for collision detection, so we do on mesh creation and then don't have to worry about it again. Then, it calls `DMesh::checkEmbree(true)`, which will initialize the mesh's Embree wrapper. This is important to do for collision detection, since we will be intersecting rays with the mesh to determine contact points.

Lastly, it calls `DMeshScene::_storeMesh` for its parent scene. This method is responsible for adding the `DMeshPartGeometry` and `std::shared_ptr<DMesh::Mesh>` pair to its private member, `_meshes_map`, which is a map of mesh pointers to part geometry pointers. The point of `_meshes_map` is to output the appropriate `DMeshPartGeometry` object given a pointer to a mesh. This is important to keep track of to interface properly with `MeshGroup`, since `MeshGroup` exclusively deals with meshes, and `DMeshScene` almost exclusively deals with scene

objects. Also in `DMeshScene::_storeMesh` is the call to `MeshGroup::_addMesh`, which adds the newly created mesh to the `MeshGroup`. This method is responsible for creating and attaching the Embree geometry to the Embree scene contained in `MeshGroup` that is used for broad phase collision. After creating the geometry, it will store the mesh, geometry, and geometry ID in a map.

**Line 11:** `little_cube.relTransform(SOAHomTran(SOARotationMatrix(0, 0, 0), SOAVector3(1.25, 4.28, 0)))`

This line applies a transform to the mesh. What is actually called here is `DMeshSceneObject::relTransform` which just sets the relative transform, `_rel_transform` to whatever is specified by the user. It also sets the boolean flag `_needs_update` to true, which indicates that the mesh needs to be updated.

**Line 12:** `sf.attachSceneObject(little_cube)`

This just adds the newly created scene object to the scene frame using the parent method `SceneFrame::attachSceneObject`.

### 19.5.1.8.3. Running the collision detection

**Line 19:** `cscene.detectCollision()`

This makes the call to `DMeshScene::detectCollision`, which first iterates through all the part geometries stored in `_meshes_map` and updates their meshes using `DMeshSceneObject::updateMesh`. What `updateMesh` does is first check if `_needs_update` is true. If so, then it goes ahead and transforms its `_mesh` field using `Mesh::transformInPlace`. The transform that is applied to the mesh is the relative transform between the mesh's current transform (which is out of date) and the scene object's current transform. That way, the mesh ends up with the same transformation as its parent scene object.

Then, `detectCollision` makes the call to `MeshGroup::detectCollision`, which performs the full collision detection process detailed in posts above. It will commit the geometries to its internal Embree scene, then run `rtcCollide` for broad phase, and then use `libccd` for narrow phase. The contact points `MeshGroup` finds will be stored internally in a vector and can be queried using `getNumContacts` or `getContactInfo`.

**Line 20:** `nc = cscene.getNumContacts()`

This just simply calls the scene's `_mesh_goup`'s `getNumContacts` method, which returns the current size of the contact points vector from the last run of `detectCollision`.

**Line 21:** `contactInfo = cscene.getContactInfo()`

This gets all current contact points by making a call to the `MeshGroup::getContactInfo` method from `_mesh_group`. This will return a list of `Mesh::ContactPoint` structs, which mirror the `DScene::CollisionContactInfo` struct, but store meshes instead of objects. The reason for this is that `MeshGroup` (at least currently) does not store scene object to mesh mappings, so there is no way for it to go from a mesh back to its associated scene object. `DMeshScene` does however, and it can get the associated scene object from a mesh pointer by looking in `_meshes_map`. This is done in `DMeshScene::_getPartGeometry`.

`DMeshScene` iterates through the contact points returned by `MeshGroup::getContactInfo`, and create analogous `DScene::CollisionContactInfo` structs, substituting the scene object pointers for the mesh pointers. Also, depending if the `origin` option is disabled, it will transform the normals and positions of the contact points to be relative to the scene object coordinate frame. By default, `MeshGroup::getContactInfo` returns contact information with respect to the global coordinate frame.

### 19.5.1.9. Some extra tidbits I think are important or should be clarified

#### 19.5.1.9.1. More specifics on libccd return values

The call to `ccdGJKPenetration` is documented [here](https://libccd.readthedocs.io/en/master/examples.html#gjk-epa-penetration-of-two-objects) (<https://libccd.readthedocs.io/en/master/examples.html#gjk-epa-penetration-of-two-objects>). The function call looks like this:

```
int intersect = ccdGJKPenetration(obj1, obj2, &ccd, &depth, &dir, &pos);
```

`intersect` takes on the value -1 if there isn't a collision and 0 if there is a collision (despite them saying `intersect` is "true" when there is a collision). `obj1` and `obj2` are the pointers to the mesh, and `ccd` is the `libccd` object that we initialize earlier (in `MeshGroup` it is stored as a class variable and initialized in the constructor). `depth` is the max penetration depth, `dir` is the minimum separation direction, and `pos` is, in their words, "the position in global coordinate system". This is pretty vague and unhelpful in describing what this `pos` actually is.

The [GitHub post](https://github.com/danfis/libccd/issues/73) (<https://github.com/danfis/libccd/issues/73>) asks a similar question and provides a tentative answer. I arrived at a similar conclusion as he did: **the pos is the midpoint on the line between a contact point's positions on each shape**. For example, if we move from `pos` along the minimum separation direction for half the max penetration distance, we will arrive at the deepest penetrating vertex for one of the shapes. In a more concrete example, say we have two spheres with radius 1, one at (0,0,0) and one at (1.8,0,0). The penetration distance of the collision is 0.2, and there is only one contact point with a `positionA` of (1,0,0) and a `positionB` of (0.8,0,0). The `pos` returned by `libccd` in this case is (0.9,0,0), halfway between `positionA` and `positionB`.

In code, it looks like the following:

```
const SOAVector3 ccd_pos0 = position - 0.5 * max_pen * ray_dir;
const SOAVector3 ccd_pos1 = position + 0.5 * max_pen * ray_dir;
```

Where `ccd_pos0` is a deepest penetrating point on the surface of mesh 0, and `ccd_pos1` is a deepest penetrating point on the surface of mesh 1. `ccd_pos0` and `ccd_pos1` is **not necessarily a vertex** on either of the meshes, depending on the type of collision. That is why I do a `computeClosestVertex` call to get the closest vertex to the `ccd_pos` as the starting point for the vertex traversal algorithm.

#### 19.5.1.9.2. Explanation of ray casting for mesh-level Embree scene

When a mesh is first created or loaded in using the `DMeshPartGeometry` class, the Embree wrapper is initialized using a call to `mesh->checkEmbree(true)`; This will create an Embree scene, and commit the current mesh to the scene, resulting in it building the BVH out of **triangle mesh primitives** for the ray tracing interface. At this point, Embree thinks the mesh is at the origin, and the mesh really is at the origin; the mesh hasn't been transformed yet.

Since we are assuming the mesh to be a rigid body, the spatial relationship between the primitives will not change, and as such we don't really **need** to rebuild the mesh-level Embree wrapper Embree scene. If we keep track of the current transform of the mesh and we want to cast intersecting rays using Embree, we can just transform the ray we want to cast back to the origin coordinate system (using the inverse of the current mesh transform).

When we transform the mesh, we are actually applying the transform to the physical vertices and faces. However, since the Embree wrapper Embree scene has already been committed (and the BVH has already been constructed), as far as the Embree scene is concerned, the mesh is still at the origin. In order to update this Embree scene, we have to explicitly make a call to `rtcCommitScene` for it to rebuild the BVH, and only then will it look at the actual mesh positions and figure out how to reconstruct the BVH at these new positions. As mentioned earlier, **we don't need to do this**, since we can transform rays back to the origin coordinate frame and cast them there.

But, and here's where it gets a little weird, the ray intersection point that Embree returns is when you cast a reverse-transformed ray towards the mesh is **what it would be if the Embree scene had been updated**. (I have no idea why this happens, but my guess is that Embree internally keeps pointers to the mesh geometry, so is able to somehow convert from Embree scene coordinates to mesh coordinates).

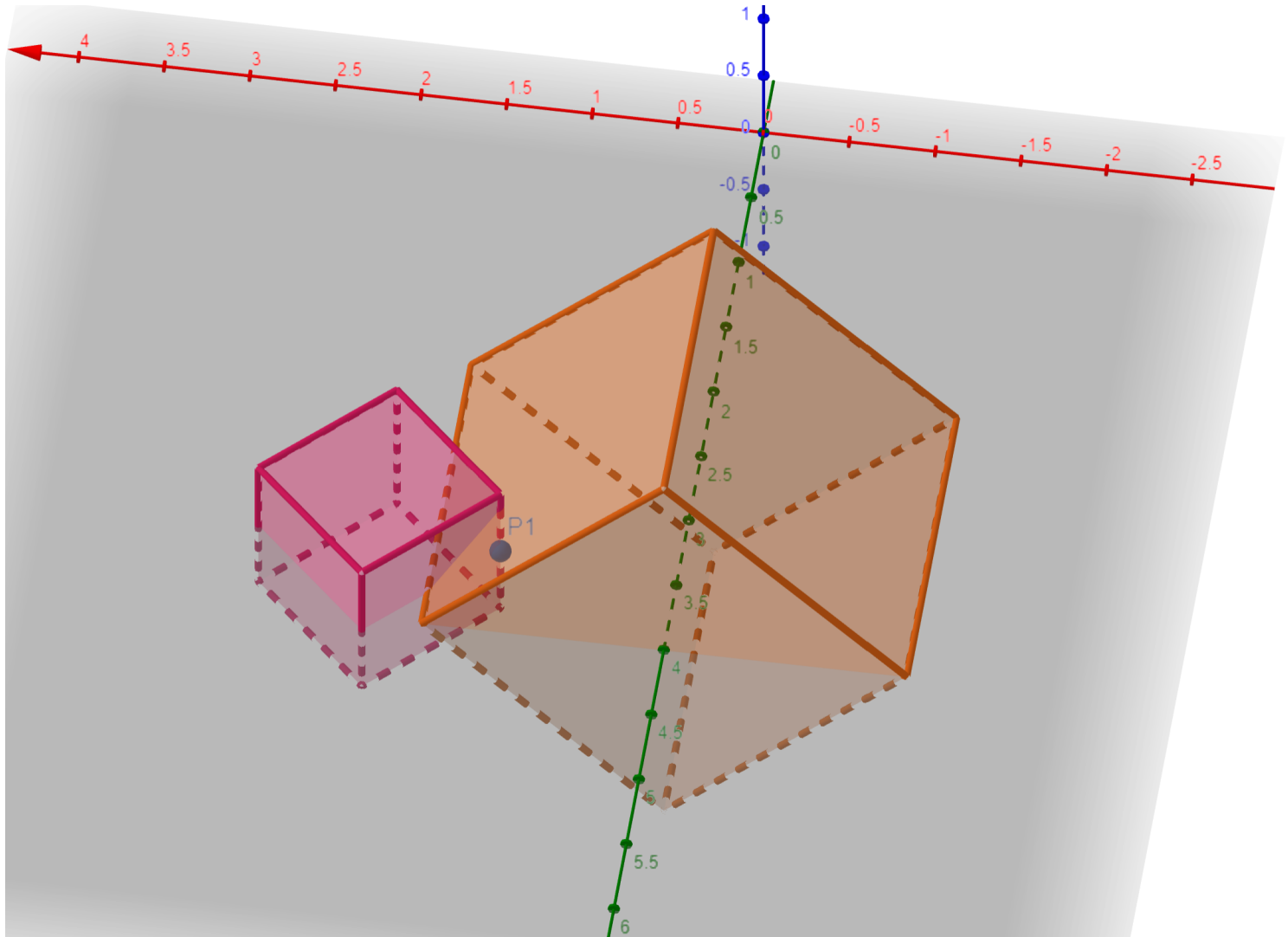
It's probably just best to see what I do in the code to understand how to approach this. Here, we want to cast a ray from `closest_vert` towards `mesh1` in the direction of `norm_ray_dir`. What we have to do to transform `closest_vert` into `mesh1`'s origin coordinate frame, and use the inverse quaternion's rotation to point `norm_ray_dir` in the correct direction.

```
// get inverse transform of mesh1
const SOAHomTran& inv_mesh1 = mesh1.getCurrentTransform().getInverse();
// get norm_ray_dir wrt mesh1 origin
const SOAVector3& rot_norm_ray_dir = inv_mesh1.getQuaternion().getRotation() * norm_ray_dir;
// construct the ray
const SOARay ray(inv_mesh1 * closest_vert, rot_norm_ray_dir);
// get the intersection hit point -- this will not be in mesh1 origin frame!
// i.e. we won't have to transform it back to the current mesh1 transformation to use it
std::optional<SOAVector3> hit_point = mesh1.computeRayIntersection(ray);
```

Ultimately, all of this is done with efficiency in mind. Since the meshes are rigid bodies, we really don't have to update the Embree scene, which is a costly operation when there are a lot of vertices in a mesh.

### 19.5.1.9.3. More on the edge-edge intersection case

The edge-edge intersection case is seen below.



As we can see, there are **no vertices or intersection points that have the deepest penetration on either shape!** That's what makes this case so hard. Normally, there is at least one vertex or intersection point on one of the shapes that has the deepest penetration, and we're able to find that using the vertex traversal algorithm. Also, the `libccd_pos` is a bit unreliable here, as I think the EPA algorithm relies on the vertices (and doesn't directly have edge information) to figure out the `pos`. So when there is no deepest vertex on either shape, it struggles to converge. That being said, `libccd` still provides accurate minimum separating direction and maximum penetration distance.

The strategy I use to find the elusive contact point is by finding the "deepest edge". This means finding the edge on the shape that is furthest along the minimum separating direction. In the example above, this would be the rightmost vertical edge on the red shape and the leftmost horizontal edge on the orange shape. Logically, the contact point is where these edges touch when the shapes first touched. So, we translate one of the deepest edges along the minimum separating direction such that these edges now intersect. Where they intersect is the contact point.

The reason I put so much emphasis on bringing this case up is because it is untested other than the case shown above. My worry is that the "deepest edge" strategy I devised is not general enough to handle all the potential edge-edge cases out there. I think it should be able to handle most of them, but there might be some corner cases I haven't thought of. That being said, I imagine that this intersection case is pretty rare to encounter in an actual simulation, especially if the meshes have large numbers of vertices.

## 20. DBullet

### 20.1. Background

#### 20.1.1. Reference & Source material

- [DBullet Doxygen docs](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/DBullet/html/) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/DBullet/html/)
- [Team talks](https://dlabportal.jpl.nasa.gov/resources/darts-lab-talks/) (https://dlabportal.jpl.nasa.gov/resources/darts-lab-talks/)
- [2019, 2020 course slides](https://dlabportal.jpl.nasa.gov/documentation/coursetutorial-links/) (https://dlabportal.jpl.nasa.gov/documentation/coursetutorial-links/)
- [Sphinx docs](https://dartslab.jpl.nasa.gov/dlabdocs/) (https://dartslab.jpl.nasa.gov/dlabdocs/)
- [Jupyter notebooks](https://dlabnotebooks.jpl.nasa.gov/hub/login) (https://dlabnotebooks.jpl.nasa.gov/hub/login)
- [DARTS Lab QA site](https://dartslab.jpl.nasa.gov/qa/) (https://dartslab.jpl.nasa.gov/qa/)

### 20.2. Design

#### 20.2.1. MeshToBullet2

`MeshToBullet2` was designed to meet the following requirements:

1. Provide interfaces for the VHACD-4.0, VHACD-2.0, and HACD algorithms.
  - These interfaces should utilize `DMesh` for importing/viewing meshes. That way, `MeshToBullet2` can support all file types that `DMesh` supports.
2. Support files that contain multiple meshes.
  - This includes importing and creating convex hulls for each mesh, as well as merging output convex hulls files into one file.
3. Include visualization of the result that optionally includes: the original mesh, the output convex hull mesh, a wireframe view of the convex hulls.

##### 20.2.1.1. Convex decomposition algorithm interfaces

VHACD-4.0, VHACD 2.0, and HACD come with interface classes that can be utilized to setup and run the algorithms. However, these interfaces are limited in terms of what types of mesh files they can import, even though the algorithms themselves only require the points/faces of the input mesh. The `ConvexDecomUtilities` library in `DBullet` writes wrappers for these interfaces that utilize `DMesh`; this allows the wrapped versions to utilize any mesh file type that can be imported into `DMesh`. In addition, the original interfaces provide hooks for data logging and data callback classes. The wrapped versions utilize these to implement logging in C++, and use swig directors so that the callbacks can be optionally implemented in Python. In `ConvexDecomUtilities` the default callbacks are ultimately implemented in Python, and utilize the `tqdm` module to show a status bar that indicates current progress.

The VHACD 4.0, VHACD 2.0, and HACD implementations in `ConvexDecomUtilities` also come equipped with methods to extract the convex hull decomposition as a convex hull file, and as a list of `DMesh::Mesh`'s. These methods are useful for the visualization.

##### 20.2.1.2. Visualization

The original mesh and convex hull meshes utilize the `DMesh::Mesh::showList` method to visualize them. The wireframe view of the convex hulls is achieved by creating a part geometry using the output convex hulls file from the convex decomposition, and turning on the Bullet debugger. To make the best use of three visualization components, original mesh, wireframe view of convex hulls, and convex hulls mesh, the following strategy is used:

- If the original mesh and wireframe are both turned on, then they are positioned so that they lie on top of one another. This allows the user to easily view the difference between the two.
- If the original mesh is not enabled, but the convex hulls mesh is enabled, then the wireframe view is positioned on top of the convex hulls mesh.
- Otherwise, the wireframe mesh sits at the origin. If the original mesh and convex hulls meshes are both enabled, then they are automatically position so they do not overlap.

#### 20.2.2. MeshToSDF

`MeshToSDF` was designed to meet the following requirements: 1. Create a signed data function (SDF) file for all mesh types supported by `DMesh`. At this time, `MeshToSDF` is only required to handle files that have one mesh per file. 2. Visualize the SDF alongside the original mesh so the user can compare the two.

##### 20.2.2.1. SDF algorithm interface

The `Discregrid` (https://github.com/InteractiveComputerGraphics/Discregrid) third-party package provides an algorithm to create an SDF given a mesh represented using its internal data structure. An interface class was written to take a `DMesh`, convert it to the mesh structure required by `Discregrid`, and use the `Discregrid` algorithm to create an SDF file.

##### 20.2.2.2. Discregrid

This section contains a brief summary of `Discregrid`. `Discregrid` divides up the domain based on the user-provided resolution. Then, for each cell in the domain, it queries the distance to the mesh. This information is saved for each cell. Then, a cubic interpolation scheme is used to interpolate the distance for any point in the domain.

The SDF file is created by simply serializing the cubic interpolation function and required data.

In addition, the gradient of the SDF can be approximated, which `Bullet` uses internally get the normal of the collision.

##### 20.2.2.3. Visualization

`CGAL` has the capability to generate a 3D surface mesh given a function. This is known as [meshing via implicit functions](https://doc.cgal.org/latest/Surface_mesher/index.html) (https://doc.cgal.org/latest/Surface\_mesher/index.html). This interface is utilized to construct a `CGAL` mesh from the SDF. Then, the mesh is converted into a `DMesh` for visualization.

### 20.3. Usage

#### 20.3.1. Collision geometry



TBD: Add sections to this, e.g., primitives. Also, flush out subsections below.



### 20.3.1.1. Convex hulls

Using a convex hulls representation of a mesh is the most common way to do collision detection. In DARTS, the `MeshToBullet2` tool provides a quick and easy way to generate a convex hulls file for use in `DBullet`: see the sections on `MeshToBullet2` for more details.

In sims, the `extra_parameters` fields of part geometries or the `filenames` parameter of the geometries field of body parameters dictionaries can be used to specify a given convex hulls file. Note, if no file is specified, then `DBullet` will generate one for you using the HACD algorithm; however, this often is not very efficient compared with the convex hulls files that can be generated by `MeshToBullet2`.

### 20.3.1.2. Signed data fields

Signed data fields are an alternative way to specify objects for collision detection. They utilize a function whose sign dictates whether an object is penetrating or not. Signed data fields should ONLY be used for static meshes. Moreover, `bullet` cannot handle collisions between two SDFs, which is in part why they should only be used for static meshes, e.g., terrains.

See the benchmark located here for details: `DBulletTest/Latest/benchmark/test_SDF_collision/script.py`.

### 20.3.2. MeshToBullet2

`MeshToBullet2` comes equipped with three convex decomposition algorithms: VHACD 4.0, VHACD-2.0, and HACD. The top-level `--algorithm` option can be used to switch between these. VHACD-4.0 is used by default, since it is the newest of the three algorithms, and typically gives better results

The `files` command is used to set the names of the input and output files. An input file must be specified in order for `MeshToBullet2` to run. If no output file is specified, then the default name will be the same as the input file, but with the `.convexhulls` extension.

The `params` command is used to modify the settings of the desired algorithm. The options of this command depend on the algorithm used. The options' help strings can be used to learn how the option affects the algorithm. In addition, the help text for the `param` command includes links for articles that explain the algorithm in detail.

The `show` command is used to modify the visualization settings. There are three visualization elements that can be enabled/disabled: \* The original mesh \* The convex hulls mesh \* The convex hulls wireframe See the visualization design section for more information on how these are displayed relative to one another. For example outputs, see the images below:

 crate convex hull

 torus convex hull

#### 20.3.2.1. Common pitfalls

The following is a list of common problems, and resolutions:

- The algorithm runs, and the visualization shows, but it appears frozen. This often happens when the number of convex hulls is large. Disabling the wireframe view, using `show --wireframe False`, typically resolves this issue.
- The HACD algorithm provides many convex hulls with few points per hull. This can be resolved to some extent by using the `--concavity` option. Increasing the value of this option will sometimes resolve this issue. However, if the issue persists, then consider switching to the VHACD4 or VHACD2 algorithm.

#### 20.3.2.2. Additional resources

- [These](https://docs.google.com/presentation/d/1OZ4mtZYrGEC8qffqb8F7Le2xzufiqaPpRbLHKKgTIM/edit?usp=sharing) (https://docs.google.com/presentation/d/1OZ4mtZYrGEC8qffqb8F7Le2xzufiqaPpRbLHKKgTIM/edit?usp=sharing) slides are a wonderful visual depiction of how the VHACD 4.0 parameters affect the convex decomposition.
- [Here](https://github.com/kmammou/v-hacd) (https://github.com/kmammou/v-hacd) is the VHACD 4.0 homepage.
- [Here](https://www.taylorfrancis.com/chapters/mono/10.1201/b21177-15/volumetric-hierarchical-approximate-convex-decomposition-eric-lengyel) (https://www.taylorfrancis.com/chapters/mono/10.1201/b21177-15/volumetric-hierarchical-approximate-convex-decomposition-eric-lengyel) is the original VHACD paper.
- [Here](http://khaledmammou.com/AllPublications/icip2009.pdf) (http://khaledmammou.com/AllPublications/icip2009.pdf) is the original HACD paper.
- [This](https://blog.roblox.com/2020/07/search-better-convex-decomposition/) (https://blog.roblox.com/2020/07/search-better-convex-decomposition/) article from Roblox shows a nice visual depiction of how the HACD algorithm works.
- [Here](http://kmamou.blogspot.com/2014/12/v-hacd-20-vs-hacd.html) (http://kmamou.blogspot.com/2014/12/v-hacd-20-vs-hacd.html) is a comparison between the VHACD 2.0 and HACD algorithms.
- Reg tests for `MeshToBullet2` are located at `DBulletTest/test/test-MeshToBullet2`.

### 20.3.3. MeshToSDF

The `MeshToSDF` tool comes with a `Dclick` interface that can be used to configure the creation/visualization of the `SDF` file.

The `files` command is used to set the names of the input and output files. An input file must be specified in order for `MeshToSDF` to run. If no output file is specified, then the default name will be the same as the input file, but with the `.sdf` extension.

The `params` command is used to modify the settings of the `SDF` algorithm. The options' help strings can be used to tell the user how and when to use each option. Of particular importance are the `resolution` and `invert` options. The main option to control the accuracy is the `resolution` option, which specifies the discretization of the domain into cells. The `invert` option is useful if the `SDF` algorithm is reading your mesh inside-out. For example, in the case of the hollow sphere below, the `invert` option was necessary to get the `SDF` to characterize the inside vs. outside of the mesh correctly:

No SDF inversion: the SDF mesh is inside out. `image::docFiles/hollow_cylinder_inside_out.png`

With SDF inversion: the mesh has the correct orientation. `image::docFiles/hollow_cylinder.png`

The `show` command is used to modify the visualization settings. There are two visualization elements that can be enabled/disabled: \* The original mesh \* The SDF mesh For example, see the image below:

 concave box

#### 20.3.3.1. Additional resources

- [This](https://doc.cgal.org/latest/Surface_mesher/index.html) (https://doc.cgal.org/latest/Surface\_mesher/index.html) is the CGAL documentation on creating a 3D surface mesh from an implicit function.
- [This](https://github.com/InteractiveComputerGraphics/Discregrid) (https://github.com/InteractiveComputerGraphics/Discregrid) is the third-party `Discregrid` library, which contains the algorithm used to create an SDF from a mesh.

- Reg tests for `MeshToSDF` are located at `DBulletTest/test/test-MeshToSDF`.

#### 20.3.3.2. Notes

- I have not exposed the tolerance used by CGAL's `Implicit_Surface_3` ([https://doc.cgal.org/latest/Surface\\_mesher/classCGAL\\_1\\_1Implicit\\_surface\\_3.html](https://doc.cgal.org/latest/Surface_mesher/classCGAL_1_1Implicit_surface_3.html)). This was not necessary for the meshes I tried it on, but may need to be exposed in the future.
- `Implicit_Surface_3` ([https://doc.cgal.org/latest/Surface\\_mesher/classCGAL\\_1\\_1Implicit\\_surface\\_3.html](https://doc.cgal.org/latest/Surface_mesher/classCGAL_1_1Implicit_surface_3.html)) claims that the implicit function must be  $< 0$  at the center of the bounding sphere, but this did not appear true in practice. I'm curious if this is an old piece of documentation. Note, I thought this was the original reason the hollow cylinder appeared inverted, but upon inspecting the SDF, without inversion, the SDF is  $< 0$  in the center, meaning it thinks the "hollow" portion is solid unless inversion is enabled.
- I did not expose the tag option of `make_surface_mesh` ([https://doc.cgal.org/latest/Surface\\_mesher/group\\_PkgSurfaceMesher3FunctionsMakeMesh.html#ga7e188adef5bfadaafd08db82c8a25dc1](https://doc.cgal.org/latest/Surface_mesher/group_PkgSurfaceMesher3FunctionsMakeMesh.html#ga7e188adef5bfadaafd08db82c8a25dc1)). This option was not necessary on the meshes I tried `MeshToSDF` on.

#### 20.4. Software

#### 20.5. Raw documents

## 21. NdartsConstraint

### 21.1. Background

#### 21.1.1. Reference & Source material

- [NdartsConstraints Doxygen docs](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/NdartsConstraints/html/) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/NdartsConstraints/html/)
- [Team talks](https://dlabportal.jpl.nasa.gov/resources/darts-lab-talks/) (https://dlabportal.jpl.nasa.gov/resources/darts-lab-talks/)
- [2019, 2020 course slides](https://dlabportal.jpl.nasa.gov/documentation/coursetutorial-links/) (https://dlabportal.jpl.nasa.gov/documentation/coursetutorial-links/)
- [Sphinx docs](https://dartslab.jpl.nasa.gov/dlabdocs/) (https://dartslab.jpl.nasa.gov/dlabdocs/)
- [Jupyter notebooks](https://dlabnotebooks.jpl.nasa.gov/hub/login) (https://dlabnotebooks.jpl.nasa.gov/hub/login)
- [DARTS Lab QA site](https://dartslab.jpl.nasa.gov/qa/) (https://dartslab.jpl.nasa.gov/qa/)

### 21.2. Design

### 21.3. Usage

### 21.4. Software

#### 21.4.1. General concepts

- The `pnode` or `virtual root body` for a compound body is effectively the **physical** parent body that the compound body is attached to (as returned by the `physicalParentBody()` method).
- `nbodies` below refers to the number of embedded physical bodies, and `nindepVels` to the number of independent dofs for the body's hinge.
- Recall that  $\phi(\text{parent}, \text{cbd}) = E_G \phi_G$ , where  $E_G^*$  is the `SKO` matrix element whose transpose propagates the physical parent body's spatial velocity to all the base bodies within the compound body, and  $\phi_G^*$  propagates the base body velocities to all the embedded body spatial velocities. Thus  $\phi^*(\text{parent}, \text{cbd})$  propagates the physical parent body's spatial velocities to the spatial velocities of all the embedded bodies.

##### 21.4.1.1. DartsCompoundBodySubhinge

###### 21.4.1.1.1. class CompoundSubhingeKinMatrices

Class for compound body kinematics matrices

| Member                 | Type      | Size                                    | Description   |
|------------------------|-----------|---|---|
| <code>m_Hst_RG</code>  | SOAMatrix | $6 * \text{nbodies}, \text{nindepVels}$ | This is $H_{RG}^* = H_G^* * X_G$ element for the compound body. Each of its 6-row contains the mapping from the independent generalized vels to the <b>body frame</b> relative spatial velocity for the physical row body. This is also the compound body's $H^*(k)$ element in the $H^*$ spatial operator. Note that it includes the $X_G$ Jacobian that maps the independent gen vels to the full gen vels for all the embedded bodies.   |
| <code>m_E_Phi_G</code> | SOAMatrix | $6 \times 6 * \text{nbodies}$           | the $E_G * \phi_G$ product matrix for the compound body. This is also the $\phi(\text{parent}, \text{cbd})$ matrix contribution to the <code>SKO</code> matrix for the system. Its elements are the rigid body transformation $6 \times 6$ matrix from the physical parent body to the <code>pnode</code> for each embedded body. Thus the elements map from the individual embedded body frames to the physical parent frame. The transpose of this matrix maps the spatial velocity of the physical parent body to the individual embedded body spatial velocities. |

###### 21.4.1.1.2. class CompoundSubhingeABIMatrices

Class for compound body ABI matrices derived from `SubhingeABIMatricesBase`.

Note that the  $P_G$  ATBI operator element for a compound body is simply a  $(6 \text{ nbody})$  size, symmetric and square block diagonal matrix with the  $6 \times 6$  ATBI `PR` matrix block elements for the individual embedded bodies along the diagonal. Each of these block elements is in its own `pnode` frame. Note that the individual embedded body `PR` matrix is obtained by a regular ATBI step where it accumulates the `external` children body ATBI contributions with its spatial inertia - and **does not** include any contributions from other children bodies that are siblings within the same parent compound body.

| Member | Type | Size | Description |
|--------|------|------|-------------|
|--------|------|------|-------------|

| Member   | Type      | Size                    | Description   |
|----------|-----------|-------------------------|---|
| m_PHst_G | SOAMatrix | 6 nbodies x nindepVels  | the ATBI $P_G * H_{RG}^*$ matrix for the compound body. This intermediate quantity is kept around since $P_G$ is a block-diagonal, and due to its sparsity we do not want to be multiplying full matrices. The individual 6x6 row matrix elements are the product of the PR matrix for the individual embedded body with the joint map matrix contribution for the same body. |
| m_D_G    | SOAMatrix | nindepVels x nindepVels | the ATBI $D_G = H_{RG} P_G H_{RG}^*$ operator element for the compound body   |
| m_Dinv_G | SOAMatrix | nindepVels x nindepVels | the ATBI $D_G^{-1}$ operator element for the compound body  |
| m_G_G    | SOAMatrix | 6 nbodies x nindepVels  | the ATBI $G_G = P_G H_{RG}^* D_G^{-1}$ operator element for the compound body   |

#### 21.4.1.1.3. class CompoundBodyElementABIVectors

Class for compound body elemental ABI vectors. An instance is stored within the hinge ATBI data structure. It is used within the ATBI filter recursion to only hold the contribution from individual embedded bodies when looping through them all. The results from each such embedded body computation are accumulated into the overall `zR`, `eps` and `aprime` values for the compound body.

| Member  | Type             | Size         | Description  |
|---------|------------------|--------------|--|
| _zR     | SOAVector6       | 6            | The 6-dimensional zR vector for the component subhinge   |
| _eps    | SOAVector        | n_indep_dofs | The eps vector for the compound body subhinge  |
| _aprime | SOASpatialVector | 3,3          | The spatial vector contribution of a specific embedded body to the overall compound body's <code>aprime</code> Coriolis term |

#### 21.4.1.1.4. class CompoundSubhingeABIVectors

Class for compound body ABI vector derived from 'SubhingeABIVectorsBase'.

| Member   | Type                          | Size       | Description  |
|----------|-------------------------------|------------|--|
| elt_vecs | CompoundBodyElementABIVectors | struct     | Holds intermediate <code>_zR</code> , <code>_eps</code> and <code>_aprime</code> values for a single embedded bodies. These values for each of the embedded bodies are accumulated into the overall <code>m_z_G</code> and <code>m_eps_G</code> ATBI values for the compound body. (There is no great reason to make this a member here, and this could be a temporary created during the computations). |
| m_z_G    | SOAVector                     | 6*nbodies  | the ATBI z vector for the compound body. It is essentially a stacked vector consisting of the 6-vector zR contributions from the individual embedded bodies.   |
| m_eps_G  | SOAVector                     | nindepVels | the ATBI $\epsilon_G = T_G - H_{RG} * z_G$ vector for the compound body  |
| m_nu_G   | SOAVector                     | nindepVels | the ATBI $v = D_G^{-1} \epsilon_G$ vector for the compound body  |

Members of a compound subhinge class.

| Member    | Type                        | Size   | Description   |
|-----------|-----------------------------|--------|---|
| _kin_mats | CompoundSubhingeKinMatrices | struct | Contains the <code>m_Hst_RG</code> and <code>m_E_Phi_G</code> kinematics matrices |

| Member                      | Type      | Size                             | Description  |
|-----------------------------|-----------|----------------------------------|--|
| m_X                         | SOAMatrix | m_sg→nGenVel(false) x nindepVels | The $X_G$ independent to full generalized velocities (Jacobian) map ( <i>without constraint dofs</i> ). The order of the columns corresponds to the the order of the subhinges in agg_sg. Thus, when the indep subhinges are in <b>non-canonical</b> order, i.e., (they are not the first nindep subhinges in agg_sg) then the first columns <b>DO NOT</b> correspond to the independent genvels.<br><br>When the indep subhinges are <b>canonical</b> , then the first nindepVels cols are for the independent gen vels, and the latter are for the dependent, and the last ones are for the constraint hinge gen vels. Hence the top nindepVels block is an identity matrix. |
| m_constraintsX              | SOAMatrix | m_sg→nGenVel(true) x nindepVels  | The $X_G$ independent to full generalized velocities (Jacobian) map ( <i>including constraint dofs</i> ). This matrix is essentially the m_X matrix stacked on top of the Jacobian mapping to the constraint hinge gen vels.   |
| _constraintXdot_indepGenVel | SOAVector | m_sg→nGenVel(true)               | (Xdot * indepGenVel) product vector. This product is the kinematic contribution in the generalized accels expression. Note that the constraint hinge dofs terms are included.  |

#### Key methods

| Method                               | Computes  | Size                                       | Description  |
|--------------------------------------|-----------|--|--|
| X0                                   | SOAMatrix | m_sg→nGenVel(false) x nindepVels           | Update and return the $X_G$ matrix ( <i>without the constraint hinge dofs rows</i> )   |
| constraintsX0                        | SOAMatrix | m_sg→nGenVel(true) x nindepVels            | Update and return the $X_G$ matrix ( <i>with the constraint hinge dofs rows included</i> )   |
| dependentGenVelsMap(withConstraints) | SOAMatrix | m_sg→nGenVel(withConstraints) x nindepVels | Return the result of $X()$ or $constraintsX()$ depending on whether withConstraints is false or true respectively.   |
| _computeBlock_Hst_RG(sub_body)       | SOAMatrix | 6 x nindepVels                             | Compute the kinematic $6 \times 6$ column element contribution for the specified embedded body in the m_Hst_RG $H^*$ operator contribution for the compound body. This method is called by the compound body's <code>_updateHiHstMatrices()</code> method.                     |
| _computeBlock_Phi_RG(sub_body)       | SOAMatrix | 6 x 6                                      | Compute the kinematic $6 \times 6$ column element contribution for the specified embedded body in the m_E_Phi_G SKO operator contribution $\phi(parent, cbbd)$ for the compound body. This method is called by the compound body's <code>_updateHiHstMatrices()</code> method. |
| _updatePhiHstMatrices0               | SOMatrix  |  | This method updates the compound body subhinge's m_Hst_RG and m_E_Phi_RG matrices by calling <code>_computeBlock_Hst_RG()</code> and <code>_computeBlock_Phi_RG()</code> for all of the embedded bodies.   |

#### 21.4.1.2. DartsCompoundHinge

| Member | Type | Size | Description |
|--------|------|------|-------------|
|--------|------|------|-------------|

| Member                       | Type  | Size | Description  |
|------------------------------|---|------|--|
| <code>_aprimeVec</code>      | <code>std::map&lt;size_t, SOASpatialVector&gt;</code> | ??   | Stores the <code>aprime</code> contributions from each of the individual embedded bodies (computed and stored within the subhinge level <code>e1t_vecs._aprime</code> member withing the hinge ATBI vectors instance for the compound body). |
| <code>_zRplus_bodyfrm</code> | <code>SOASpatialVector</code>                         | 3, 3 | Need this so we can avoid a cache look up in the <code>accumATBI*</code> calls. These should really be passed as input data to the caches  |

#### Key methods

| Method                                      | Computes               | Size                                 | Description   |
|---|------------------------|--------------------------------------|---|
| <code>bodyReferencedJointMapMatrix()</code> | <code>SOAMatrix</code> | (6 nbodies) x <code>nindepVel</code> | Returns the <code>Hst_RG</code> (i.e. $H_{RG}^*$ ) matrix for the compound body that is its contribution to the $H^*$ operator. Each 6-row contains the mapping from the generalized vels to the body frame spatial velocity for an embedded body |

### 21.4.1.3. DartsCompoundBody

#### Members

| Member                              | Type  | Size | Description   |
|-------------------------------------|---|------|---|
| <code>_dep_subhinge_data</code>     | <code>DartsSubhingeData</code>                |      | subhinge data for the list of dependent subhinges list for use by <code>_iksolver</code>  |
| <code>_dependents_iksolver</code>   | <code>DartsInverseKinematicsSolverBase</code> |      | Inverse kinematics solver for the compound body's internal constraints. This <code>iksolver</code> knows about the independent and dependent bodies in the compound body.   |
| <code>_physical_parent_onode</code> | <code>DartsHingeOnode</code>                  |      | an onode for one of the base bodies in the compound body's subgraph that will be used as a conduit for passing on the compound body's ATBI properties to the parent body. Note that the parent body for this onode is the virtual root body (i.e. the physical parent body). When there are multiple base bodies, the first one in the sorting order is used. |
| <code>_cb_spatialInertia</code>     | <code>SOASpatialInertia</code>                |      | The instantaneous spatial inertia of the compound body derived from its component bodies  |

#### Key methods

| Method                            | Computes               | Size | Description  |
|-----------------------------------|------------------------|------|--|
| <code>physicalParentBody()</code> | <code>DartsBody</code> |      | Return the physical body that this compound body is attached to. This is the same as the <code>virtual root</code> body for the associated <code>aggregation</code> subgraph for the compound body. Note that this physical body in turn may be embedded within a compound body, and so may not be the topological parent following constraint embedding. This body effectively serves as the <code>pnode</code> for a compound body, and many of its quantities are expressed in this body's frame. |
| <code>bodyOffset(subbody)</code>  | <code>size_t</code>    |      | Return the offset for the specified child body in the list of component bodies in the <code>aggregation</code> subgraph. This is used for indexing by <code>DCraft's BStar</code> operator.  |

| Method                                   | Computes  | Size   | Description   |
|--|-----------|--|---|
| <code>_updateHiHstMatrices()</code>      | SOAMatrix |  | This method is updates the compound body subhinge's <code>m_Hst_RG</code> and <code>m_E_Phi_RG</code> matrices.   |
| <code>fromParentEOperator(parent)</code> | SOAMatrix | $6 * (6 \text{ nbodies})$                          | This returns the <code>m_E_Phi_G</code> value for the subhinge. This is the $\phi(\text{parent}, \text{cbdd}) = E_G * \phi_G$ SKO operator element for the physical parent body to this compound body. This matrix is also the lift-up matrix since the transpose of this matrix operator propagates the spatial velocity of the physical parent body to each of the embedded body spatial velocities. The parent body should be the physical parent body if it is not itself embedded, or its compound body if it is embedded.           |
| <code>toChildBOperator(child)</code>     | SOAMatrix | $(6 \text{ nbodies}) * 6$                          | Returns the $B_G$ pick-off operator to the child body. This operator is sparse and just contains a 6x6 identity matrix in the slot for the child body's physical parent body (which is embedded within this compound body). $B_G^*$ picks out the spatial velocity of the child body's physical parent body. The child body will be a physical body if it is not itself embedded, or its compound body if it is embedded.   |
| <code>fromParentPhiMatrix(parent)</code> | SOAMatrix | $(6 \text{ parent.nbodies}) * (6 \text{ nbodies})$ | Return the SKO entry from the parent body to this compound body. This is essentially the product <code>parent.toChildBOperator(this) * fromParentEOperator(parent)</code> . The transpose of this matrix propagates the $V_G(\text{parent})$ full spatial velocities of the parent (compound or not) body to the $V_G(\text{this})$ full spatial velocities of this body.   |
| <code>toChildPhiMatrix(child)</code>     | SOAMatrix | $(6 \text{ nbodies}) * (6 \text{ child.nbodies})$  | Return the SKO entry from this body to the child body. This is essentially the product <code>toChildBOperator(child) * child.fromParentEOperator(this)</code> . The transpose of this matrix propagates the $V_G(\text{this})$ full spatial velocities of this compound body to the $V_G(\text{child})$ full spatial velocities of the child (compound or not) body. The child body will be a physical body if it is not itself embedded, or its compound body if it is embedded.   |
| <code>bodyReferencedATBI_PO</code>       | SOAMatrix | $6 \text{ nbodies} \times 6 \text{ nbodies}$       | Return the ATBI $P_G$ articulated body inertia matrix referenced to the each embedded body's pnode frame. It is a (6 nbodies) square, block-diagonal, symmetric matrix, with the ATBI $P$ matrix for each aggregated body referenced to its pnode along the block diagonal. <b>CAUTION:</b> Note that for a rigid body, this same method returns the $P$ matrix reference to its body frame instead of to the pnode frame. Hence there is a potential inconsistency here between rigid and compound bodies which needs to be looked into. |

| Method                          | Computes  | Size                        | Description   |
|---------------------------------|-----------|-----------------------------|---|
| osi_Upsilon0                    | SOAMatrix | 6 nbodies x 6 nbodies       | Return the $Y_G$ OSC operator's block diagonal, symmetric matrix element for the compound body. While square, this matrix is itself not block diagonal or have sparse structure. Its $(i, j)$ 6x6 block matrix element maps a spatial force in the $j$ embedded body frame to a spatial acceleration in the $i$ embedded body frame.  |
| osi_Upsilon(other)              | SOAMatrix | 6 nbodies x 6 othernbodies0 | Return the $\Omega_G(\text{this}, \text{other})$ OSC operator's cross-entry for the $(\text{this}, \text{other})$ body pair. The $(i, j)$ block element maps a spatial force in the $j$ other body's embedded body frame to a spatial acceleration in this body's $i$ embedded body frame. <b>NOTE:</b> When <code>other</code> is a rigid body, the $j$ frame is its <code>pnode</code> frame and <i>not</i> its body frame. |
| bodyReferencedATBI_osi_Upsilon0 | SOAMatrix | 6 nbodies x 6 nbodies       | Returns the same value as <code>osi_Upsilon()</code> .  |

## 21.5. Raw documents



## 22. NdartsContact

### 22.1. Background

#### 22.1.1. Reference & Source material

- [NdartsContact Doxygen docs](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/NdartsContact/html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/NdartsContact/html)
- [Team talks](https://dlabportal.jpl.nasa.gov/resources/darts-lab-talks/) (https://dlabportal.jpl.nasa.gov/resources/darts-lab-talks/)
- [2019, 2020 course slides](https://dlabportal.jpl.nasa.gov/documentation/coursetutorial-links/) (https://dlabportal.jpl.nasa.gov/documentation/coursetutorial-links/)
- [Sphinx docs](https://dartslab.jpl.nasa.gov/dlabdocs/) (https://dartslab.jpl.nasa.gov/dlabdocs/)
- [Jupyter notebooks](https://dlabnotebooks.jpl.nasa.gov/hub/login) (https://dlabnotebooks.jpl.nasa.gov/hub/login)
- [DARTS Lab QA site](https://dartslab.jpl.nasa.gov/qa/) (https://dartslab.jpl.nasa.gov/qa/)

### 22.2. Design

#### 22.2.1. Unilateral constraints

The DARTS unilateral constraint places a constraint between a pair of frames. These constraints are used in contact dynamics to model the normal and tangential forces due to the contact, collision, or penetration. Each unilateral constraint has an associated contact frame, whose axes point in the following directions:

- The z-axis of the contact frame is parallel to the normal direction of the contact.
- The x-axis is parallel to and points in the direction of the relative tangential velocity of the contact (source tangential velocity minus target tangential velocity).
- The y-axis completes a right-handed coordinate system.

Currently, there are two methods available for calculating the contact force using the penalty method: one has an isotropic friction model and the other has an anisotropic friction model. They are described in detail below. Once the contact force has been calculated, it can be applied to the unilateral constraint using the `applyPenaltyContactForce` method. For example, to apply a penalty method force with isotropic friction one would do,

```
SOASpatialVector F = uc.getPenaltyMethodForces(springConst, dampingCoeff);
uc.applyPenaltyContactForce(F);
```

where `F` is a `SOASpatialVector` containing the contact force in the inertial frame and `uc` is an instance of a `DARTSUnilateralConstraint`. See the `NdartsContactModels.CollisionForces` model (used in DshellCommon's `FullBodyCollision` assembly) for example usage of the isotropic friction penalty forces and the `EELSModels.CollisionForcesAnisotropic` model (used in EELS's `EELSCollisionAssembly`) for example usage of the anisotropic friction penalty forces.

#### 22.2.1.1. Normal contact force

The penalty method normal force is calculated using the following:

$$\begin{aligned}v_{rel} &= v_{source} - v_{target} \\v_n &= (v_{rel} \cdot \hat{n})\hat{n} \\ \hat{v}_n &= \frac{v_n}{||v_n||} \\ F_n &= (Kg + C ||v_n||)\hat{v}_n\end{aligned}$$

where  $v_{source}$  is the source velocity in the inertial frame,  $v_{target}$  is the target velocity in the inertial frame,  $v_{rel}$  is the relative contact velocity,  $\hat{n}$  is a unit-vector in the direction of the contact normal,  $v_n$  is the component of velocity parallel to the contact normal,  $K$  is the spring constant,  $C$  is the damping coefficient,  $g$  is the penetration distance, and  $F_n$  is the penalty method contact force in the normal direction.

#### 22.2.1.2. Isotropic friction

For isotropic friction, the normal force is calculated using the method shown above. Then, the tangential force is calculated using the following:

$$\begin{aligned}v_t &= v_{rel} - v_n \\ \hat{v}_t &= \frac{v_t}{||v_t||} \\ \text{If } ||v_t|| > 1e-5: \\ F_t &= -\mu ||F_n|| \hat{v}_t\end{aligned}$$

where  $v_t$  is the component of velocity in the tangential direction,  $\mu$  is the friction coefficient, and  $F_t$  is the tangential contact force. The total contact force when using the isotropic friction method is  $F_{tot} = F_n + F_t$ . Note that a tangential force is only calculated if the magnitude of the component of velocity in the tangential direction is greater than  $1e-5$ .

#### 22.2.1.3. Anisotropic friction

The anisotropic friction model uses a friction ellipse to define the friction force. This ellipse is determined by  $\mu$ , the friction coefficient, and the eccentricity vector  $e_v$ : the magnitude of this vector defines the level of anisotropy and the direction defines the direction of anisotropy. For anisotropic friction, the normal force is calculated using the method shown above. Then, the friction force is calculated using the following:

$$\begin{aligned}
v_t &= v_{rel} - v_n \\
\hat{v}_t &= \frac{v_t}{||v_t||} \\
\text{If } ||v_t|| > 1e-5: \\
e &= e_v - (e_v \cdot \hat{n})\hat{n} \\
\hat{e} &= \frac{e}{||e||} \\
v_e &= \hat{v}_t + (||e_v|| - 1)(\hat{v}_t \cdot \hat{e})\hat{e} \\
F_t &= -\mu ||F_n|| v_e
\end{aligned}$$

where  $e$  is the eccentricity vector and  $F_t$  is the tangential contact force. The total contact force when using the anisotropic friction method is  $F_{tot} = F_n + F_t$ . Like in the isotropic friction case, a tangential force is only calculated if the magnitude of the component of velocity in the tangential direction is greater than  $1e-5$ .

The derivation for the anisotropic friction force follows. If we were working in a frame where the x- and y-axes were aligned with the principle friction axes respectively (z-axis is normal), then the force of friction would be

$$F_t = -||F_n|| \begin{bmatrix} \mu ||e_v|| & 0 & 0 \\ 0 & \mu & 0 \\ 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} t_1 \\ t_2 \\ 0 \end{bmatrix},$$

where  $t_1$  is the magnitude of the component of  $\hat{v}_t$  parallel to the  $e$ -vector and  $t_2$  is the magnitude of the component of  $\hat{v}_t$  perpendicular to the  $e$ -vector. Mathematically,

$$\begin{aligned}
t_1 &= \hat{v}_t \cdot \hat{e} \\
t_2 &= ||\hat{v}_t - \hat{e}t_1||.
\end{aligned}$$

Recall that the x- and y-axes were aligned with  $\hat{e}$  and a direction perpendicular to  $\hat{e}$ . Thus, the friction force is

$$\begin{aligned}
F_t &= -F_n \mu (||e_v|| \hat{e}(\hat{v}_t \cdot \hat{e}) + \hat{v}_t - \hat{e}(\hat{v}_t \cdot \hat{e})) \\
F_t &= -F_n \mu (\hat{v}_t + (||e_v|| - 1)(\hat{v}_t \cdot \hat{e})\hat{e})
\end{aligned}$$

Note, as a sanity check we can set  $e = 1$  (the isotropic case); then we recover,

$$F_t = -F_n \mu \hat{t}$$

which is the normal isotropic friction equation. Moreover, if we set  $e = 0$ , then we expect to see no friction in the  $\hat{e}$  direction,

$$F_t = -F_n \mu (\hat{t} - (\hat{t} \cdot \hat{e})\hat{e})$$

which is exactly what we get; the component of  $F_t$  in the  $\hat{e}$  direction is completely removed.

## 22.3. Background

## 22.4. Software

### 22.4.1. CollisionNdarts

The `visualizeContactForces` method is used to visualize the tangential and normal components of the contact forces. It is called with the following parameters:

- `scene` (*DartsFacadeScene*) - scene where the forces should be displayed.
- `enable` (*bool, optional*) - used to enable/disable visualization of the contact forces. (Default = True)
- `normal` (*bool, optional*) - used to enable/disable visualization of the normal contact forces. (Default = False)
- `tangent` (*bool, optional*) - used to enable/disable visualization of the tangent contact forces. (Default = True)
- `scale` (*float, optional*) - used to scale the external force vectors. (Default = 0.1)

## 22.5. Raw documents

## 23. NdartsFlex

### 23.1. Background

#### 23.1.1. Reference & Source material

- [NdartsFlex Doxygen docs](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/NdartsFlex/html/) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/NdartsFlex/html/)
- [Team talks](https://dlabportal.jpl.nasa.gov/resources/darts-lab-talks/) (https://dlabportal.jpl.nasa.gov/resources/darts-lab-talks/)
- [2019, 2020 course slides](https://dlabportal.jpl.nasa.gov/documentation/coursetutorial-links/) (https://dlabportal.jpl.nasa.gov/documentation/coursetutorial-links/)
- [Sphinx docs](https://dartslab.jpl.nasa.gov/dlabdocs/) (https://dartslab.jpl.nasa.gov/dlabdocs/)
- [Jupyter notebooks](https://dlabnotebooks.jpl.nasa.gov/hub/login) (https://dlabnotebooks.jpl.nasa.gov/hub/login)
- [DARTS Lab QA site](https://dartslab.jpl.nasa.gov/qa/) (https://dartslab.jpl.nasa.gov/qa/)

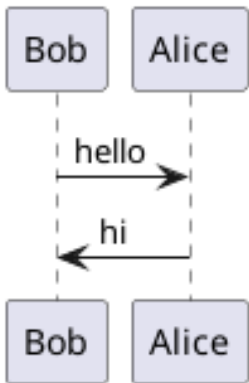
### 23.2. Design

#### 23.2.1. Frame conventions



TBD: Flesh out and clean up this section

- need to keep track of what is in what frame
- All nodal matrices are in body frame. Hence any computations involving them should be in the body frame - instead of in the onode or pnode frame.
- On the other hand all hinge ATBI stuff for now appears to be in the child body's pnode frame (which is unaffected by flex)
- All the accelerations are in pframe (i.e the local frame)



### 23.3. Usage

#### 23.3.1. Flexible body constraints

Flexible body constraints can be added to a DARTS flexible body via flexible constraint nodes. Flexible constraint nodes have the same positioning methods as flexible sensor nodes or flexible actuator nodes:

- `bodyToNodeTransform` - Defines the homogenous transform between the body and the constraint node. This can be set via the `setBodyToNodeTransform` method. Pieces of this transform can be set using other methods, e.g., `setBodyToNode` sets the translational portion of this transform.
- `nodalMatrix` - Defines the nodal matrix for the constraint node. This can be set via the `nodalMatrix` method.

For example, suppose one used the `FModal` module to bring in a flexible body via an H5 file from the NASTRAN-to-DARTS pipeline, and they want to place the constraint node so that it corresponds with node 1 from the corresponding H5 file. Further, let this H5 file be denoted by the variable `H5`. Then, this node could be created via the following:

```
cn1 = body.createConstraintNode("constraint").upCast()
cn1.setBodyToNode(SOA_Py.SOAVector3(H5.getGridPosition(1).tolist()))
cn1.nodalMatrix(SOA_Py.SOAMatrix(H5.getFlexModeShape(1).tolist()))
```

PYTHON

where the body the constraint is being added to is denoted by the variable `body`. Furthermore, the above assumes that the quaternion between the body frame and node 1 in the undeformed state is identity, since the `setBodyToNode` method was used, which only sets the translation of the flexible constraint node.

Once one has defined two constraint nodes, then a constraint can be setup between these nodes. Note, that one can mix flexible- and rigid-body constraint nodes if desired, i.e., both constraint nodes in the pair do not need to be flexible constraint nodes. For example, a DARTS closure constraint that mimics a `LOCKED` hinge can be created between two constraint nodes `cn1` and `cn2` via:

```
cc = Ndarts_Py.DartsClosureConstraint(sim.mbody(), "my_constraint", cn1, cn2, Ndarts_Py.HINGE_LOCKED, [])
```

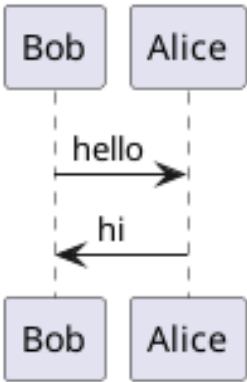
PYTHON

Note that like rigid-body-only constraints, one must add this constraint to a constrained subgraph and setup a dynamic solver and integrator that can handle constraints in order to use the constraint in the dynamics calculations. Examples of constraints between flexible and rigid bodies can be found at `FModal/test/test_flex_constraints/script1.py` and `FModal/test/test_flex_constraints/script2.py`. An example of a constraint between two flexible bodies can be found at `FModal/test/test_flex_constraints/script3.py`.

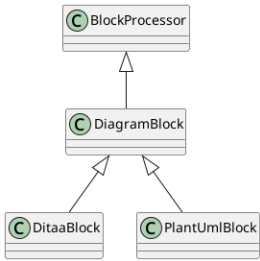
## 23.4. Software

### 23.4.1. Flex ATBI recursion structure

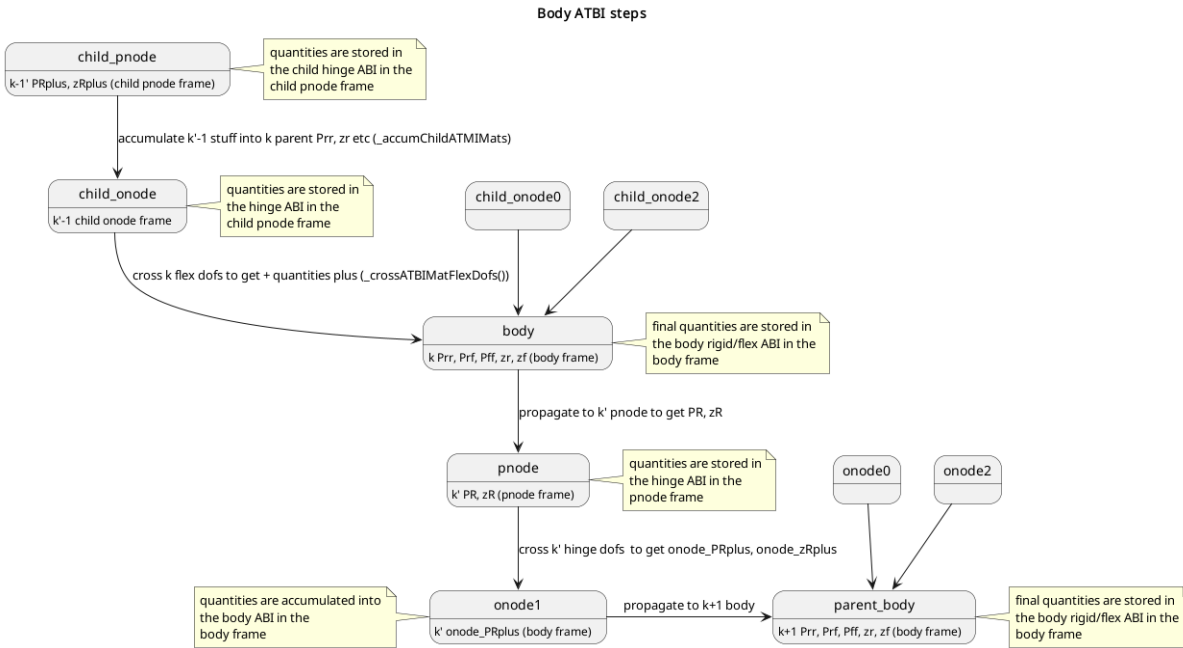
The following is a plantuml example in gitlab from <https://docs.gitlab.com/ee/administration/integration/plantuml.html#configure-your-plantuml-server>



The following is a plantuml example from asciidocs site at <https://docs.asciidoc.org/diagram-extension/latest/>



The following is the flex dynamics figure from notes.org



## 23.5. Raw documents

## 24. FModal

### 24.1. Background

Flexible multibody dynamics models are often used to develop and analyze guidance and control (G&C) systems for flexible body vehicle platforms. Multibody dynamics models can handle the nonlinear dynamics from large articulation, configuration changes from the attachment and detachment of bodies, as well as mass property changes from fuel depletion. Such multibody models can be used to extract linearized models for control system design as well as for closed-loop, time-domain dynamics simulations to verify system performance. Multibody dynamics models often rely on reduced-order models for G&C development and analysis. Such reduced-order models are typically based on modal representations derived from finite element method (FEM) structural analysis models for the component flexible bodies. The generation and use of such FEM-based modal models is a complex process requiring: (1) modal analysis for the flexible body, (2) computation and transfer of a large set of quantities from modal analysis such as frequencies, mode shapes, modal integrals, etc. (3) use of this data to create corresponding flexible bodies within a multibody dynamics tool. The process requires a user to have expertise (quite uncommon) with both FEM and multibody software and the careful and error-free generation and transfer of a large amount of data across the tools. The process can be quite challenging, slow, and error-prone for even basic flexible body models and remains a significant source of friction for model transfer from the structural analysis to the G&C domains. Due to the complexity of the modal data extraction and transfer process, it is an unfortunate but common practice in the community to simplify the process by ignoring most of the modal integral and geometric stiffening terms resulting in reduced fidelity of the multibody dynamics models. One of the key goals of the FModal tool is to avoid such unnecessary devaluation of model fidelity by bridging the gap between the structural dynamics and G&C domains.



TBD: Include some of the flowchart figures from the NESC report

The recently developed FModal tool addresses the aforementioned problem by streamlining the process of generating modal data (including modal integrals and geometric stiffening terms) from component NASTRAN structural dynamics models for use in flexible multibody dynamics tools. FModal's output is a portable and open-source HDF5 file that is a universal, hierarchical, well-organized, and labeled data-set that can be interfaced with multibody tools to automate and simplify the model data transfer process. The FModal tool can be used to extract the critical data needed for flexible multibody dynamics simulations from a NASTRAN structure model without the need for an expert in structural dynamics. Furthermore, since FModal includes the quantities needed to add modal integrals and geometric stiffening due to inertial loads to the flexible body equations of motion, the user is able to tailor the fidelity of the simulation to meet their needs. Consequently, FModal can reduce the time needed to create flexible body models and reduce errors in the model transfer process, which leads to faster design iterations—features that can result in shorter overall project timelines and reduced costs.

#### 24.1.1. Reference & Source material

- [FModal Doxygen docs](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/FModal/html/) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/FModal/html/)
- [Team talks](https://dlabportal.jpl.nasa.gov/resources/darts-lab-talks/) (https://dlabportal.jpl.nasa.gov/resources/darts-lab-talks/)
- [2019, 2020 course slides](https://dlabportal.jpl.nasa.gov/documentation/coursetutorial-links/) (https://dlabportal.jpl.nasa.gov/documentation/coursetutorial-links/)
- [Sphinx docs](https://dartslab.jpl.nasa.gov/dlabdocs/) (https://dartslab.jpl.nasa.gov/dlabdocs/)
- [Jupyter notebooks](https://dlabnotebooks.jpl.nasa.gov/hub/login) (https://dlabnotebooks.jpl.nasa.gov/hub/login)
- [DARTS Lab QA site](https://dartslab.jpl.nasa.gov/qa/) (https://dartslab.jpl.nasa.gov/qa/)

## 24.2. Design

### 24.2.1. NASTRAN-to-DARTS Pipeline



TBD: Flesh out and clean up this section

- Discuss pipeline requirements
- Include figures and quotes from the NESCFlex document

#### 24.2.1.1. OP2 Reader

The `OP2Reader` is designed to read NASTRAN OP2 files, process the data, and write the refined data to an HDF5 file. OP2 files are binary files designed to be read by FORTRAN. Information on how these files are structured can be found [here](https://docs.plm.automation.siemens.com/tdoc/nxnastran/11/help/#uid:index_dmap:xid666580:id496821) (https://docs.plm.automation.siemens.com/tdoc/nxnastran/11/help/#uid:index\_dmap:xid666580:id496821).

#### 24.2.2. Modal Analysis

The `ModalAnalysis` tool from `FModal` is used to calculate mode shapes for the bodies in a given subgraph. The modal analysis tool calculates these mode shapes by forming a mass and stiffness matrix for the subgraph, and solving the associated eigenproblem to compute the modal frequencies and mode shapes; the `eig` function from `scipy` is used to solve the aforementioned eigenproblem. In the general case, the mode shapes may include rigid-body modes, e.g., when calculating the mode shapes of a free-free beam. Hence, the `ModalAnalysis` tool comes with a `tol` variable that can be used to remove mode shapes with a frequency less than `tol`, since the rigid-body mode shapes of the system have a smaller modal frequency than the flexible mode shapes.

#### 24.2.2.1. Incorporating constraints

When a subgraph contains constraints, the problem becomes more complicated, since not all DoFs of the subgraph are independent DoFs, i.e., some of these DoFs must be used to satisfy the constraint. In these cases, a set of dependent DoFs must be chosen, and are used to form a projection matrix  $X$ . This projection matrix is used to obtain the mass and stiffness matrices for the independent DoFs via,

$$M_I = X^T M X$$
$$K_I = X^T K X$$

where  $M_I$  and  $K_I$  are the mass and stiffness matrices associated with the independent DoFs only, and  $M$  and  $K$  are the mass and stiffness matrices associated with all the DoFs of the subgraph. Once the modal frequencies and mode shapes,  $\omega$  and  $v_I$  respectively, are obtained from solving the eigenproblem associated with  $M_I$  and  $K_I$ , the results are projected back to the original set of coordinates using

$$v = X v_I$$

where  $v$  are the mode shapes for the full set of DoFs.

### 24.2.2.2. Choosing dependent DoFs

In order to form the projection matrix, a set of dependent DoFs must be chosen. If the user has intuition for what these dependent DoFs should be, then they can specify them directly. However, in general, the user may not know which DoFs to use as dependent DoFs.

Ultimately, the dependent DoFs should produce a projection matrix that has a low condition number. The user is welcome to implement their own algorithm to try and find dependent DoFs that do this. However, if constraints are supplied to `ModalAnalysis` without any dependent DoFs, then the `ModalAnalysis` has an algorithm that will be used to find suitable dependent DoFs for the user. That algorithm proceeds as follows:

1. Generate a candidate set of dependent DoFs using the Enhanced Pattern Shifting (EPS) algorithm; see [here](https://www.sciencedirect.com/science/article/pii/S0094576516308232) (https://www.sciencedirect.com/science/article/pii/S0094576516308232) for details.
2. Calculate the projection matrix using this set of DoFs.
3. If the project matrix has a sufficiently low condition number (controller by the `condTo1` keyword argument to `ModalAnalysis`), then function returns this candidate set. Otherwise, the algorithm repeats starting at step 1.

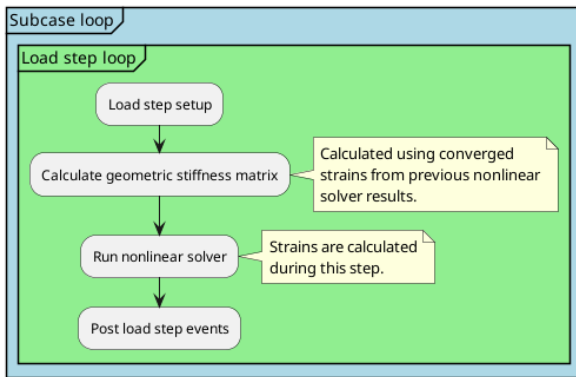
### 24.2.3. Geometric stiffening due to inertial loads



TBD: Include discussion of Banerjee's method and why we use it

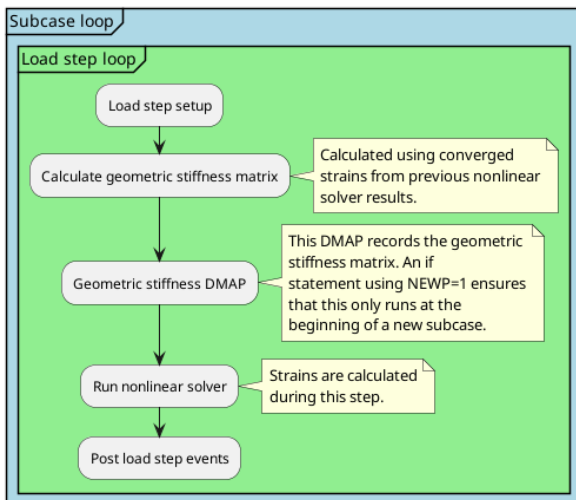
A NASTRAN DMAP is used to extract the geometric stiffness matrix of the model at the 21 loading conditions required by Banerjee's method. Some important points regarding the NASTRAN geometric stiffness calculation and SOL 106 non-linear solver process:

- NASTRAN applies the loads for a given subcase in increments. It calculates the difference in loads between the previous subcase (or 0 loading if there was no subcase prior) and the loading for the current subcase. Then, it applies this difference incrementally. The incremental application of loads is called a load step.
- NASTRAN utilizes the strain of the model to calculate the geometric stiffness matrix (often called the differential stiffness matrix in NASTRAN documentation). The calculation frequency of this matrix is controlled by settings in `NLPARAM`. The `Nastran` tool modifies these settings so that the geometric stiffness matrix is computed at each load step.
- The SOL 106 process proceeds as follows:



- The geomtric stiffness matrix computation is quite complex, so the DMAP should not re-calculate the matrix if possible. Rather, it should be placed somewhere in this iteration loop such it records the geometric stiffness matrix after the strains for the subcase have been completely converged.

This information leads to the following placement of the geometric stiffness DMAP:



This placement of the geometric stiffness DMAP means that the geometric stiffness matrix for subcase k is actually recorded at the beginning of subcase k+1. This means that to record the geometric stiffness matrix for subcase 21, there needs to be a subcase 22. Hence, a 22nd, dummy subcase with 0 loading is added to the DAT file by `Nastran`. In addition, this also means that the first time the DMAP records a geometric stiffness matrix, it will have useless values, since no subcases have run yet. Therefore,

## 24.3. Usage

### 24.3.1. NASTRAN-to-DARTS Pipeline



TBD: Flesh out and clean up this section

- Discuss pipeline usage at a top level
- Discuss `Nastran` and `ReadHDF5Flex` classes in detail

## 24.4. Software

### 24.5. Raw documents

Unresolved directive in `darts_lab_documentation_developer.asciidoc` - `include:::/dartsgitlab_internal_docs_https/jpl-internal/all-access/development/numerics/timekeeper/TimeKeeper-documentation.asciidoc[]`

## 25. Dtest

### 25.1. Background

#### 25.1.1. Reference & Source material

- [Dtest Doxygen docs](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/Dtest/html/) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/Dtest/html/)
- [Team talks](https://dlabportal.jpl.nasa.gov/resources/darts-lab-talks/) (https://dlabportal.jpl.nasa.gov/resources/darts-lab-talks/)
- [2019, 2020 course slides](https://dlabportal.jpl.nasa.gov/documentation/coursetutorial-links/) (https://dlabportal.jpl.nasa.gov/documentation/coursetutorial-links/)
- [Sphinx docs](https://dartslab.jpl.nasa.gov/dlabdocs/) (https://dartslab.jpl.nasa.gov/dlabdocs/)
- [Jupyter notebooks](https://dlabnotebooks.jpl.nasa.gov/hub/login) (https://dlabnotebooks.jpl.nasa.gov/hub/login)
- [DARTS Lab QA site](https://dartslab.jpl.nasa.gov/qa/) (https://dartslab.jpl.nasa.gov/qa/)

### 25.2. Design

### 25.3. Usage



TBD: Flesh out and clean up this section

### 25.4. Software

### 25.5. Raw documents

### 25.6. Sphinx documentation

#### 25.6.1. Introduction

The Dtest module contains an automated software validation framework, featuring extensive support for regression testing of C/C++ and Python code. This module contains several useful program verification utilities including:

- `dtest` - Runs sets of regression tests
- `compareCheckpointFiles` - Compares checkpoint output files
- `compareModelFiles` - Compares model definition files
- `compareDictsInFiles` - Compares Python dictionaries embedded in text files
- `Ddoctest` - a drop-in replacement for the python 'doctest' library that does numerical comparisons with relative and absolute tolerances.

#### 25.6.2. JPL Darts/Dshell Testing Framework

##### 25.6.2.1. Dartslab Regression Testing

- Hourly, Daily, and Release-driven Builds
- Currently over 500 regression tests
- **Sandboxes used for testing**
  - Source
  - Link
- **Test mechanism**
  - Python doctest
  - Arbitrary scripts
- **Testing Reporting**
  - Email
  - Website

##### 25.6.2.2. The Darts/Dshell dtest Testing Harness

- Modules can run `'gmake -f Makefile.yam regtest'` to run all regression tests in the module
- Runs all tests in a specified test directory and all sub-directories (recursively): **\*Runs in directories that match the `test_` file name pattern** (the older `test-*` format is still supported, but developers are encouraged to use the new "underscore" naming convention).
- **'dtest' uses a configuration file: DTESTDEFS**
  - DTESTDEFS files use the *configObj* format (a super set of Windows .ini files)
  - +
    - Define test command(s) for sub-directories**
      - Can be an arbitrary command (run executable, run python script, etc)
      - Define where output results go (for comparison tests)



- Define how to do comparisons
- Define any cleanup procedures
- Each test directory can override any parent configuration information with its own DTESTDEFS file (e.g., run multiple and/or specialized tests in the directory)

### 25.6.2.2.1. Typical Testing Directory Structure

```

|-- DTESTDEFS
|-- test_basic
|   |-- graphics.py
|   |-- test_algmbody
|   |   |-- model.py
|   |   |-- output.orig
|   |   |-- script.py
|   |-- test_rigidbody
|   |   |-- test_tumbling
|   |   |   |-- DTESTDEFS
|   |   |   |-- model.py
|   |   |   |-- output.orig
|   |   |   |-- outputEulerSemiImplicit.orig
|   |   |   |-- script.py
|   |   |   |-- scriptEulerSemiImplicit.py
|   |   |   . . . . . snip . . . . .
|   |   |-- utils.py
|   |-- test_subgraphs
|   |   |-- model.py
|   |   |-- output.orig
|   |   |-- script.py
|-- test_shapes
|   |-- test_cylinder2D
|   |   |-- output.orig
|   |   |-- script.m
|   |   |-- script.py
|   |-- test_torus2D
|   |   |-- output.orig
|   |   |-- script.m
|   |   |-- script.py

```

#### NOTES

- Nested test directories
- Base DTESTDEFS file defines defaults for all tests
- DTESTDEFS files in sub-directories
- DTESTDEFS files in individual test directories can override any higher-level defaults, as seen in the above layout of the `test_tumbling` directory structure.
- Each test sub-directory has complete control over how its test works using its own DTESTDEFS file.

### 25.6.2.2.2. Sample DTESTDEFS file

```

# comma separated list of tests to skip
SKIPTESTS = test_func1, test_func2

# file suffix for the "known good" comparison file
TRUTHSUFFIX = orig

# "debris" files to be deleted after the test
DELETE=.*\pzc

# the default comparison program and options
CMP = cmp.prg 0, cmp.prg 1

[COMPARE]

model.out$ = compareModelFiles, /usr/bin/diff

[RUN]

cmd1 = python script.py >& output
cmd2 = python script2.py >& output2

```

#### NOTES

- Files with the designated "truth" suffix are used to determine correctness of the run.
- Custom comparison programs can be specified (e.g., the system's diff utility)
- Comparisons can be defined in the [COMPARE] section as shown above, or alternatively, a custom comparison operation can be performed as part of the test command in the [RUN] section. Keep in mind that individual tests are free to override the default options for the comparison utility.

### 25.6.2.2.3. Running dtest in a test directory

- dtest reads local DTESTDEFS file (if any)
- dtest runs all the test commands
- Check for test failures (error return status)
- Compare output files with the known "truth" output (e.g., manually verified, standard data set, etc)
  - If any of the comparisons fail, then the entire test fails

- Delete any debris files (as defined in DTESTDEFS)
- Save test results into the regtest.data text file. This output file can be analyzed at a later time with automated tools (see `usage`).

#### 25.6.2.2.4. Testing Flexibility with dtest

- Tests can be completely user-defined
- Output files that will be compared with "truth" versions can be redirected from standard output or created in any manner by the test (e.g., image files)
- We have comparison scripts that take lists of regexps to ignore superfluous differences (e.g., file paths)
- We sometimes do specialized file comparisons as tests themselves to do "fuzzy" tests
- Tests that are obsolete can be skipped by the DTESTDEFS file

### 25.6.3. Using the Dtest Utilities

#### 25.6.3.1. Installing Dtest

In order to install the Dtest utilities, simply include the Dtest module in the user's sandbox as a work module (directory) or link module and relink. All necessary executables and libraries will then be usable.

#### 25.6.3.2. Using the dtest test harness

The dtest test program can be used in one of three ways:

1. Run dtest for an entire sandbox. In the top level of the sandbox, do this:

```
$ gmake regtest
```

SH

Note this should work without difficulties for sandboxes where all modules are checked out as work modules. If any modules are link modules, there may be issues if the user cannot write test files in the module release area.

Note that this is the way that hourly/overnight regression tests are run: A dedicated all-work sandbox is set up and regression tests and this command is executed at the top level to run regtests for all modules.

2. Run dtest for a module. Go to the top level of the module and execute this command:

```
$ gmake -f Makefile.yam regtest
```

SH

3. Run dtest for a set of regression tests. Go to the level in the test hierarchy where you want to run tests and execute this command:

```
$ <Drun> dtest
```

SH

where dtest is the Drun executable for this sandbox. Note that this command can be run at any level in the test directory. It only runs the regression tests it finds in the directory in which it is run along with any children test directories.

#### Dtest options

#### 25.6.3.3. Dtest options

Dtest accepts several command line options when it is executed directly. To specify options, you may give them on the dtest command line:

```
$ <Drun> dtest <options>
```

SH

You may see the available options by executing dtest with the '--help' option.

#### Dtest tags

#### 25.6.3.4. Selective execution of Dtest tests with TAGS (Dtest tags)

It is possible to selectively exclude regtests based on the 'TAGS' property defined in the 'DTESTDEFS' file withing a test sub-directory (but not child directories). Dtest TAGS offer significant flexibility in controlling the execution of regression tests. For instance, if the following tag is defined in a regtest's DTESTDEFS file:

```
TAGS = logging
```

and dtest is executed with the '--exclude-tags' option:

```
$ <Drun> dtest --exclude-tags logging
```

SH

then any regtest that defines TAGS to include 'logging' in its DTESTDEFS file will be excluded (skipped).

You may also run only regtest that contain a specific tag:

```
$ <Drun> dtest --run-only-tags validation
```

SH

then only regtests that defines TAGS to include 'validation' in its DTESTDEFS file will be executed. Note that all regtests that are marked for skipping or quarantining, will not be run.

The `dtest` argument `--exclude-tags` can be given multiple times on the command line and all the exclude tags will be combined. The `dtest` argument `--run-only-tags` is handled similarly.

Note that `TAGS` only affects the subdirectory that it is defined in (in a `DTESTDEFS` file). If you wish to cause all child directories to inherit a tag, use the `CHILD_TAGS` property:

```
“ CHILD_TAGS = logging
```

With this defined, all tests in child directories will get the 'logging' tag—as if 'TAGS = logging' was defined in the `DTESTDEFS` file in all subdirectories below this directory.



**Note**

To see a TIM summary presentation on Dtest Tags, please see:

- [Dtest Tags Presentation \(April 2016 TIM\) <documents/Dtest-Tags-2016-04-08.pdf>](#)

#### 25.6.3.4.1. Special Dtest tags: skip and quarantined

There are two special tags: 'skip' and 'quarantined'. These do the same things as listing the regtests in parent `DTESTDEFS` file 'SKIP' or 'QUARANTINED' lists (respectively).

One advantage in using tagging to skip or quarantine a regtest is that skip/exclusion flag goes into the same directory as the regtest itself (not its parent).

#### 25.6.3.4.2. Commonly used Dtest tags

There are several tags that have been defined to provide consistent tagging of tests. These tags are:

- 'code' - Identifies tests that perform code checks.
- 'graphics' - Identifies tests that use OGRE or other graphics packages that might not be available on all platforms.
- 'lcm' - Identifies tests that use Lightweight Communications and Marshalling (LCM) software which might not be available on all platforms.
- 'gtest' - Identifies tests that use GoogleTest (gtest) software which might not be available on all platforms.

#### 25.6.3.5. Useful Tools for Automated Testing

JPL provides the following scripts and utilities for automated code testing tasks:

- `Ddoctest` : extends `doctest` by calling `numarray.allclose` so numeric differences that fall under a specified tolerance will pass. To make existing script.py scripts work, add `from Dutils import Ddoctest` after `import doctest`. By default, `Ddoctest` passes the following parameters to `numarray.allclose` :

- ```
“ ◦ relative error (stored in doctest.rtol) = 1.0000000000000001e-05
◦ absolute error (stored in doctest.atol) = 1e-08
```

- `compareDictsInFiles` : performs a comparison between specially delimited Python dictionaries embedded in text files.

```
“ Usage: compareDictsInFiles [options] <dict-file1> <dict-file2>
```

SH

#### Options

**-h, --help show this help message and exit**

**-v, --verbose** Show the diffs. **--start\_re=START\_RE** Regular expression immediately before the dictionary (on its own line). [Default: '<START>']. **--end\_re=END\_RE** Regular expression immediately the dictionary (on its own line). [Default: '<END>']. **--abs\_eps=ABS\_EPS** The absolute epsilon to be used in comparisons (defaults to 1.0e-12) **--rel\_eps=REL\_EPS** The relative epsilon to be used in comparisons (when numbers are not small, defaults to 1.0e-12)

- `compareCheckpointFiles` : performs a comparison between two different checkpoint files.

```
“ compareCheckpointFiles [options] <state-file1> <state-file2>
Options:
-h, --help show this help message and exit
-v, --verbose Show the diffs.
```

- `compareModelFiles` : performs a comparison between two different model files.

```
“ compareModelFiles [options] <model-file1> <model-file2>
Options:
-h, --help show this help message and exit
-v, --verbose Show the diffs.
```

- `genRegtestMail` : takes a `regtest.data` file (the output of a `dtest` run) and constructs and then sends emails based on various options.

```
“ genRegtestMail [options] -dataFile <regtest data file>
Options:
```

SH

```

-h, --help          show this help message and exit
--dataFile=DATAFILE  Regtest data file.
--title=TITLE        Title to print above the table
--html              Generate html output to this file
--url=URL            URL for detailed html results (module name is
                    appended
                    for each module)
--moduleOwner=MODULEOWNER
                    string to be evaluated (or file to be executed) to
                    construct dictionary:
                    moduleOwner = {'<module-name>' : '<owner>', ...}
--file=FILE          file for detailed html results
--output=OUTPUT      Output file for generated HTML file (defaults to
                    stdout)
--email=EMAIL        Email address or comma-separated list of emails to
                    send the summary to. If omitted, print results or
                    write to output file.

```

- `genRegtestHtml`: takes a `regtest.data` file (the output of a dtest run) and generates an HTML formatted status page according to several options.

```

“ genRegtestHtml [options] -dataFile <regtest data file> SH

Options:
-h, --help          show this help message and exit
--dataFile=DATAFILE  Regtest data file.
--cssURL=CSSURL      Full URL for CSS file
--title=TITLE        Title for regression test results HTML page.
--output=OUTPUT      Output file for generated HTML file (defaults to
                    stdout)
--showVal            Show the VAL outputs for regtests (defaults to False)

```

- `cmp.prg`: a UNIX shell script for comparing differences between text files.: White space (and blank lines) are ignored, as well as expected differences, which are specified as regular expressions in a separate file. The first command line argument is a Boolean (i.e., "0" or "1") debug flag; when enabled, unexpected differences between files are printed to the screen. The second argument is the text file containing regular expressions, written using the GNU grep syntax; the regular expressions define what text strings should match the expected variance between two files, whose names are given as the third and fourth arguments respectively. The script returns 1 if the files contain unexpected differences, and 0 otherwise.

## 25.6.4. DUnit Testing Framework

### 25.6.4.1. Introduction

DUnit extends and uses the native python unittest framework. DUnit is a specialized unit tester facilitating the collection of evaluation results and its requirement identifier at the evaluation statement rather than the method.

#### DUnit add the following capabilities

- Comparisons between lists, with or without a tolerance
- Comparisons between SOA Vectors, Matrices, Eulers, and Quaternions
- Comparisons between rotations, recognizing equivalent but not-identical values
- Comparisons using variance, to measure relative difference as opposed to absolute difference
- Comparisons that convey metadata for the purposes of process / metrics

### 25.6.4.2. Extends Python Unittest

DUnit is an extension of Python Unittest. Therefore, it has the same file structure and basic behavior as Unittest, with a few exceptions. Most importantly, the previous assertion methods have been deprecated, and replaced with our own assertion methods. You may, however, use the setup and teardown methods as you would in Unittest.

### 25.6.4.3. DUnit Output

DUnit provides custom formatted output results for each individual assertion you make to both ASCII and XML formats. This output must be turned on if desired though, as it is suppressed by default. To turn on the file output, set the environment variable `DUNIT_FILE_OUT` to 'True'.

DUnit also provides output to standard out, if desired. The following arguments control that output::: \* `evalStdOut True` sends results to standard out. **False sends no results to standard out.** \* `evalStdOutOnlyFail True` sends only FAIL results to standard out. **False** sends all results to standard out.

### 25.6.4.4. Example Usage Syntax

```

from Dtest.dunit import DUnitTest
from Dtest.dunit import DUnitTestRunner
from Dtest.dunit.DUnitTest import RequirementIdentifier as Req

class CovarianceDispersionTest(DUnitTest.DUnit):
    def test_evaluations(self):
        expectedValue = 1
        actualValue = 1
        self.evalEqual(Req("REQ-001"), expectedValue, actualValue)
        self.evalEqualDelta(Req("REQ-001"), expectedValue, actualValue, 1e-10)
        self.evalExpTrue(Req("REQ-001"), expectedValue > actualValue)

if __name__ == '__main__':
    DUnitTestRunner.DUnitRunner(evalStdOut=True, evalStdOutOnlyFail=True)

```

PYTHON

The above code shows a basic DUnit test. After importing the necessary libraries, the tester makes his test a subclass of DUnit. The test method shows 3 of the common evaluation types:

- *evalEqual* will compare the value 1 to 1, and evaluate as PASS.
- *evalEqualDelta* will compare the same numbers within a tolerance of 1e-10, also evaluating to PASS.
- *evalExpTrue* will test the expression '1 >1', which will evaluate to FAIL.

The final line shows the arguments used for output, turning on standard output, but only for tests that FAIL.

#### 25.6.4.5. DUnit Methods

##### 25.6.4.5.1. DUnitFactoryEvaluation

###### 25.6.4.5.1.1. Class Documentation

###### 25.6.4.5.1.1.1. Introduction

A factory-based object and collection of EvaluationObj objects used to perform comparisons.

###### 25.6.4.5.1.1.2. Example Usage Syntax

```
evaluationObj = FactoryEvaluation.newEvaluationObj(self._expectedValue, self._actualValue, self._delta, self._deltaVariance,
self._quaternionRotation)
```

PYTHON

###### 25.6.4.5.1.1.3. Related Regression Tests

###### 25.6.4.5.1.1.4. DUnitFactoryEvaluation Class API Documentation



###### Note

Dtest.dunit.DUnitFactoryEvaluation

##### 25.6.4.5.2. DUnitFactoryResult

###### 25.6.4.5.2.1. Class Documentation

###### 25.6.4.5.2.1.1. Introduction

A factory-based object and collection of Result objects used to perform comparisons and format results.

###### 25.6.4.5.2.1.2. Example Usage Syntax

```
resultObj = FactoryResult.newResultObj(self._evalMessage(requirementIdentifier), expectedValue, actualValue, delta, deltaVariance,
quaternionRotation, msg, collect, bypass, resultObjSpecial)
```

PYTHON

TODO: Additional comments concerning the code example.

###### 25.6.4.5.2.1.3. Related Regression Tests

TODO: Insert link to vDUnitUnittest.rst file.

###### 25.6.4.5.2.1.4. DUnitFactoryResult Class API Documentation



###### Note

Dtest.dunit.DUnitFactoryResult

##### 25.6.4.5.3. DUnitFactorySoa

###### 25.6.4.5.3.1. Class Documentation

###### 25.6.4.5.3.1.1. Introduction

FactorySoa is a factory-based object used to create Math.SOA\_Py objects.

###### 25.6.4.5.3.1.2. Example Usage Syntax

```
expectedValue = FactorySoa.newQuaternion(1.1, 2.2, 3.3, 4.4)
```

PYTHON

TODO: Additional comments concerning the code example.

###### 25.6.4.5.3.1.3. Related Regression Tests

TODO: Insert link to vDUnitUnittest.rst file.

###### 25.6.4.5.3.1.4. DUnitFactorySoa Class API Documentation



###### Note

Dtest.dunit.DUnitFactorySoa

##### 25.6.4.5.4. DUnitRunner

###### 25.6.4.5.4.1. Class Documentation

###### 25.6.4.5.4.1.1. Introduction

DUnitRunner extends and uses the native python unittest framework. DUnitRunner is a specialized unit test runner that allows passing command line arguments to the test case itself rather than only passing arguments to the unit test runner. Output generated by DUnitRunner includes JUnit compliant XML, standard unittest text, and ascii text report for each evaluation statement.

###### 25.6.4.5.4.1.2. Example Construction Syntax

Using DUnitRunner via the command line:

```
srunk python vDUnitUnit.py evalStdOut=True evalStdOutOnlyFail=True
```

PYTHON

Using DUnitRunner via the python test script:

```
if __name__ == '__main__':
    DUnitTestRunner.DUnitRunner(evalStdOut=True, evalStdOutOnlyFail=True)
```

PYTHON

25.6.4.5.4.1.3. Related Regression Tests  
TODO: Insert link to vDUnitUnittest.rst file.

25.6.4.5.4.1.4. DUnitTestRunner Class API Documentation



*Note*

Dtest.dunit.DUnitTestRunner

25.6.4.5.5. DUnitUnittest

25.6.4.5.5.1. Class Documentation

25.6.4.5.5.1.1. Introduction

DUnitUnit extends and uses the native python unittest framework. DUnitUnit is a specialized unit tester facilitating the collection of evaluation results and its requirement identifier at the evaluation statement rather than the method.

25.6.4.5.5.1.2. Example Usage Syntax

```
from Dtest.dunit import DUnitUnittest
from Dtest.dunit import DUnitTestRunner
from Dtest.dunit.DUnitUnittest import RequirementIdentifier as Req

class vDUnitUnit(DUnitUnittest.CompassUnit):
    def test_evalEqual(self):
        expectedValue = 1
        actualValue = 1
        self.evalEqual(Req("evalEqual-Integer-01"), expectedValue, actualValue)

if __name__ == '__main__':
    DUnitTestRunner.DUnitRunner(evalStdOut=True, evalStdOutOnlyFail=True)
```

PYTHON

TODO: Additional comments concerning the code example.

25.6.4.5.5.1.3. Related Regression Tests  
TODO: Insert link to vDUnitUnittest.rst file.

25.6.4.5.5.1.4. DUnitUnittest Class API Documentation



*Note*

Dtest.dunit.DUnitTest

## 26. SiteDefs

### 26.1. Background

#### 26.1.1. Reference & Source material

- [SiteDefs Doxygen docs](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SiteDefs/html/) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SiteDefs/html/)
- [Team talks](https://dlabportal.jpl.nasa.gov/resources/darts-lab-talks/) (https://dlabportal.jpl.nasa.gov/resources/darts-lab-talks/)
- [2019, 2020 course slides](https://dlabportal.jpl.nasa.gov/documentation/coursetutorial-links/) (https://dlabportal.jpl.nasa.gov/documentation/coursetutorial-links/)
- [Sphinx docs](https://dartslab.jpl.nasa.gov/dlabdocs/) (https://dartslab.jpl.nasa.gov/dlabdocs/)
- [Jupyter notebooks](https://dlabnotebooks.jpl.nasa.gov/hub/login) (https://dlabnotebooks.jpl.nasa.gov/hub/login)
- [DARTS Lab QA site](https://dartslab.jpl.nasa.gov/qa/) (https://dartslab.jpl.nasa.gov/qa/)

### 26.2. Design

### 26.3. Usage



TBD: Flesh out and clean up this section

### 26.4. Software

### 26.5. Raw documents

### 26.6. Sphinx documentation

#### 26.6.1. Makefiles

Every YaM module is required to have a top level Makefile.yam makefile consistent with the YaM module makefile requirements. YaM uses this makefile to build individual modules. The standard look and feel of a module makefile also allows users to be able to reuse and rebuild unfamiliar YaM modules with ease. The rule requirements for a module Makefile.yam are described in `DLabDocs_Makefile_rules`. To facilitate the easy creation of a Makefile.yam for new modules, a standard set of rules are available for use in the SiteDefs module.

#### The "overall.mk" file:

This file defines several standard variables such as `YAM_ROOT`, `YAM_NATIVE` etc. and rules for exporting links to header files etc. It is virtually a requirement that this file be included the first thing in a Makefile.yam.

#### The "makefile-yam-tail.mk" file:

This file is made available for user convenience. Its use allows users to reduce the job of writing the Makefile.yam to one of setting variables that define the libraries and binaries to be built, the names of the source files etc. making their structure very simple. By accomplishing the reasonable fairly easily, it allows users to set up makefiles quickly for normal modules, while allowing the flexibility of creating specialized makefiles for complex (and typically third party) modules. This file should be included at the end of a Makefile.yam. The standard variables available for use in a *Makefile.yam* are described in

For more information about the content of module *Makefile.yam* files, please see these pages:

#### 26.6.1.1. Sandbox Makefiles Rules

Information on the rules supported by a sandbox's top level Makefile can be obtained by running the command

```
gmake help
```

from the top level. Below is a sample help message generated by running this command:

This top-level makefile provides a convenient way to loop thru all the modules in the sandbox and build the target rule for all of them. The available rules are:

```
yam-mklink: Export links for all link/work modules
yam-rmlink: Remove exported links for all link/work modules
links: Build the 'links' rule for all work modules
depends: Build the 'depends' rule for all work modules
docs: Build the 'docs' rule for all work modules
libs: Build the 'libs' rule for all work modules
libsso: Build the 'libsso' rule for all work modules
bins: Build the 'bins' rule for all work modules
clean: Build the 'clean' rule for all work modules
regtest: Run available regression tests all work modules
supp-map: Build the 'supp-map' rule for all work modules

all: Build the 'all' rule for all work modules
build: Build the 'build' rule for all work modules
```

The following 'alltgt' rules additionally allow the building of rules for all the modules for all supported targets. The ALLTGT variable can be used to restrict the 'alltgt' targets.

```
alltgt-<xxx>: Build the 'xxx' rule for all modules
quiet-alltgt-<xxx>: Build (quietly) the 'xxx' rule for all modules
bg-alltgt-<xxx>: Build (background mode) the 'xxx' rule for all modules
```

Other available utility rules are:

```
help: Generate this message
rshtest: Check for access to the remote build hosts
config: Returns the branch names for all work modules
cvs-update: Runs 'cvs update' for all the work modules
cvscheck: Runs 'cvscheck' for all the work modules
release-diffs: Runs 'yam diff' for all the work modules
```

Recognized Unix targets: hppa-hpux10 hppa-hpux9 i486-linux mips-irix5 mips-irix6.5 mips-irix6.5-gcc sparc-sunos5 sparc-sunos5.6 sparc-sunos5.7 sparc-sunos5.7-CC  
Recognized VxWorks targets: m68k-vxworks ppc-vxworks5.2 ppc-vxworks5.3

### 26.6.1.2. Module Makefiles Rules

You can get a list of the current module level Makefile.yam rules by running the command:

```
gmake -f Makefile.yam help
```

Below is the sample help message with information on the supported rules:

```
Specify one of the following module build targets:
mklink, rmlink, links, docs, libs, libsso, bins, clean, regtest, supp-map
or build => links, docs, libs, libsso, bins
or all => mklink, build

Other available rules are:
help => generates this message
alltgt-<RULE> => builds the specified RULE for
all available targets
quiet-alltgt-<RULE> => quiet version of alltgt-<RULE>
bg-alltgt-<RULE> => background mode version of alltgt-<RULE>
The ALLTGT variable can be used to restrict the 'alltgt' targets.
```

### 26.6.1.3. Makefiles Variables

This describes the various variables related to the the Darts Lab makefiles. The variables can be classified into different categories based on the extent of their definition and their variability.

#### Variables set at run-time

This class of variables persist only across the specific make invocation. They are used to override the default values of the build targets. They are defined in `DLabDocs_Makefile_runtime`.

#### Variables specific to a module

This values of this class of variables is specific to individual modules. They help to define the binaries, libraries, source files, build flags specific to the module. These are defined in Makefileyam: Module specific variables. For the special subclass of **model** modules, the makefiles are very streamlined due the uniformity of their structure. The small variability in teh build procedure is set by the variables defined in Makefileyam: for Dshell model YaM modules. By default modules are assumed to buildable for all the site build targets. However there are usually some modules that build for only a subset of the available targets. This target restriction for a module is done via the variables described in Makefileyam: supported targets and OS.

#### Variables defining the development environment and tools

This class of variables defines the development and tool environment for the builds. They specify the compilers, locations of header files, libraries etc. from third party tools. These variable values apply to all modules. Some of these variables are specific to the build target and apply to all modules and sites. They are described in Makefileyam: Target specific but site global. Variables that help to define the build variables such as compilers, compilation flags, linkers etc. are defined in Makefileyam: site and target specific build flags and variables. Since gcc is a widely used compiler, variables specific to this compiler have been grouped together and are described in Makefileyam: GCC related variables. The largest class of variables are those defining information on the supporting third party tools such as Tcl, Matlab, OpenInventor etc.that are required for builds. These are described in Makefileyam: tool related variables. This definition of this family of variables changes as the development environment evolves - new tool versions are installed and new tools start being used for the builds.

#### System configuration variables



These are variables that change only on occasion and define the general development parameters such as the available build targets, the ones available for each site etc. These are described in Makefileyam: YaM project configuration variables.

### System defined variables

This class of variables are **never** set by the user. These include variables such as the name of the build platform, the module name, directory location etc. They are made available for potential use in module makefiles. These variables are described in Makefileyam: Standard derived variables.

### Internal template variables

This class of variables are also **never** set by the user. These variables are ones created within the generic Makefile template file. These variables are described in Makefileyam: Internal variables.

#### 26.6.1.4. Makefiles Module Variables

##### 26.6.1.4.1. Project Variables

Variable	Definition	Example values	Example module
PROJ_BINS	List of binaries to build for the module	eg. Dshell++, SERoamsPy4	
PROJ_LIBS	List of libraries to build for the module	eg. libNdarts libDshell++	Ndarts
PROJ_BINS_INTERNAL	Identical to PROJ_BINS except that links are not exported for these binaries	eg. Dvalue_test, SOA_test	
PROJ_LIBS_INTERNAL	Identical to PROJ_LIBS except that links are not exported for these libraries	eg. libInternal	

##### 26.6.1.4.2. Compilation Variables

Variable	Definition	Example values	Example module
CC_SRC-<proj>	List of C source files for the <proj> target	tcl/tclCompat.c	DshellEnv
CPLUSPLUS_SRC-<proj>	List of C++ source files for the <proj> target	DshellDview.cc DshellDviewCIF.cc	DshellDview
F77_SRC-<proj>	List of Fortran source files for the <proj> target	rk45.f dop853.f	IntegratorTest
CFLAGS-<proj>	Additional compilation flags for the specific <proj> target	-DUSE_DGETSET -I\$(TCL_INCDIR)	Dmex
LIBS-<proj>	Libraries to use for building the <proj> binary or shared library	-ltclCompat \$(TCL_LIBS)	Dshtcl++
FLAVORS-<proj>	The different "flavors" to be built for the "proj" library/binary target. The object files for each flavor are built under \$(YAM_TARGET)/[FLAVOR], and the [FLAVOR] suffix is added to the names of the libraries	plain TrNdcs TrNdcs	AtbeTclMesg
FLAVORS_EXT-<proj>-<FLAVOR>	The library name "sufix" to use for the specific [proj] project's [FLAVOR] flavor. The default suffix string is [FLAVOR] itself. The "-NONE-" value specifies that an empty suffix should be used for the flavor.	-NONE-	AtbeTclMesg
BIN_USE_FLAVOR	The specific flavor object files to use for building the binaries.	plain	AtbeTclMesg
LINKER-<proj>	The linker type to use for creating binaries. The default is to use the C linker.	CPLUSPLUS	Darts++

##### 26.6.1.5. Makefile Run-Time Directives

Variable	Values	Definition	Default Value
ALLTGT	x86_64_fedora15-linux etc.	Restricts "alltgt-" rules to specified targets	All supported targets for the site
YAM_TARGET	sparc-sunos5.7-CC	Build target to really build for	\$(YAM_NATIVE)
YAM_TARGETS	x86_64_fedora15-linux etc.	All targets to build for from the "top-level"	\$(YAM_TARGET)

- Examples of "Makefile.yam" files are described in Example Makefile.yams

##### 26.6.1.6. Makefile internal details

--Makefile.yam include hierarchy--

The various files used to define this family of variables used in module **Makefile.yams** and their inclusion hierarchy is defined are described [here](http://dartslab.jpl.nasa.gov/internal/dshell/YAM_WWW/SiteDefs-Readme.contents) (http://dartslab.jpl.nasa.gov/internal/dshell/YAM\_WWW/SiteDefs-Readme.contents)

### 26.6.1.6.1. YaM project configuration variables

Variable	Values	Definition	Location
ALL_YAM_OS	unix vx	Available operating systems	site.env
ALL_YAM_TARGETS	sparc-sunos5.7 i486-linux m68k-vxworks etc.	Available build targets	site.env
unix_targets	sparc-sunos5.7 i486-linux etc.	Available unix build targets	site.env
vx_targets	m68k-vxworks ppc-vxworks5.3 etc.	Available VxWorks build targets	site.env

Variable	Values	Definition	Location
SITE_SUPPORTED_TARGETS	sparc-sunos5.7 i486-linux etc.	Targets supported for "SITE" site	[SITE]-supported.mk
SITE_UNSUPPORTED_TARGETS	hppa-hpux10 mips-irix5 etc.	Targets unsupported for "SITE" site	[SITE]-supported.mk

Variable	Values	Definition	Location
COMPILE_HOST-i486-linux	inu	Default host machine for "i486-linux" target for "SITE" site	[SITE]-supported.mk
COMPILE_HOST-sparc-sunos5.7	bertie	Default host machine for "sparc-sunos5.7" target for "SITE" site	[SITE]-supported.mk
etc. etc.			

Variable	Values	Definition	Location
YAM_BUILD_RULES	links depends libs libssso bins	component sub-rules of the "build" rule	site.env
YAM_LINKMOD_RULES	regtest install-doxygen-docs setup-doxygen-docs	build rules for both work and link modules	site.env
YAM_WORKMOD_RULES	\$(YAM_BUILD_RULES) clean docs moddeps	build rules for work modules only	site.env

Variable	Values	Definition	Location
RSHCMD	ssh -x	Command to use to login to remote build hosts	site.env

### 26.6.1.6.2. Makefile.yam for Dshell model YaM modules

Variable	Values	Definition	Example
ADDT_CSRCs	load_stars_meas.c starlkup.c	Additional C source files for including in the models' library	MpfModels
ADDT_CPLUSPLUSRCs		Additional C++ source files for including in the models' library	None so far
ADDT_INCLUDES	-I\$(YAM_ROOT)/include/libMSIM	Additional compilation flags for building the models	RoverppModels
ADDT_SOLIBS	-lKsolv -lLinAlg	Additional libraries to link into the models' shared library	Ksolv
NEEDS_DARTSPP	true	Flag to signify that the models require Darts++ and cannot be used with Darts	RoverppModels
SKIP_MEX_BUILDS	true	Flag to avoid building the ModelMex wrappers for the models	DhssModels

### 26.6.1.6.3. Makefile.yam: supported targets and OS

\*CAUTION: It is highly recommended that when the variables described below need to set, their setting be conditioned upon other variables in the site-config-xx files.\* The rationale is as follows. A typical reason for needing to restrict the list of supported targets for a module is because a required tool/library (eg. Matlab, OpenInventor) is not available. Using an entry such as

```
ifneq ($(HAVE_MATLAB),true)    MODULE_UNSUPPORTED_TARGETS = $(YAM_TARGET) endif`
```

captures and also documents this dependency. Thus when the required tool does become available at a future date for a new target, the module automatically is geared to support the target.

Variable	Values	Definition	Location	Example
MODULE_SUPPORTED_OS		The list of OS' supported for the module. The default value is "unix vx", i.e. all OS are assumed to be supported.	.supported.mk	None so far
MODULE_SUPPORTED_TARGETS	sparc-sunos5.7	The list of targets supported for the module. The default value is \$(SITE_SUPPORTED_TARGETS), i.e. all targets are assumed to be supported.	.supported.mk	Dmex
MODULE_UNSUPPORTED_OS	vx	The list of OS' not supported for the module. The default value is "".	.supported.mk	Dspace
MODULE_UNSUPPORTED_TARGETS	i486-linux	The list of targets not supported for the module. The default value is "".	.supported.mk	aejTools

#### 26.6.1.6.4. Makefile: Run-time directives

Variable	Values	Definition	Example
ALLTGT	sparc-sunos5.7 i486-linux etc.	Restricts "alltgt-" rules to specified targets	All supported targets for the site
YAM_TARGET	sparc-sunos5.7-CC or m68k-vxworks	Build target to really build for	\$(YAM_NATIVE)
YAM_TARGETS	sparc-sunos5.7 i486-linux etc.	All targets to build for from the "top-level"	\$(YAM_TARGET)

#### 26.6.1.6.5. Makefileyam: Target specific but site global

The variables below are meant to be target-specific, whose values are valid for all sites and modules.

Variable	Example value	Definition	Location
CC_DEFINES	-DSUNOS5	Compilation flag to be used for C builds.	[TARGET].mk
CC_LIBS	-lm -lnsl -lsocket -lpix4	Standard list of libraries to be used for C linking.	[TARGET].mk

Variable	Example value	Definition	Location
CPLUSPLUS_DEFINES	-DSUNOS5	Compilation flag to be used for C++ builds.	[TARGET].mk
CPLUSPLUS_LIBS	-lm -lnsl -lsocket -lpix4	Standard list of libraries to be used for C++ linking.	[TARGET].mk

Variable	Example value	Definition	Location
RTI_OS	vx	RTI's equivalent OS name for the [TARGET] build target.	[TARGET].mk
RTI_TARGET	sparcSol2.6	RTI's equivalent name for the [TARGET] build target.	[TARGET].mk
SHARED_COMPILE_FLAG	-KPIC	The compiler specific flag to use for building relocatable object files when building shared libraries for the [TARGET] target.	[TARGET].mk
MEXEXT	mexsol	The Mex extension to use for the [TARGET] target.	[TARGET].mk

#### 26.6.1.6.6. Makefileyam: site and target specific build flags and variables

Variable	Example value	Definition
BUILDING_SHARED_LIBS	true	When set to true, shared libraries are built for the module.
LD_SHARED	\$(LD) -G	The linker to use for building a shared library.

Variable	Example value	Definition
SHARED_LIBDIR	/home/atbe/pkgs/lib/\$(YAM_TARGET)-shared	Location of external tool shared libraries.

Variable	Example value	Definition
CC	/usr/bin/gcc	The C compiler
CC_DEPEND_FLAG	-MM	C compiler flag for generating dependency information.
CC_EXTRA_SHLIB	-lgcc	Library to link into shared libraries.
CC_OPTIMIZATION	-g -O3	Optimization flags for compiling
CC_WARNINGS	-Wall -Wstrict-prototypes	Compiler flags for controlling warning messages.

Variable	Example value	Definition
CPLUSPLUS	/usr/bin/gcc	C++ compiler
CPLUSPLUS_DEPEND_FLAG	-MM	C++ compiler flag for generating dependency information.
CPLUSPLUS_EXTRA_SHLIB	-lgcc	Library to link into shared libraries.
CPLUSPLUS_OPTIMIZATION	-g -O3	Optimization flags for compiling
CPLUSPLUS_WARNINGS	-Wall -Wstrict-prototypes -Woverloaded-virtual	Compiler flags for controlling warning messages.
LIBSTDCPP	-lstdc++	Standard C++ library to be used during linking.

Variable	Example value	Definition
F77	/usr/bin/gcc	Fortran compiler
HAVE_G77	true	Is the Fortran compiler available?
LIBF77	-lg2c	Library to link for Fortran object files.

Variable	Example value	Definition	Location
LIBKCS	-lkcs	The name of the kcs library needed by StethoScope for some platforms.	[TARGET].mk
_PROF	true	Flag to turn on code profiling during compilation.	site-config-xx

#### 26.6.1.6.7. Makefile.yam: tool related variables

Variable	Values	Definition	Location
HAVE_DOXYGEN			
DOXYGEN			
DOXYGEN_DOCS_DIR			
WWW_DOCS_SUBDIR			
WWW_URL			

Variable	Values	Definition	Location
PERL			

Variable	Values	Definition	Location
HAVE_X			
X11_INCDIR			

Variable	Values	Definition	Location
HAVE_TCL			

Variable	Values	Definition	Location
TCLSH			
TCL_INCDIR			
TCL_LIBDIR			
TCL_LIBS			
TCL_SRCDIR			
TCL_VERSION			
USE_TCLTK80			
HAVE_TK			
TK_INCDIR			
TK_LIBDIR			
TK_LIBS			
TK_SRCDIR			
TK_VERSION			
HAVE_TIX			
TIX_INCDIR			
TIX_LIBDIR			
TIX_LIBS			

Variable	Values	Definition	Location
HAVE_IV			
IV_FL_LIB			
IV_IMG_LIB			
IV_INCDIR			
IV_LIB			
IV_LIBS			
IV_LIB_DIR			
IV_XT_LIB			

Variable	Values	Definition	Location
HAVE_MATHEMATICA			
MATHEMATICA_BASEDIR			
MATHEMATICA_BINDIR			
MATHEMATICA_INCDIR			
MATHEMATICA_LIBDIR			
MATHEMATICA_LIBS			

Variable	Values	Definition	Location
HAVE_MATLAB			
MATLAB			
MATLAB42c			
MATLAB_DIR			

Variable	Values	Definition	Location
MATLAB_INCDIR			
MATLAB_LIBDIR			
MATLAB_LIBS			
MATLAB_TARGET			
HAVE_SIMULINK			
SIMULINK_INCDI			
HAVE_RTW			
RTW_INCDIR			

Variable	Values	Definition	Location
HAVE_MESA			
MESA_DIR			
MESA_INCDIR			
MESA_LIBS			

Variable	Values	Definition	Location
GL_LIBS			
JPEG_LIBS			

Variable	Values	Definition	Location
HAVE_MOTIF			
MOTIF_INCDIR			
MOTIF_LIBS			

Variable	Values	Definition	Location
RTIHOME			
RTLIBHOM			
HAVE_SCOPE			
SCOPEGCCEXT			
SCOPE_INCDIR			
SCOPE_LIBS			
STETHOSCOPEHOME			
HAVE_NDDS			
NDDSHOME			
NDDS_INCDIR			
NDDS_LIBS			
NDDS_SOLIBS			

Variable	Values	Definition	Location
HAVE_LIBIPC			
HAVE_LIBSITE			
USE_SITE_TERRAIN			

Variable	Values	Definition	Location
HAVE_TRAMEL			
TRAMEL_INCDIR			
TRAMEL_LIBDIR			
TRAMEL_LIBS			

Variable	Values	Definition	Location
HAVE_TMATH			
TMATH_INCDIR			
TMATH_LIBDIR			
TMATH_LIBS			

Variable	Values	Definition	Location
M4			
SDFBIN			

#### 26.6.1.6.8. Makefile.yam: Internal variables

Internal variables in `makefile-yam.mk`

```

ALLBINS ALLLIBS BUILD_SHARED_LIB BUILD_STATIC_LIB
CMEX_FLAGS COMMON_DIR CRLINKS URDIR DATETAG DEPPFILES
DOXYGEN_EXCLUDE_PATTERNS DOXYGEN_EXTRACT_ALL DOXYGEN_PREDEFINED
DOXYGEN_RULE DOXYGEN_WARN_FILE F77_COMPILE_FLAGS ECHOVARS FAPP
FILT_ALLTGT FLAVOR FLAVOR_EXT INSTALLDOX_STR IS_VALID_TARGET LIB_SUFFIX
LINKDIR_BIN_LINKS LINKDIR_BIN_MODULE_LINKS LINKDIR_DOC_LINKS
LINKDIR_DOC_MODULE_LINKS LINKDIR_ETC_LINKS LINKDIR_ETC_MODULE_LINKS
LINKDIR_INC_LINKS LINKDIR_INC_MODULE_LINKS LINKDIR_LIB_LINKS
LINKDIR_LIB_MODULE_LINKS LINKER MAKEFLAGS MDLHDRS MDLLIST MDLTCL MEX
MEXFILES MEXLDFLAGS MEXLIST MODELS MODULE_DIR NATIVE_SYS OBJ
OBJ_YAMVERSION POD2MAN POD2TEXT PROJ PROJ_TARGETS RELLNK_DIR
RELMODLNK_DIR RELTGLNK_DIR SHARED_LIBS_FLAGS SITECONFIGFILE
SITEDEFSHOME SITE_SUPPORTED_OS SITE_UNSUPPORTED_OS SUFFIXES
TAGFILES_EXPANDED TOP_LINKDIR_TYPES TRACE_XARGS USE_XARGS VALID_STATUS
binflavs blinkfile flavext flinkfile hostsstr libflavs HAVE_MEXAUTOGEN
GCCFLAG MAKE INTERVALS
MAKE MAKECMDGOALS MAKEFILES
MAKEINFO MAKELEVEL MAKE_COMMAND MAKE_VERSION

```

#### 26.6.1.6.9. Makefile.yam: Module specific variables

Variable	Example value	Definition	Example module
MODULE_COMPILE_FLAGS	\$(TCL_INCDIR)	Compilation flags to use for all the object files for the module.	Darts++
MODULE_LIBS	\$(LIBSTDCPP)	Libraries to be appended to CC_LIBS and CPLUPLUS_LIBS for linking all shared libraries and binaries for the module.	MathUtils
MODULE_LINK_FLAGS	-Lsrc/graphics	Link time flags to use for all the shared libraries and binaries for the module.	
MODULE_LINKER	\$(CPLUSPLUS)	Linker to use for all the shared libraries and binaries for the module.	MathUtils

Variable	Values	Definition	Example
SKIP_STD_BINS	true	Directive to skip the standard built-in "bins" rule.	Dmex
SKIP_STD_LIBS	true	Directive to skip the standard built-in "libs" rule.	mathc90
SKIP_STD_LIBSSO	true	Directive to skip the standard built-in "libss" rule.	mathc90
SKIP_STD_CLEAN	true	Directive to skip the standard built-in "clean" rule.	libMSIM

Variable	Values	Definition	Example
SKIP_YAM_VERSION	true	Directive to skip linking in YamVersion.o.	qhull

\*CAUTION: Do not set these link export variables directly!!! Only append to them.\*

Variable	Values	Definition	Example
BIN_LINKS	AcsFswProto_start	List of file links to be exported to the top level bin/ directory	AcsFswProto
BIN_MODULE_LINKS		List of file links to be exported to the top level bin/\$(MODULE_NAME) directory	None so far
BIN_TARGET_LINKS	Acs/\$(YAM_TARGET)/acsif	List of file links to be exported to the top level bin/\$(YAM_TARGET) directory	RsrAcsModels
BIN_sparc-sunos5.7_LINKS		List of file links to be exported to the top level bin/sparc-sunos5.7 directory only. This flag is used only when \$(YAM_TARGET) matches the specified ttarget.	None so far
DOC_LINKS		List of file links to be exported to the top level doc/ directory	None so far
DOC_MODULE_LINKS	doc/Doxyfile-generic	List of file links to be exported to the top level doc/\$(MODULE_NAME) directory	DshellEnv
ETC_LINKS		List of file links to be exported to the top level etc/ directory	None so far
ETC_MODULE_LINKS	tests	List of file links to be exported to the top level etc/\$(MODULE_NAME) directory	Dvalue
INC_LINKS	Qmv/include/qmv	List of file links to be exported to the top level include/ directory	EphemPropagatorModels
INC_MODULE_LINKS	DshellDviewCIF.h DshellDview.h	List of file links to be exported to the top level include/\$(MODULE_NAME) directory	DshellDview
LIB_LINKS	\$(wildcard tcl/lib/tcl*)	List of file links to be exported to the top level lib/ directory	thirdParty
LIB_MODULE_LINKS	src/DspaceLib.tcl	List of file links to be exported to the top level lib/\$(MODULE_NAME) directory	Dspace
LIB_TARGET_LINKS	\$(YAM_TARGET)/libsitCIF.so	List of file links to be exported to the top level lib/\$(YAM_TARGET) directory	libMSIM
LIB_sparc-sunos5.7_LINKS		List of file links to be exported to the top level lib/sparc-sunos5.7 directory only. This flag is used only when \$(YAM_TARGET) matches the specified ttarget.	None so far

#### 26.6.1.6.10. Makefile.yam: GCC related variables

Variable	Values	Definition	Location
USE_GCC272	true	If set to true, then gcc 2.7.2 is used (obsolete)	site.gcc
USE_GCC295	true	If set to true, then a gcc 2.95 class compiler is used	site.gcc
GCC_VERSION	2.95.1	The gcc version	site.gcc
LIBGCC	-lgcc	The gcc library	site.gcc
GCC_EXEC_PREFIX	/home/atbe/pkgs/lib/gcc-lib/sparc-sun-solaris2.7/2.95.2	Path for the gcc supporting binaries (eg. cpp)	site.gcc
FPERMISSIVE_OPT	-fpermissive	Flag to turn off warnings from X header files	site.gcc
FWRITABLE_STRINGS_OPT	-fwritable-strings	Flag to allow the writing into string constant areas. Needed by Td 7.4.	site.gcc



Variable	Values	Definition	Location
FEXTERNAL_TEMPLATES	-fexternal-templates	Flag needed at times for building with templates	site.gcc

#### 26.6.1.6.11. Makefile.yam: Standard derived variables

Variable	Values	Definition	Location
YAM_ROOT	Directory path	Full directory path to the sandbox	overall.mk
YAM_OS	unix or vx	The operating system (Unix or VxWorks)	site.env
YAM_NATIVE	eg. sparc-sunos5.7	The name of the build platform determined using "uname -a"	yamNative.mk
YAM_TARGET	eg. sparc-sunos5.7	The name of the target platform, default is \$(YAM_NATIVE)	yamNative.mk
YAM_VERSIONS	/home/atbe/repo/YaM/Dshell/Module-Releases	Directory path to the module release area	site.env
YAM_VERSION	eg. Darts-R3-45d	The module version string extracted from YamVersion.h	bldRules.mk
LOCAL_DIR	Full path to the module location	This is \$(YAM_ROOT)/src/[Module] for work modules, and is \$(YAM_VERSIONS)/[Module]/[Module-Release]	localDir.mk
MODULE_NAME	eg. Dshell++	The name of the module	moduleVars.mk
MODULE_DIR	eg. Dshell++ or Darts-R3-33g	The basename of the module directory. It is \$(MODULE_NAME) for work modules, and includes the release suffix for link modules	moduleVars.mk
MODULE_TYPE	work or link	Specifies whether a module is a "work" or a "link" module	moduleVars.mk

Variable	Values	Definition	Location
CC_INCLUDES	-\$\$(YAM_ROOT)/include	Include paths to use during C compilation.	bldFlags.mk
CC_COMPILE_FLAGS	\$(CC_OPTIMIZATION) \ \$(CC_DEFINES) \$(CC_WARNINGS) \$(CC_INCLUDES)	Flags to use during C compilation.	bldFlags.mk
CC_LINK_FLAGS	-\$\$(YAM_ROOT)/lib/\$\$(YAM_TARGET)	Flags to use during C linking.	bldFlags.mk

Variable	Values	Definition	Location
CPLUSPLUS_INCLUDES	-\$\$(YAM_ROOT)/include	Include paths to use during C++ compilation.	bldFlags.mk
CPLUSPLUS_COMPILE_FLAGS	\$(CPLUSPLUS_OPTIMIZATION) \ \$(CPLUSPLUS_DEFINES) \$(CPLUSPLUS_WARNINGS) \$(CPLUSPLUS_INCLUDES)	Flags to use during C++ compilation.	bldFlags.mk
CPLUSPLUS_LINK_FLAGS	-\$\$(YAM_ROOT)/lib/\$\$(YAM_TARGET)	Flags to use during C++ linking.	bldFlags.mk

Variable	Values	Definition	Location
AR	ar	The "ar" tool to use for building libraries	bldFlags.mk
AR_FLAGS	r	The flags to pass to \$(AR)	bldFlags.mk
LD	ld	The "ld" tool to use for building shared libraries	bldFlags.mk
RANLIB	true	The ranlib tool (pretty much obsolete these days)	bldFlags.mk

## 27. pyam

### 27.1. Background

#### 27.1.1. Reference & Source material

- [pyam Doxygen docs](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/pyam/html/) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/pyam/html/)
- [Team talks](https://dlabportal.jpl.nasa.gov/resources/darts-lab-talks/) (https://dlabportal.jpl.nasa.gov/resources/darts-lab-talks/)
- [2019, 2020 course slides](https://dlabportal.jpl.nasa.gov/documentation/coursetutorial-links/) (https://dlabportal.jpl.nasa.gov/documentation/coursetutorial-links/)
- [Sphinx docs](https://dartslab.jpl.nasa.gov/dlabdocs/) (https://dartslab.jpl.nasa.gov/dlabdocs/)
- [Jupyter notebooks](https://dlabnotebooks.jpl.nasa.gov/hub/login) (https://dlabnotebooks.jpl.nasa.gov/hub/login)
- [DARTS Lab QA site](https://dartslab.jpl.nasa.gov/qa/) (https://dartslab.jpl.nasa.gov/qa/)

### 27.2. Design

### 27.3. Usage

#### 27.3.1. Using baseline package releases

The `--baseline` option exists on `pyam`'s `register-new-package`, `save-package` and `sync` commands.

For the `register-new-package` command, the `--baseline` option will create a new `BaselinePkg` package with just the `SiteDefs` module. This package will be used for baseline releases. The specification of a package name, or a list of defining modules, are not allowed with the `--baseline` option. The definition of the package's with just a single module is just a stub definition. All releases of this specific package (aka `baseline mode`) will require the user to specify a `YAM.config` file with tagged module entries for the actual list of modules and releases making up the baseline release. This command is expected to be used just once to start the use of the baselines workflow. A typical use of this command is

```
# one time to enable use of baseline releases
pyam register-new-package --baseline
```

#### 27.3.1.1. Creating a baseline package release

For the `save-package` command, the `--baseline` option will make a release of the `BaselinePkg` package. The use of the `--baseline` option will require the user to specify a `YAM.config` file with the modules and releases that are to be used for the baseline release. The use of the baseline model will disallow the specification of the package name for this command since the `BaselinePkg` package name will be used automatically. It is expected that command will be used periodically to record baseline releases when the regression test passing percentage is above a specified threshold. There is currently no automated invocation of such releases. Typical uses of this command is (note that no package name is specified with the `--baseline` option):

```
# normal pkg release with bleeding edge module releases
pyam save-package <mypkg>

# normal pkg release with specific module releases
pyam save-package <mypkg> \
  -c /home/dlab3/repo/PKGBUILDS/ROAMSDshellPkg/BASELINE-FC36/YAM.config

# make baseline package release
pyam save-package --baseline \
  -c /home/dlab3/repo/PKGBUILDS/ROAMSDshellPkg/BASELINE-FC36/YAM.config
```

#### 27.3.1.2. Syncing to a baseline package release

For the `sync` command, the `--baseline` option will cause the module syncing to use the module releases associated with the specified baseline release for each module sync. In baseline mode, the `--release` option value is used to specify the baseline release version rather than the module release version. When the `--revision` option is not specified, the latest baseline release is used. Without the `--baseline` option, we get back the old behavior where syncs are done to the bleeding edge. Typical uses of this command are

```
# sync all modules in a sandbox to the bleeding edge
pyam sync --all

# sync all modules in a sandbox to the latest baseline release
pyam sync --all --baseline

# sync all modules in a sandbox to a specific baseline release
pyam sync --all --baseline --release R1-00b
```

#### 27.3.1.3. Updating a `YAM.config` file to a baseline package release

For the `config` command, the `--baseline` option will cause the new config file to have releases for the modules in the input config file to be from the specified baseline release specified by the `--release` option. If no `--release` option is specified, then the latest baseline release is used. Example uses of this command are:

```
# normal use to transform a config file with options
pyam config -c <sbx>/YAM.config -o new-YAM.config <options>

# transform a config file with module releases from the latest baseline release
pyam config -c <sbx>/YAM.config -o new-YAM.config --baseline

# transform a config file with module releases from a specific baseline release
pyam config -c <sbx>/YAM.config -o new-YAM.config --baseline --release R1-00b

# transform a config file with module releases from the latest baseline releases
# along with transformation options
pyam config -c <sbx>/YAM.config -o new-YAM.config --baseline --release R1-00b \
--all-to-link
```

#### 27.3.1.4. Creating a baseline package release sandbox

To create a sandbox for a package with modules from a stable baseline release instead of the bleeding edge you would do this in two steps: - Create the package sandbox from the bleeding edge as usual using `pyam setup <mypkg>` - Run `pyam sync --all --baseline` in the sandbox to change the module versions to a baseline release

Creating such a baseline based sandbox for a `docker` based external release can be done as well as in the following example:

```
# check out the DAREPkg package with bleeding edge module releases
dockerSandbox.py --pkg DAREPkg --src_dir /home/dlabdriveS/repo/docker/src

# check out the DAREPkg package with module releases from the latest baseline
dockerSandbox.py --pkg DAREPkg --src_dir /home/dlabdriveS/repo/docker/src --baseline -

# check out the DAREPkg package with module releases from the specific R1-00a baseline
dockerSandbox.py --pkg DAREPkg --src_dir /home/dlabdriveS/repo/docker/src --baseline R1-00a
```

### 27.3.2. Recipes for using git based pyam modules

Usage questions.

#### 27.3.2.1. Handy references

- [Git Merge vs Rebase: The Three Types of Merge](https://itnext.io/git-merge-vs-rebase-938950fb218) (https://itnext.io/git-merge-vs-rebase-938950fb218) article

##### 27.3.2.1.1. What is the difference between git rebase, git pull rebase and git config pull --rebase etc?

Various commands possibilities and options:

- `git rebase`
- `git merge`
- `git merge --ff`
- `git squash`
- `git pull rebase`
- `git pull ff`
- `git config pull --rebase true/false`
- `git config pull --ff`

We need to clarify when to use. The first four appear to be for the local repo, the the remainder have to do with multiple repos.

##### 27.3.2.2. How do we set up git credentials to avoid endless prompts for passwords?

There are a few common ways to cache `git` credentials:

1. With the `gnome-keyring`
2. In an encrypted file. Here, we will be using GPG.
3. As plain text.

The `gnome-keyring` is likely the safest and best option if you are already using the `gnome-keyring`. If you are accessing machines via passwordless SSH without an X display, and are not unlocking the `gnome-keyring`, it may be worth pursuing the other two options. The plain text option is the least safe but the easiest option to set up. If you are going to use this, I highly recommend setting a short timeout window: less than or equal to 15 minutes.

Useful resources:

- [Here](https://stackoverflow.com/questions/5343068/is-there-a-way-to-cache-https-credentials-for-pushing-commits) (https://stackoverflow.com/questions/5343068/is-there-a-way-to-cache-https-credentials-for-pushing-commits) is a stack overflow page on git credentials.

##### 27.3.2.2.1. Gnome keyring

To use the `gnome-keyring` for password caching, use the `libsecret` git credential store. On Fedora, this can be installed using

```
sudo dnf install git-credential-libsecret
```

Enable `libsecret` credential storage can be done with

```
git config --global credential.helper /usr/libexec/git-core/git-credential-libsecret
```

#### 27.3.2.2.2. GPG encrypted file

Caching `git` credentials using an encrypted file can be done using GPG and the `netrc` `git` credential manager. Steps for setting this up can be found in [Q&A 1892](https://dartslab.jpl.nasa.gov/qa/1892/how-do-i-encrypt-my-git-credentials-with-gpg) (<https://dartslab.jpl.nasa.gov/qa/1892/how-do-i-encrypt-my-git-credentials-with-gpg>).

#### 27.3.2.2.3. Plain text

Caching `git` credentials using plain text is the easiest to set up, but the least safe. You can enable it by running

```
git config --global credential.helper cache
```

The default timeout window is 15 minutes. If you would like to modify that, you can run

```
git config --global credential.helper "cache --timeout=<timeout_window_in_seconds>"
```

#### 27.3.2.3. How do we set up SSL certificates?

[Q&A 1887](https://dartslab.jpl.nasa.gov/qa/1887/how-do-i-add-a-self-signed-certificate-to-git) (<https://dartslab.jpl.nasa.gov/qa/1887/how-do-i-add-a-self-signed-certificate-to-git>) shows how to setup SSL certificates for `git`.

#### 27.3.2.4. How does one create a `git` hosted `pyam` module?

All `git`-hosted `pyam` modules should use a repository keyword that has `git` or `Git` in the name or a repository keyword that has been specifically listed as using a `git` version control system.

##### 27.3.2.4.1. On DLAB machines

Choose a repository keyword for your `git` module that has either `git` or `Git` in the name or that is listed as a keyword that uses the `git` version control system in `pyam`. Then, run

```
pyam register-new-module --repository-keyword <repo_keyword> <module_name>
```



TBD: Need to add info on how to modify things so only certain groups can access the module.

##### 27.3.2.4.2. Hosted elsewhere (GitLab, GitHub, etc.)

Create an empty repository in the are you would like to host it on. Then, add a repository keyword to the `pyamrc` file whose URL contains everything but the module name: you should use the `https://` URL. For example, if the module's URL for a module called `MyModule` was `https://my-gitlab-server.com/projects/modules/MyModule`, then the repository keyword should be `https://my-gitlab-server.com/projects/modules`

Then, register the new `pyam` module using

```
pyam register-new-module --repository-keyword <repo_keyword> <module_name>
```

#### 27.3.2.5. How to get `git` to work from outside a checked out folder?

`svn` command often optionally take the path to a checked out folder as an argument. So they can be run from anywhere, so you can run `svn status` when within a folder; and run `svn status <folder>` when outside.

`git` commands work similarly when within a checked out folder. However, when outside the folder, we can use `git -C <folder_path> ...` usage to specify the folder to 'step into' before running the `git` command. For example, one can run `git -C <folder_path> status` to check the status of a folder without being within the folder.

#### 27.3.2.6. How do we find out the location/URL of the repository used for the checkout?

For `svn` you can get such info by running `svn info`.

For `git` you can run `git remote show origin`.

#### 27.3.2.7. How do we get rid of local changes and revert back to checked out code?

For `svn` you can do `svn revert file1 file2 ...` or `svn revert -R ..`

For `git` you would do `git restor file1 file2 ...` or `git restore ..`

#### 27.3.2.8. How do we check the branch that a checked out module is on?

For `svn` you can run `svn info` and check the URL to get the branch info.

```
git -C <module path> branch
```

#### 27.3.2.9. How do we switch the branch of the module in place to a different pre-existing one using `git`?

For `svn` we can use the `svn switch ...` command to switch in-place to a different branch.

To switch to a different branch, run

```
git checkout <branch_name>
```

To get a list of all branches for a module, run

```
git branch -a
```

### 27.3.2.10. How do we directly check out an existing branch/tagged/dead/main trunk version of a module using git?

For low level debugging and fixing things, we sometimes need to be able to step down to the version control system (like we can do with `svn checkout ...`) to check out code.

First, clone the `git` module. To do so, you will need the module's URL. If you have a copy of the module somewhere, you can use

```
git remote get-url origin
```

to obtain it's URL. Otherwise, you can use the `git` module's repository keyword from `pyam` to find out where it lives. Then, use

```
git clone <URL>
```

to obtain a local copy of the `main` branch. If you want to be on a different branch/tag than `main`, then use `git checkout` to switch to whichever branch/tag you would like.

### 27.3.2.11. How do we get a list of the previous feature branches, dead branches, and releases using git?

For `svn`, we can do this by running `svn ls <repo_path>` where `repo_path` points to the release, feature branches and dead branches of the SVN repository for a module.

One can obtain a list of all branches for a git repository by running `git branch -a`. All dead branches have the pattern `*_dead` to identify them as dead branches. The remaining branches are feature branches.

When modules are saved (released) using `pyam`, they are tagged in git. One can get a list of git tags by running `git tag` in the module.

### 27.3.2.12. What do we do if we have committed a file that should not be on revision control?

For `svn`, we can use `svn rm ...` followed by a commit to remove unwanted files.

1. Remove that file from revision control using `git rm --cached <file>`
2. If this file will often exist, but should not be on revision control, e.g., built libraries, then add it to the `.gitignore` file.

### 27.3.2.13. How do we move/rename a repository?

This describes how to move/rename a repository. This will copy over all of the history of the original module, but it will not be connected to the main branch, i.e., all of the old branches/tags will be there, but they will not be connected to the main branch like they were in the original repository.

1. Create the module at the new location with the new name the same way you would register any new `pyam` module.
2. `git clone --mirror <old-repo-url> <new-repo-name>`
3. `cd <new-repo-name>; git remote remove origin`
4. `git remote add origin <new-repo-url>`
5. `git push --all; git push --tags` This step is going to give warnings/errors about the main branch. This will essentially push everything except for the main branch, since the histories of the two repo's main branches are different.
6. `pyam checkout <new-repo-name>`
7. `git pull`
8. `git checkout <most-recent-tag-of-old-repo> - .`
9. `git commit`
10. `git push`
11. `pyam save <new-repo-name>`

[This](https://itnext.io/git-repository-transfer-keeping-all-history-670fe04cd5e4) (<https://itnext.io/git-repository-transfer-keeping-all-history-670fe04cd5e4>) and [this](https://stackoverflow.com/questions/20790014/git-copy-source-code-to-new-branch-without-history) (<https://stackoverflow.com/questions/20790014/git-copy-source-code-to-new-branch-without-history>) were useful when creating this answer:

### 27.3.2.14. Mirroring git repositories

This guide will help you mirror a `pyam` git repository to a GitLab repository. The benefits of doing this are: \* External users can work directly with git and don't need to work with `pyam`. They do this by branching off of main, and merging back into main whenever they are ready to make their changes live. \* Issues in the GitLab repository can be referenced in the commits and vice-versa. This makes it a lot easier to track which code changes go with which issues.

To start the process, first you must have a module you have created with `pyam`. We will call this module `MY_MODULE` for the rest of the guide. Then, create a project on the DARTS GitLab server that will be associated with this module.

#### 27.3.2.14.1. Pull mirroring

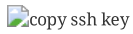
1. On the GitLab page for your project, go to Settings → Repository → Mirroring repositories.
2. Enter your GitLab URL. This URL should include [your username](#). For example, `ssh://MY_USERNAME@darts1ab.jpl.nasa.gov/alt/SVN/git/Modules/MY_MODULE`.
3. Change mirror direction to Pull.
4. Click Detect host keys
5. Set the authentication method to SSH public key.
6. Check Mirror only protected branches.
7. Click Mirror repository. Note, this will create the mirror try to do a pull immediately, which will fail since we have not added the GitLab repository's SSH key to the list of authorized SSH keys.
8. Click on Copy SSH public key.



9. Paste this SSH public key into your user's authorized SSH keys file, i.e., `/home/leake/MY_USERNAME/authorized_keys`.

#### 27.3.2.14.2. Push mirroring

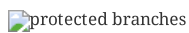
1. On the GitLab page for your project, go to Settings → Repository → Mirroring repositories.
2. Enter your GitLab URL. This URL should include your username. For example, `ssh://MY_USERNAME@dartslab.jpl.nasa.gov/alt/SVN/git/Modules/MY_MODULE`.
3. Change mirror direction to Push.
4. Click Detect host keys
5. Set the authentication method to SSH public key.
6. Check Mirror only protected branches.
7. Click Mirror repository.
8. Click on Copy SSH public key.



9. Paste this SSH public key into your user's authorized SSH keys file, i.e., `/home/leake/MY_USERNAME/authorized_keys`.

#### 27.3.2.14.3. Protected branches

Per the settings above, the repository will only mirror protected branches. It is recommended you setup your protected branches as follows:



#### 27.3.2.14.4. Pyam releases via CD pipeline

As mentioned in the benefits earlier, we want our GitLab repository to automatically make pyam releases whenever we commit to main. To do that, we can create a GitLab pipeline. We do this by adding a `.gitlab-ci.yml` to the main directory of the repository. Inside, the text should contain:

```
variables:
  PYAM: /home/dlab/pkgs/x86_64-fedora36-linux/bin/pyam
  MODULE_NAME: MY_MODULE

merge_pyam:
  rules:
    - if: $CI_COMMIT_BRANCH == "main"
  script:
    - umask 002 # Set umask correctly so that files are group writable
    - sleep 60 # Give the push mirror 60 seconds to update the pyam git repository
    - export YAM_SITE='telerobotics'
    - export YAM_SEND_EXCEPTION_TO_HOST_PORT_EMAIL='smtp.jpl.nasa.gov:25:jain@jpl.nasa.gov'
    - export YAM_PROJECT_CONFIG_DIR='/home/dlab/repo/yamConfigs'
    - export YAM_PROJECT='Dshell'
    - export YAM_VERSIONS='/home/atbe/repo/YaM/Dshell/Module-Releases'
    - git config --global user.email "gitlab-runner@jpl.nasa.gov"
    - git config --global user.name "gitlab-runner"
    - mkdir tmpSandbox
    - cd tmpSandbox
    - mkdir common
    - touch common/YAM.modules
    - touch YAM.config
    - touch Makefile
    - $PYAM checkout --no-build $MODULE_NAME
    - cd src/$MODULE_NAME
    - git merge origin/main
    - git add -A
    - git diff-index --quiet HEAD || git commit . -m "Automatic release from merge with main on GitLab."
    - git push
    - cd ..
    - $PYAM save $MODULE_NAME
```

where `MY_MODULE` in the variables section is replaced with the name of your module.

## 27.4. Software

## 27.5. Raw documents

## Model modules

## 28. DshellCommonModels Dshell model library

### Doxygen reference to Models in DshellCommonModels by Type

- [Actuators](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/DshellCommonModels/html/group__Actuator_group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/DshellCommonModels/html/group\_\_Actuator\_group.html)
- [Sensors](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/DshellCommonModels/html/group__Sensor_group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/DshellCommonModels/html/group\_\_Sensor\_group.html)
- [Motors](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/DshellCommonModels/html/group__Motor_group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/DshellCommonModels/html/group\_\_Motor\_group.html)
- [Encoders](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/DshellCommonModels/html/group__Encoder_group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/DshellCommonModels/html/group\_\_Encoder\_group.html)
- [Flows](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/DshellCommonModels/html/group__Flow_group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/DshellCommonModels/html/group\_\_Flow\_group.html)
- [Continuous](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/DshellCommonModels/html/group__ContStates_group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/DshellCommonModels/html/group\_\_ContStates\_group.html)

### 28.1. DshellCommonModels::Accelerometer Sensor Model



This model documentation is auto-generated. Documentation updates should be made to the model's `mdl` file to avoid the changes being lost.

[Doxygen link to all DshellCommonModels models of Sensor class](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/DshellCommonModels/html/group__Sensor_group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/DshellCommonModels/html/group\_\_Sensor\_group.html)

#### Description

models accelerometer measurements

from the DshellCommonModels models library for the xref:Dshellpp\_module simulation framework.

#### Model Accelerometer class details

For more information on the members and functions of this model class, please see [DshellCommonModels::Accelerometer model class doxygen documentation](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/DshellCommonModels/html/group__Accelerometer_group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/DshellCommonModels/html/group\_\_Accelerometer\_group.html)

#### Enums

##### cntIntType Enum

In the Accelerometer model definition, the original enum cntIntType is defined as:

```
enum cntIntType
{
    8BIT = 0,
    16BIT = 1,
    32BIT = 2
};
```

In the C model code, these enum values can be accessed by using the follow C enum definitions:

```
/// AccelerometercntIntType enum
enum AccelerometercntIntType
{
    ACCELEROMETER_CNT_INT_TYPE_8BIT = 0,
    ACCELEROMETER_CNT_INT_TYPE_16BIT = 1,
    ACCELEROMETER_CNT_INT_TYPE_32BIT = 2
};
```

##### stateTypeEnum Enum

In the Accelerometer model definition, the original enum stateTypeEnum is defined as:

```
enum stateTypeEnum
{
    CONTINUOUS = 0,
    DISCRETE = 1
};
```

In the C model code, these enum values can be accessed by using the follow C enum definitions:

```
/// AccelerometerstateTypeEnum enum
enum AccelerometerstateTypeEnum
{
    ACCELEROMETER_STATE_TYPE_ENUM_CONTINUOUS = 0,
    ACCELEROMETER_STATE_TYPE_ENUM_DISCRETE = 1
};
```

#### Parameters



Name	Type	Size	Quantity	Units	Description
arw	double				acceleration random walk constant [m/s <sup>2/3</sup> (3/2)]
axisBody	double	3			accelerometer axis in body frame
biasStblty	double				acceleration bias (in)stability constant [m/s <sup>2</sup> ]
constAccelBias	double		Acceleration		acceleration bias constant [m/s <sup>2</sup> ]
misAlign	double		Angle		accelerometer axis misalignment relative to body frame [rad]
scaleFactor	double				scale factor error
seed	long				PRN seed
stateType	stateTypeEnum				accelerometer state type
velocityCntIntType	cntIntType				accelerometer velocity integer type
velocityLSB	double				accelerometer velocity least significant bit [m/s/cnt]
vrw	double				velocity random walk constant [m/s/s <sup>1/2</sup> ]
vwn	double				velocity white noise constant [m/s/Hz <sup>1/2</sup> ]

#### Scratch

Name	Type	Size	Quantity	Units	Description
velocity_bias	double		Velocity		accelerometer velocity bias

#### Discrete States

Name	Type	Size	Quantity	Units	Description
arw	double				acceleration random walk state
biasStblty	double				acceleration bias (in)stability state
velocity	double		Velocity		accumulated accelerometer velocity
vrw	double				velocity random walk state

#### Continuous States

Name	Type	Size	Quantity	Units	Description
arw	double				acceleration random walk state
biasStblty	double				acceleration bias (in)stability state
velocity	double		Velocity		accumulated accelerometer velocity
vrw	double				velocity random walk state

#### Flow Outs

Name	Type	Size	Quantity	Units	Description
accel	double				measured accelerometer acceleration
deltaVelocity	double				measured accelerometer delta velocity
deltaVelocityCnts	long				measured accelerometer delta velocity counts
velocity	double		Velocity		measured accelerometer velocity
velocityCnts	long				measured accelerometer velocity counts

## 28.2. DshellCommonModels::BallEncoder Encoder Model



This model documentation is auto-generated. Documentation updates should be made to the model's `mdl` file to avoid the changes being lost.

[Doxygen link to all DshellCommonModels models of Encoder class](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/DshellCommonModels/html/group__Encoder_group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/DshellCommonModels/html/group\_\_Encoder\_group.html)

### Keywords Doxygen groups

- [Hinge](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/DshellCommonModels/html/group__Hinge_keyword_group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/DshellCommonModels/html/group\_\_Hinge\_keyword\_group.html)
- [Hinge!Ball](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/DshellCommonModels/html/group__Hinge_Ball_keyword_group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/DshellCommonModels/html/group\_\_Hinge\_Ball\_keyword\_group.html)

### Description

Outputs the ball rotation displacements

Author: Carl Merry <br> The BallEncoder reports the angular displacements a ball hinge has undergone in both quaternion and Euler form.

from the DshellCommonModels models library for the xref:Dshellpp\_module simulation framework.

### Model BallEncoder class details

For more information on the members and functions of this model class, please see [DshellCommonModels::BallEncoder model class doxygen documentation](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/DshellCommonModels/html/group__BallEncoder_group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/DshellCommonModels/html/group\_\_BallEncoder\_group.html)

### Enums

#### SOAEulerType Enum

In the BallEncoder model definition, the original enum SOAEulerType is defined as:

```
enum SOAEulerType
{
    BAD = 0,
    ZYX = 1,
    ZXY = 2,
    YZX = 3,
    YXZ = 4,
    XZY = 5,
    XYZ = 6,
    ZXZ = 7,
    ZYZ = 8,
    YXY = 9,
    YZY = 10,
    XYX = 11,
    XZX = 12
};
```

In the C model code, these enum values can be accessed by using the follow C enum definitions:

```

/// BallEncoderSOAEulerType enum
enum BallEncoderSOAEulerType
{
    BALL_ENCODER_SOAEULER_TYPE_BAD = 0,
    BALL_ENCODER_SOAEULER_TYPE_ZYX = 1,
    BALL_ENCODER_SOAEULER_TYPE_ZXY = 2,
    BALL_ENCODER_SOAEULER_TYPE_YZX = 3,
    BALL_ENCODER_SOAEULER_TYPE_YXZ = 4,
    BALL_ENCODER_SOAEULER_TYPE_XZY = 5,
    BALL_ENCODER_SOAEULER_TYPE_XYZ = 6,
    BALL_ENCODER_SOAEULER_TYPE_ZXZ = 7,
    BALL_ENCODER_SOAEULER_TYPE_ZYZ = 8,
    BALL_ENCODER_SOAEULER_TYPE_YXY = 9,
    BALL_ENCODER_SOAEULER_TYPE_YZY = 10,
    BALL_ENCODER_SOAEULER_TYPE_XYX = 11,
    BALL_ENCODER_SOAEULER_TYPE_XZX = 12
};

```

#### Parameters

Name	Type	Size	Quantity	Units	Description
EulerSequence	SOAEulerType				The angle sequence used in determining euler angles for the attitude.

#### Flow Outs

Name	Type	Size	Quantity	Units	Description
angle1	double	1	Angle		ball hinge first axis rotation
angle2	double	1	Angle		ball hinge second axis rotation
angle3	double	1	Angle		ball hinge third axis rotation
rotation	double	4	Quaternion		ball hinge rotation

### 28.3. DshellCommonModels::BallMasking Actuator Model



This model documentation is auto-generated. Documentation updates should be made to the model's `mdl` file to avoid the changes being lost.

[Doxygen link to all DshellCommonModels models of Actuator class](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/DshellCommonModels/html/group__Actuator__group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/DshellCommonModels/html/group\_\_Actuator\_\_group.html)

#### Description

Mask a BALL subhinge

from the DshellCommonModels models library for the xref:Dshellpp\_module simulation framework.

#### Model BallMasking class details

For more information on the members and functions of this model class, please see [DshellCommonModels::BallMasking model class doxygen documentation](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/DshellCommonModels/html/group__BallMasking__group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/DshellCommonModels/html/group\_\_BallMasking\_\_group.html)

#### Parameters

Name	Type	Size	Quantity	Units	Description
maskingActive	bool				Flag: True sets masking to Q

#### Flow Ins

Name	Type	Size	Quantity	Units	Description
genAccelMask	double	3	AngularAcceleration		UDOT masking data
genCoordMask	double	4	Quaternion		Q masking data
genVelMask	double	3	AngularVelocity		U masking data

## 28.4. DshellCommonModels::CmFrameStateSensor Sensor Model



This model documentation is auto-generated. Documentation updates should be made to the model's `mdl` file to avoid the changes being lost.

[Doxygen link to all DshellCommonModels models of Sensor class](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/DshellCommonModels/html/group__Sensor__group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/DshellCommonModels/html/group\_\_Sensor\_\_group.html)

### Keywords Doxygen groups

- [Attitude](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/DshellCommonModels/html/group__Attitude__keyword__group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/DshellCommonModels/html/group\_\_Attitude\_\_keyword\_\_group.html)
- [Idealized](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/DshellCommonModels/html/group__Idealized__keyword__group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/DshellCommonModels/html/group\_\_Idealized\_\_keyword\_\_group.html)
- [Position](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/DshellCommonModels/html/group__Position__keyword__group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/DshellCommonModels/html/group\_\_Position\_\_keyword\_\_group.html)

### Description

Maintains the CM based frame origin/attitude and outputs the orientation of a vehicle relative to the frame.

Author: Scott Nemeth <br> This sensor model uses node and frame objects to calculate the orientation of a body relative to the desired frame.

from the DshellCommonModels models library for the xref:Dshellpp\_module simulation framework.

### Model CmFrameStateSensor class details

For more information on the members and functions of this model class, please see [DshellCommonModels::CmFrameStateSensor model class doxygen documentation](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/DshellCommonModels/html/group__CmFrameStateSensor__group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/DshellCommonModels/html/group\_\_CmFrameStateSensor\_\_group.html)

### Enums

#### SOAEulerType Enum

In the CmFrameStateSensor model definition, the original enum SOAEulerType is defined as:

```
enum SOAEulerType
{
    _NOT_USED_ = 0,
    ZYX = 1,
    ZXY = 2,
    YZX = 3,
    YXZ = 4,
    XZY = 5,
    XYZ = 6,
    ZXZ = 7,
    ZYZ = 8,
    YXY = 9,
    YZY = 10,
    XYX = 11,
    XZX = 12
};
```

In the C model code, these enum values can be accessed by using the follow C enum definitions:

```
/// CmFrameStateSensorSOAEulerType enum
enum CmFrameStateSensorSOAEulerType
{
    CM_FRAME_STATE_SENSOR_SOAEULER_TYPE__NOT_USED_ = 0,
    CM_FRAME_STATE_SENSOR_SOAEULER_TYPE_ZYX = 1,
    CM_FRAME_STATE_SENSOR_SOAEULER_TYPE_ZXY = 2,
    CM_FRAME_STATE_SENSOR_SOAEULER_TYPE_YZX = 3,
    CM_FRAME_STATE_SENSOR_SOAEULER_TYPE_YXZ = 4,
    CM_FRAME_STATE_SENSOR_SOAEULER_TYPE_XZY = 5,
    CM_FRAME_STATE_SENSOR_SOAEULER_TYPE_XYZ = 6,
    CM_FRAME_STATE_SENSOR_SOAEULER_TYPE_ZXZ = 7,
    CM_FRAME_STATE_SENSOR_SOAEULER_TYPE_ZYZ = 8,
    CM_FRAME_STATE_SENSOR_SOAEULER_TYPE_YXY = 9,
    CM_FRAME_STATE_SENSOR_SOAEULER_TYPE_YZY = 10,
    CM_FRAME_STATE_SENSOR_SOAEULER_TYPE_XYX = 11,
    CM_FRAME_STATE_SENSOR_SOAEULER_TYPE_XZX = 12
};
```

#### CenterOfMassType Enum

In the CmFrameStateSensor model definition, the original enum CenterOfMassType is defined as:

```
enum CenterOfMassType
{
    FIXED = 0,
    COMPONENT = 1,
    COMPOSITE = 2
};
```

In the C model code, these enum values can be accessed by using the follow C enum definitions:

```
/// CmFrameStateSensorCenterOfMassType enum
enum CmFrameStateSensorCenterOfMassType
{
    CM_FRAME_STATE_SENSOR_CENTER_OF_MASS_TYPE_FIXED = 0,
    CM_FRAME_STATE_SENSOR_CENTER_OF_MASS_TYPE_COMPONENT = 1,
    CM_FRAME_STATE_SENSOR_CENTER_OF_MASS_TYPE_COMPOSITE = 2
};
```

### AngleModuloType Enum

In the CmFrameStateSensor model definition, the original enum AngleModuloType is defined as:

```
enum AngleModuloType
{
    ZERO_TO_TWO_PI = 0,
    MINUS_PI_TO_PI = 1
};
```

In the C model code, these enum values can be accessed by using the follow C enum definitions:

```
/// CmFrameStateSensorAngleModuloType enum
enum CmFrameStateSensorAngleModuloType
{
    CM_FRAME_STATE_SENSOR_ANGLE_MODULO_TYPE_ZERO_TO_TWO_PI = 0,
    CM_FRAME_STATE_SENSOR_ANGLE_MODULO_TYPE_MINUS_PI_TO_PI = 1
};
```

### VelocityReferenceType Enum

In the CmFrameStateSensor model definition, the original enum VelocityReferenceType is defined as:

```
enum VelocityReferenceType
{
    INERTIAL = 0,
    RELATIVE = 1
};
```

In the C model code, these enum values can be accessed by using the follow C enum definitions:

```
/// CmFrameStateSensorVelocityReferenceType enum
enum CmFrameStateSensorVelocityReferenceType
{
    CM_FRAME_STATE_SENSOR_VELOCITY_REFERENCE_TYPE_INERTIAL = 0,
    CM_FRAME_STATE_SENSOR_VELOCITY_REFERENCE_TYPE_RELATIVE = 1
};
```

### FrameType Enum

In the CmFrameStateSensor model definition, the original enum FrameType is defined as:

```
enum FrameType
{
    NONE = 0,
    VELOCITY_ALIGN = 1,
    LVLH = 2,
    UVW = 3,
    CW = 4
};
```

In the C model code, these enum values can be accessed by using the follow C enum definitions:

```
/// CmFrameStateSensorFrameType enum
enum CmFrameStateSensorFrameType
{
    CM_FRAME_STATE_SENSOR_FRAME_TYPE_NONE = 0,
    CM_FRAME_STATE_SENSOR_FRAME_TYPE_VELOCITY_ALIGN = 1,
    CM_FRAME_STATE_SENSOR_FRAME_TYPE_LVLH = 2,
    CM_FRAME_STATE_SENSOR_FRAME_TYPE_UVW = 3,
    CM_FRAME_STATE_SENSOR_FRAME_TYPE_CW = 4
};
```

### Parameters

Name	Type	Size	Quantity	Units	Description
CM_FRAME_UUID	int				<b>Not for user input</b> - UUID for the Center of Mass based frame.

Name	Type	Size	Quantity	Units	Description
PCI_FRAME_UUID	int				<b>Not for user input</b> - UUID for the planet centered inertial frame.
PCR_FRAME_UUID	int				<b>Not for user input</b> - UUID for the planet centered relative frame.
angleModulo	AngleModuloType				Specifies the modulo at which each angle will be output.
centerOfMassType	CenterOfMassType				Specifies whether to use the component body or composite body for CM location.
eulerSequence	SOAEulerType				The angle sequence used in determining euler angles for the attitude.
frameType	FrameType				The type of the frame that this model will create to measure the frame attitude state.
velocityReference	VelocityReferenceType				Specifies whether to inertial or relative velocity in creating the CM based frame.

#### Flow Outs

Name	Type	Size	Quantity	Units	Description
angles	double	3	Angle		Euler angles extracted using desired sequence.
quat	double	4	Quaternion		Body orientation relative to input reference frame.

## 28.5. DshellCommonModels::GimbalAngleEncoder Encoder Model



This model documentation is auto-generated. Documentation updates should be made to the model's `mdl` file to avoid the changes being lost.

[Doxygen link to all DshellCommonModels models of Encoder class](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/DshellCommonModels/html/group__Encoder_group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/DshellCommonModels/html/group\_\_Encoder\_group.html)

#### Keywords Doxygen groups

- [Hinge](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/DshellCommonModels/html/group__Hinge_keyword_group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/DshellCommonModels/html/group\_\_Hinge\_keyword\_group.html)
- [Hinge!Gimbal](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/DshellCommonModels/html/group__Hinge_Gimbal_keyword_group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/DshellCommonModels/html/group\_\_Hinge\_Gimbal\_keyword\_group.html)

#### Description

Outputs the gimbal angular displacements

Author: David Henriquez <br> / Garrett Sohl (converted to flowIns/Outs) The GimbalEncoder encoder reports the angular displacements a gimbal hinge has undergone (i.e. angle[3]). If GimbalEncoder is attached to a hinge with less than three degrees of freedom, the extra degree(s) of freedom will be set to zero. If GimbalEncoder is attached to a hinge with more than three degrees of freedom, only the first three will be reported; however, such a scenario may cause a system error (i.e. segmentation fault).

from the DshellCommonModels models library for the xref:Dshellpp\_module simulation framework.

#### Model GimbalAngleEncoder class details

For more information on the members and functions of this model class, please see [DshellCommonModels::GimbalAngleEncoder model class doxygen documentation](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/DshellCommonModels/html/group__GimbalAngleEncoder_group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/DshellCommonModels/html/group\_\_GimbalAngleEncoder\_group.html)

#### Flow Outs

Name	Type	Size	Quantity	Units	Description
rotation	double	3	Angle		gimbal hinge angle values
xangle	double	1	Angle		gimbal hinge angle values
yangle	double	1	Angle		gimbal hinge angle values
zangle	double	1	Angle		gimbal hinge angle values

## 28.6. DshellCommonModels::NodePosVelAccelSensor Sensor Model



This model documentation is auto-generated. Documentation updates should be made to the model's `mdl` file to avoid the changes being lost.

[Doxygen link to all DshellCommonModels models of Sensor class](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/DshellCommonModels/html/group__Sensor__group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/DshellCommonModels/html/group\_\_Sensor\_\_group.html)

### Keywords Doxygen groups

- [Accelerometer](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/DshellCommonModels/html/group__Accelerometer__keyword__group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/DshellCommonModels/html/group\_\_Accelerometer\_\_keyword\_\_group.html)
- [Gyroscope](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/DshellCommonModels/html/group__Gyroscope__keyword__group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/DshellCommonModels/html/group\_\_Gyroscope\_\_keyword\_\_group.html)
- [IMU](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/DshellCommonModels/html/group__IMU__keyword__group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/DshellCommonModels/html/group\_\_IMU\_\_keyword\_\_group.html)
- [Inertial](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/DshellCommonModels/html/group__Inertial__keyword__group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/DshellCommonModels/html/group\_\_Inertial\_\_keyword\_\_group.html)

### Description

Model of a IMU truth sensor.

Returns linear and angular rates and accelerations. Linear values are given in the inertial frame, while angular values are given in the local/node frame. Partly based on RoverDynModels/RoverTruthIMU

from the DshellCommonModels models library for the xref:Dshellpp\_module simulation framework.

### Model NodePosVelAccelSensor class details

For more information on the members and functions of this model class, please see [DshellCommonModels::NodePosVelAccelSensor model class doxygen documentation](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/DshellCommonModels/html/group__NodePosVelAccelSensor__group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/DshellCommonModels/html/group\_\_NodePosVelAccelSensor\_\_group.html)

### Scratch

Name	Type	Size	Quantity	Units	Description
accelMag	double		Acceleration		Linear acceleration magnitude (with respect to the inertial frame, m/sec**2)
bodySensedAccelMag	double		Acceleration		Linear acceleration magnitude in body axis (with respect to the inertial frame, m/sec**2)
bodyVelMag	double		Velocity		Linear inertial velocity magnitude (with respect to the body frame, m/sec)
posMag	double		Length		Position magnitude relative to inertial frame (m)
quat	double	4	Quaternion		Node attitude quaternion
velMag	double		Velocity		Linear inertial velocity magnitude (with respect to the inertial frame, m/sec)

### Flow Outs

Name	Type	Size	Quantity	Units	Description
accel	double	3	Acceleration		Linear accelerations (with respect to the inertial frame, m/sec**2)

Name	Type	Size	Quantity	Units	Description
alpha	double	3	AngularAcceleration		Angular accel (with respect to the inertial frame, rad/sec**2)
bodyOmega	double	3	AngularVelocity		Angular rates (with respect to the body frame, rad/sec)
bodySensedAccel	double	3	Acceleration		Linear accelerations in body axis (with respect to the inertial frame, m/sec**2)
bodyVel	double	3	Velocity		Linear inertial velocity (with respect to the body frame, m/sec)
omega	double	3	AngularVelocity		Angular rates (with respect to the inertial frame, rad/sec)
pos	double	3	Length		Position relative to inertial frame (m)
quat	double	4	Quaternion		Attitude quaternion (with respect to the inertial frame)
vel	double	3	Velocity		Linear inertial velocity (with respect to the inertial frame, m/sec)

## 28.7. DshellCommonModels::NoisyIMU Sensor Model



This model documentation is auto-generated. Documentation updates should be made to the model's `mdl` file to avoid the changes being lost.

[Doxygen link to all DshellCommonModels models of Sensor class](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/DshellCommonModels/html/group__Sensor__group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/DshellCommonModels/html/group\_\_Sensor\_\_group.html)

### Keywords Doxygen groups

- [Accelerometer](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/DshellCommonModels/html/group__Accelerometer__keyword__group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/DshellCommonModels/html/group\_\_Accelerometer\_\_keyword\_\_group.html)
- [Gyroscope](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/DshellCommonModels/html/group__Gyroscope__keyword__group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/DshellCommonModels/html/group\_\_Gyroscope\_\_keyword\_\_group.html)
- [IMU](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/DshellCommonModels/html/group__IMU__keyword__group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/DshellCommonModels/html/group\_\_IMU\_\_keyword\_\_group.html)
- [Inertial](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/DshellCommonModels/html/group__Inertial__keyword__group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/DshellCommonModels/html/group\_\_Inertial\_\_keyword\_\_group.html)
- [Random](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/DshellCommonModels/html/group__Random__keyword__group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/DshellCommonModels/html/group\_\_Random\_\_keyword\_\_group.html)
- [noise](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/DshellCommonModels/html/group__noise__keyword__group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/DshellCommonModels/html/group\_\_noise\_\_keyword\_\_group.html)

### Description

Model of a Noisy IMU (gyro+accel) with noise.

Model of a Noisy IMU with noise (based on EDLR4AeroModels/EdIIMU). The noisy output is computed as follows:  $y = (1+sf) * x + b + w$  where  $y$ : measured value  $sf$ : scale factor  $x$ : true value  $b$ : bias  $w$ : noise

from the DshellCommonModels models library for the xref:Dshellpp\_module simulation framework.

### Model NoisyIMU class details

For more information on the members and functions of this model class, please see [DshellCommonModels::NoisyIMU model class doxygen documentation](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/DshellCommonModels/html/group__NoisyIMU__group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/DshellCommonModels/html/group\_\_NoisyIMU\_\_group.html)

### Parameters

Name	Type	Size	Quantity	Units	Description
Accel_derivNoise	double	3	Dimensionless		Accel random walk noise (3-axis), NOT USED



Name	Type	Size	Quantity	Units	Description
Accel_initBias	double	3	Acceleration		Initial bias error for accel (3-axis)
Accel_nodriftPole	double	3	Dimensionless		no drift pole, NOT USED
Accel_quantization	double		Velocity		quatization factor for accel delta-v counts, NOT USED
Accel_randomWalkFactor	double	3	Dimensionless		Random Walk factor inputed by user, NOT USED
Accel_rateNoise	double	3	Dimensionless		Accel rate noise (3-axis)
Accel_scaleFactor	double	3	Dimensionless		scale factor error for accel (0=no scale factor error); measured_value = (1+scale_factor)*real_value + bias + noise
Gyro_derivNoise	double	3	Dimensionless		Gyro random walk noise (3-axis), NOT USED
Gyro_initBias	double	3	AngularVelocity		Initial bias error for gyro (3-axis)
Gyro_nodriftPole	double	3	Dimensionless		no drift pole, NOT USED
Gyro_quantization	double		Angle		quatization factor for gyro counts, NOT USED
Gyro_randomWalkFactor	double	3	Dimensionless		Random Walk factor inputed by user, NOT USED
Gyro_rateNoise	double	3	Dimensionless		Gyro rate noise (3-axis)
Gyro_scaleFactor	double	3	Dimensionless		scale factor error for gyro (0=no scale factor error); measured_value = (1+scale_factor)*real_value + bias + noise
seed	int				Seed #1 for uniform pseudo-random number generator

#### Scratch

Name	Type	Size	Quantity	Units	Description
prior_IO_dv_at_IMU	double	3	Velocity		dv at IMU in last IO step
prior_IO_theta	double	3	Angle		Theta at IMU in last IO step

#### Continuous States

Name	Type	Size	Quantity	Units	Description
dvAtIMU	double	3	Velocity		Integral of accelerometer
noiseAccel	double	3	Velocity		Noise Term for Accel
noiseOmega	double	3	Angle		Noise Term for Omega
thetaReal	double	3	Angle		Integral of angular rate

#### Flow Outs

Name	Type	Size	Quantity	Units	Description
accumulated_dv_at_IMU_IO	double	3	Velocity		Delta-V (linear) in node frame with noise and without gravity accel

Name	Type	Size	Quantity	Units	Description
dv_at_IMU_IO	double	3	Velocity		Delta-V (linear) in node frame with noise and without gravity accel; delta in IO step
theta	double	3	Angle		Gyro angles with noise
theta_IO	double	3	Angle		Delta-Theta for each IO step with noise

## 28.8. DshellCommonModels::NoisyNodePosVelAccelSensor Sensor Model



This model documentation is auto-generated. Documentation updates should be made to the model's `mdl` file to avoid the changes being lost.

[Doxygen link to all DshellCommonModels models of Sensor class](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/DshellCommonModels/html/group__Sensor_group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/DshellCommonModels/html/group\_\_Sensor\_group.html)

### Keywords Doxygen groups

- [Accelerometer](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/DshellCommonModels/html/group__Accelerometer_keyword_group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/DshellCommonModels/html/group\_\_Accelerometer\_keyword\_group.html)
- [Gyroscope](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/DshellCommonModels/html/group__Gyroscope_keyword_group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/DshellCommonModels/html/group\_\_Gyroscope\_keyword\_group.html)
- [IMU](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/DshellCommonModels/html/group__IMU_keyword_group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/DshellCommonModels/html/group\_\_IMU\_keyword\_group.html)
- [Inertial](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/DshellCommonModels/html/group__Inertial_keyword_group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/DshellCommonModels/html/group\_\_Inertial\_keyword\_group.html)
- [Random](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/DshellCommonModels/html/group__Random_keyword_group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/DshellCommonModels/html/group\_\_Random\_keyword\_group.html)
- [noise](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/DshellCommonModels/html/group__noise_keyword_group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/DshellCommonModels/html/group\_\_noise\_keyword\_group.html)

### Description

Adds noise to the node's attitude and body sensed accel.

Based on GeneralModels/NoisyAttitude. This model gets the node's attitude quaternion and multiplies it with an error quaternion. The error quaternion noise added to its four elements and is then normalized to guarantee that the norm of error quaternion is equal to unity. The model parameters determine the noise characteristics of the error quaternion. `noise_switch` is a flag to turn the noise on or off; 0 = no noise; 1 = noise included. If noise is turned off, the output of the model is the true attitude of the node. If the noise is turned on, the model initializes its error quaternion in `setup0`. The initial error quaternion adds uniformly distributed noise (i.e.  $\text{sig\_init\_error} * [-1, 1]$ ) to each element of the identity quaternion. `seed[2]` are the random number seeds used to generate ONLY for generating the initial error quaternion. For every I/O step in the simulation, the error quaternion has uniformly distributed noise (i.e.  $\text{sig\_att\_noise} * [-1, 1]$ ) added to each of its elements. Also a random walk term is added (i.e.  $\text{sig\_gyro\_drift} * \text{dt} * [-1, 1] + \text{gyro\_random\_walk} * \sqrt{\text{dt}} * [-1, 1]$ ) to each of the error quaternion elements. The model states keep the current value of the random number seeds (i.e. `seed_state[2]`) and next value for the random walk term (i.e. `drift_rand_state[4]`) for the error quaternion.

from the DshellCommonModels models library for the xref:Dshellpp\_module simulation framework.

### Model NoisyNodePosVelAccelSensor class details

For more information on the members and functions of this model class, please see [DshellCommonModels::NoisyNodePosVelAccelSensor model class doxygen documentation](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/DshellCommonModels/html/group__NoisyNodePosVelAccelSensor_group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/DshellCommonModels/html/group\_\_NoisyNodePosVelAccelSensor\_group.html)

### Parameters

Name	Type	Size	Quantity	Units	Description
gyro_random_walk	double		Dimensionless		gyro random walk
noise_switch	int				switch for noise; 0 = no noise; 1 = noise included
seed	double	2	Dimensionless		random number seed value
sig_att_noise	double		Angle		attitude noise
sig_gyro_drift	double		AngularVelocity		gyro drift
sig_init_error	double		Angle		initial attitude error

### Scratch

Name	Type	Size	Quantity	Units	Description
------	------	------	----------	-------	-------------

Name	Type	Size	Quantity	Units	Description
bodySensedAccel_truth	double	3	Acceleration	m/s**2	Truth linear accelerations in body axis (with respect to the inertial frame, m/sec**2)
quat_noise_level	double	4	Quaternion		Noise level of attitude quaternion
quat_truth	double	4	Quaternion		Truth attitude quaternion

#### Discrete States

Name	Type	Size	Quantity	Units	Description
drift_rand_state	double	4	Dimensionless		
seed_state	double	2	Dimensionless		

## 28.9. DshellCommonModels::RateGyro Sensor Model



This model documentation is auto-generated. Documentation updates should be made to the model's `mdl` file to avoid the changes being lost.

[Doxygen link to all DshellCommonModels models of Sensor class](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/DshellCommonModels/html/group__Sensor__group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/DshellCommonModels/html/group\_\_Sensor\_\_group.html)

#### Description

models rate gyro measurements

from the DshellCommonModels models library for the xref:Dshellpp\_module simulation framework.

#### Model RateGyro class details

For more information on the members and functions of this model class, please see [DshellCommonModels::RateGyro model class doxygen documentation](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/DshellCommonModels/html/group__RateGyro__group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/DshellCommonModels/html/group\_\_RateGyro\_\_group.html)

#### Enums

##### cntIntType Enum

In the RateGyro model definition, the original enum cntIntType is defined as:

```
enum cntIntType
{
    8BIT = 0,
    16BIT = 1,
    32BIT = 2
};
```

In the C model code, these enum values can be accessed by using the follow C enum definitions:

```
/// RateGyrocntIntType enum
enum RateGyrocntIntType
{
    RATE_GYRO_CNT_INT_TYPE_8BIT = 0,
    RATE_GYRO_CNT_INT_TYPE_16BIT = 1,
    RATE_GYRO_CNT_INT_TYPE_32BIT = 2
};
```

##### stateTypeEnum Enum

In the RateGyro model definition, the original enum stateTypeEnum is defined as:

```
enum stateTypeEnum
{
    CONTINUOUS = 0,
    DISCRETE = 1
};
```

In the C model code, these enum values can be accessed by using the follow C enum definitions:

```

/// RateGyrostateTypeEnum enum
enum RateGyrostateTypeEnum
{
    RATE_GYRO_STATE_TYPE_ENUM_CONTINUOUS = 0,
    RATE_GYRO_STATE_TYPE_ENUM_DISCRETE = 1
};

```

### Parameters

Name	Type	Size	Quantity	Units	Description
angleCntIntType	cntIntType				gyro angle integer type
angleLSB	double				gyro angle least significant bit [rad/cnt]
arw	double				angle random walk constant [rad/sec^(1/2)]
awn	double				angle white noise constant [rad/Hz^(1/2)]
axisBody	double	3			gyro axis in body frame
biasStblty	double				rate bias (in)stability constant [rad/sec]
constRateBias	double		AngularVelocity		rate bias constant [rad/sec]
driftRate	double				linear drift rate constant [rad/sec^2]
misAlign	double		Angle		gyro axis misalignment relative to body frame [rad]
rrw	double				rate random walk constant [rad/sec^(3/2)]
scaleFactor	double				scale factor error
seed	long				PRN seed
stateType	stateTypeEnum				gyro state type

### Scratch

Name	Type	Size	Quantity	Units	Description
angle_bias	double		Angle		gyro angle bias

### Discrete States

Name	Type	Size	Quantity	Units	Description
angle	double		Angle		accumulated gyro angle
arw	double				angle random walk state
biasStblty	double				rate bias (in)stability state
driftRate	double				linear drift rate
rrw	double				rate random walk state

### Continuous States

Name	Type	Size	Quantity	Units	Description
angle	double		Angle		accumulated gyro angle
arw	double				angle random walk state
biasStblty	double				rate bias (in)stability state
driftRate	double				linear drift rate
rrw	double				rate random walk state

## Flow Outs

Name	Type	Size	Quantity	Units	Description
angle	double		Angle		measured gyro angle
angleCnts	long				measured gyro angle counts
deltaAngle	double				measured gyro delta angle
deltaAngleCnts	long				measured gyro delta angle counts
rate	double				measured gyro rate

## 28.10. DshellCommonModels::ReactionWheel Motor Model



This model documentation is auto-generated. Documentation updates should be made to the model's `mdl` file to avoid the changes being lost.

[Doxygen link to all DshellCommonModels models of Motor class](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/DshellCommonModels/html/group__Motor_group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/DshellCommonModels/html/group\_\_Motor\_group.html)

### Description

from the DshellCommonModels models library for the xref:Dshellpp\_module simulation framework.

#### Model ReactionWheel class details

For more information on the members and functions of this model class, please see [DshellCommonModels::ReactionWheel model class doxygen documentation](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/DshellCommonModels/html/group__ReactionWheel_group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/DshellCommonModels/html/group\_\_ReactionWheel\_group.html)

#### Enums

##### FrictionModel Enum

In the ReactionWheel model definition, the original enum FrictionModel is defined as:

```
enum FrictionModel
{
    NONE = 0,
    LUGRE = 1,
    DAHL = 2,
    COULOMB_VISCOUS = 3
};
```

In the C model code, these enum values can be accessed by using the follow C enum definitions:

```
/// ReactionWheelFrictionModel enum
enum ReactionWheelFrictionModel
{
    REACTION_WHEEL_FRICTION_MODEL_NONE = 0,
    REACTION_WHEEL_FRICTION_MODEL_LUGRE = 1,
    REACTION_WHEEL_FRICTION_MODEL_DAHLE = 2,
    REACTION_WHEEL_FRICTION_MODEL_COULOMB_VISCOUS = 3
};
```

##### PowerState Enum

In the ReactionWheel model definition, the original enum PowerState is defined as:

```
enum PowerState
{
    OFF = 0,
    ON = 1
};
```

In the C model code, these enum values can be accessed by using the follow C enum definitions:

```
/// ReactionWheelPowerState enum
enum ReactionWheelPowerState
{
    REACTION_WHEEL_POWER_STATE_OFF = 0,
    REACTION_WHEEL_POWER_STATE_ON = 1
};
```

#### Parameters

Name	Type	Size	Quantity	Units	Description
k_coulomb	double		Torque		coulomb torque
k_dahl_exp	double		Dimensionless		dahl exponent
k_dahl_stiff	double		RotationalSpringStiffness		dahl stiffness coefficient
k_disable_spd	double		AngularVelocity		wheel speed disable threshold
k_enable_spd	double		AngularVelocity		wheel speed enable threshold
k_friction_model	FrictionModel				NONE=0, LUGRE=1, DAHL=2, COULOMB_VISCOUS=3
k_lugre_damp	double				lugre damping coefficient
k_lugre_stiff	double		RotationalSpringStiffness		lugre stiffness coefficient
k_mtr_trq_constant	double				motor torque constant
k_static	double		Torque		static torque
k_stiction_coeff	double				stiction coefficient
k_stribeck_exp	double		Dimensionless		stribeck exponent
k_stribeck_vel	double		AngularVelocity		stribeck velocity
k_tach_pulses_per_rev	int				number of pulses per wheel revolution
k_tach_read_period	double		Time		tachometer read period
k_viscous	double				viscous friction coefficient

#### Discrete States

Name	Type	Size	Quantity	Units	Description
overspeed	bool				wheel overspeed condition
power_state	PowerState				wheel power state

#### Continuous States

Name	Type	Size	Quantity	Units	Description
z	double		Angle		mean bristle deflection

#### Flow Ins

Name	Type	Size	Quantity	Units	Description
mtr_current	double		Current		motor current
power_cmd	bool				wheel power command

#### Flow Outs

Name	Type	Size	Quantity	Units	Description
trq_output	double		Torque		torque output
wheel_tach	double		AngularVelocity		tachometer wheel speed

## 28.11. DshellCommonModels::SpiceFramePCIBodySync Actuator Model



This model documentation is auto-generated. Documentation updates should be made to the model's `mdl` file to avoid the changes being lost.

[Doxygen link to all DshellCommonModels models of Actuator class](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/DshellCommonModels/html/group__Actuator__group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/DshellCommonModels/html/group\_\_Actuator\_\_group.html)

#### Description

Keeps a body in sync with an inertial Spice frame.

from the DshellCommonModels models library for the xref:Dshellpp\_module simulation framework.

#### Model SpiceFramePCIBodySync class details

For more information on the members and functions of this model class, please see [DshellCommonModels::SpiceFramePCIBodySync model class doxygen documentation](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/DshellCommonModels/html/group__SpiceFramePCIBodySync__group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/DshellCommonModels/html/group\_\_SpiceFramePCIBodySync\_\_group.html)

#### Parameters

Name	Type	Size	Quantity	Units	Description
PCI_SPICE_FRAME_UUID	int				<b>Not for user input</b> - UUID for the planet centered inertial frame.
SPICE_ROOT_FRAME_UUID	int				<b>Not for user input</b> - UUID for the Darts root inertial frame.
spiceETSimEpoch	double		Time		Spice Frame Container time corresponding to sim time=0.0.

#### Scratch

Name	Type	Size	Quantity	Units	Description
pciBodyOmega	double	3	AngularVelocity	rad/s	Rotational velocity of the PCI body with respect to the Darts root frame.
pciBodyPosition	double	3	Length	m	Position of the PCI body with respect to the Darts root frame.
pciBodyQuaternion	double	4	Quaternion	unitless	Orientation of the PCI body with respect to the Darts root frame.
pciBodyVelocity	double	3	Velocity	m/s	Velocity of the PCI body with respect to the Darts root frame.

#### Flow Outs

Name	Type	Size	Quantity	Units	Description
spiceTime	double		Time		Time according to the Spice Frame Container. This signal is used to correctly order the Spice models.

## 28.12. DshellCommonModels::SpiceFramePCIBodySync Actuator Model



This model documentation is auto-generated. Documentation updates should be made to the model's `mdl` file to avoid the changes being lost.

[Doxygen link to all DshellCommonModels models of Actuator class](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/DshellCommonModels/html/group__Actuator__group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/DshellCommonModels/html/group\_\_Actuator\_\_group.html)

#### Description

Keeps a body in sync with a rotating Spice frame.

Keeps a body in sync with a rotating Spice frame.

from the DshellCommonModels models library for the xref:Dshellpp\_module simulation framework.

#### Model SpiceFramePCRBodySync class details

For more information on the members and functions of this model class, please see [DshellCommonModels::SpiceFramePCRBodySync model class doxygen documentation](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/DshellCommonModels/html/group__SpiceFramePCRBodySync__group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/DshellCommonModels/html/group\_\_SpiceFramePCRBodySync\_\_group.html)

#### Parameters

Name	Type	Size	Quantity	Units	Description
PCI_SPICE_FRAME_UUID	int				<b>Not for user input</b> - UUID for the planet centered inertial frame.
PCR_SPICE_FRAME_UUID	int				<b>Not for user input</b> - UUID for the planet centered rotating frame.
SPICE_ROOT_FRAME_UUID	int				<b>Not for user input</b> - UUID for the Darts root inertial frame.

#### Scratch

Name	Type	Size	Quantity	Units	Description
pcrBodyOmega	double	3	AngularVelocity	rad/s	Rotational velocity of the PCR body with respect to the PCI body.
pcrBodyQuaternion	double	4	Quaternion	unitless	Orientation of the PCR body with respect to the PCI body.

#### Flow Ins

Name	Type	Size	Quantity	Units	Description
spiceTime	double		Time		Time according to the Spice Frame Container. This signal is used to correctly order the Spice models.

## 28.13. DshellCommonModels::UserClock Sensor Model



This model documentation is auto-generated. Documentation updates should be made to the model's `mdl` file to avoid the changes being lost.

[Doxygen link to all DshellCommonModels models of Sensor class](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/DshellCommonModels/html/group__Sensor__group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/DshellCommonModels/html/group\_\_Sensor\_\_group.html)

#### Keywords Doxygen groups

- [MET](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/DshellCommonModels/html/group__MET__keyword__group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/DshellCommonModels/html/group\_\_MET\_\_keyword\_\_group.html)
- [clock](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/DshellCommonModels/html/group__clock__keyword__group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/DshellCommonModels/html/group\_\_clock\_\_keyword\_\_group.html)
- [time](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/DshellCommonModels/html/group__time__keyword__group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/DshellCommonModels/html/group\_\_time\_\_keyword\_\_group.html)
- [timer](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/DshellCommonModels/html/group__timer__keyword__group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/DshellCommonModels/html/group\_\_timer\_\_keyword\_\_group.html)

#### Description

userClock sensor for providing vehicle/site-specific times to models

userClock sensor for providing vehicle/site-specific times to models

from the DshellCommonModels models library for the xref:Dshellpp\_module simulation framework.

#### Model UserClock class details



For more information on the members and functions of this model class, please see [DshellCommonModels::UserClock model class doxygen documentation](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/DshellCommonModels/html/group__UserClock_group.html) ([https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/DshellCommonModels/html/group\\_\\_UserClock\\_group.html](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/DshellCommonModels/html/group__UserClock_group.html))

#### Parameters

Name	Type	Size	Quantity	Units	Description
baseTime	int	2			Value of clock at sim time = 0.0

#### Flow Outs

Name	Type	Size	Quantity	Units	Description
clockTime	int	2			Value of userClock at current sim time in seconds and nanoseconds
time	double		Time		Value of userClock at current sim time as a double

## 29. GeneralSGModels Dshell model library

### Doxygen reference to Models in GeneralSGModels by Type

- [Actuators](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/GeneralSGModels/html/group__Actuator_group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/GeneralSGModels/html/group\_\_Actuator\_group.html)
- [Sensors](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/GeneralSGModels/html/group__Sensor_group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/GeneralSGModels/html/group\_\_Sensor\_group.html)
- [Motors](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/GeneralSGModels/html/group__Motor_group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/GeneralSGModels/html/group\_\_Motor\_group.html)
- [Encoders](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/GeneralSGModels/html/group__Encoder_group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/GeneralSGModels/html/group\_\_Encoder\_group.html)
- [Flows](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/GeneralSGModels/html/group__Flow_group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/GeneralSGModels/html/group\_\_Flow\_group.html)
- [Continuous](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/GeneralSGModels/html/group__ContStates_group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/GeneralSGModels/html/group\_\_ContStates\_group.html)

### 29.1. GeneralSGModels::BallJointSpringDamper Motor Model



This model documentation is auto-generated. Documentation updates should be made to the model's `mdl` file to avoid the changes being lost.

[Doxygen link to all GeneralSGModels models of Motor class](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/GeneralSGModels/html/group__Motor_group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/GeneralSGModels/html/group\_\_Motor\_group.html)

#### Keywords Doxygen groups

- [Hinge](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/GeneralSGModels/html/group__Hinge_keyword_group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/GeneralSGModels/html/group\_\_Hinge\_keyword\_group.html)
- [HingeBall](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/GeneralSGModels/html/group__Hinge_Ball_keyword_group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/GeneralSGModels/html/group\_\_Hinge\_Ball\_keyword\_group.html)

#### Description

Spring Damper for ball joints

This model computes and applies generalized forces representing spring dampers that resist rotation of a ball joint.

from the GeneralSGModels models library for the xref:Dshellpp\_module simulation framework.

#### Model BallJointSpringDamper class details

For more information on the members and functions of this model class, please see [GeneralSGModels::BallJointSpringDamper model class doxygen documentation](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/GeneralSGModels/html/group__BallJointSpringDamper_group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/GeneralSGModels/html/group\_\_BallJointSpringDamper\_group.html)

#### Parameters

Name	Type	Size	Quantity	Units	Description
C_x	double		RotationalSpringDamping		damping constant for rotations about the x axis
C_y	double		RotationalSpringDamping		damping constant for rotations about the y axis
C_z	double		RotationalSpringDamping		damping constant for rotations about the z axis
K_x	double		RotationalSpringStiffness		spring force constant for rotations about the x axis
K_y	double		RotationalSpringStiffness		spring force constant for rotations about the y axis
K_z	double		RotationalSpringStiffness		spring force constant for rotations about the z axis

#### Scratch

Name	Type	Size	Quantity	Units	Description
gen_coord	double	4	Quaternion		Hinge coordinates (quaternion)
gen_f	double	3	Torque		Generalized force (torque) on hinge

Name	Type	Size	Quantity	Units	Description
gen_vel	double	3	AngularVelocity		Hinge rotational velocity (rad/sec)
pitch	double		Angle		Pitch of hinge (rotation about the attachment node Y axis)
roll	double		Angle		Roll of hinge (rotation about the attachment node X axis)
yaw	double		Angle		Yaw of hinge (rotation about the attachment node Z axis)

## 29.2. GeneralSGModels::BearingAngle Sensor Model



This model documentation is auto-generated. Documentation updates should be made to the model's `mdl` file to avoid the changes being lost.

[Doxygen link to all GeneralSGModels models of Sensor class](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/GeneralSGModels/html/group__Sensor__group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/GeneralSGModels/html/group\_\_Sensor\_\_group.html)

### Keywords Doxygen groups

- [Celestial](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/GeneralSGModels/html/group__Celestial__keyword__group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/GeneralSGModels/html/group\_\_Celestial\_\_keyword\_\_group.html)
- [Relative](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/GeneralSGModels/html/group__Relative__keyword__group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/GeneralSGModels/html/group\_\_Relative\_\_keyword\_\_group.html)
- [angle](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/GeneralSGModels/html/group__angle__keyword__group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/GeneralSGModels/html/group\_\_angle\_\_keyword\_\_group.html)
- [vector](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/GeneralSGModels/html/group__vector__keyword__group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/GeneralSGModels/html/group\_\_vector\_\_keyword\_\_group.html)

### Description

Outputs angle between a node-fixed vector and an inertial vector

Author: David Henriquez <br> / Garrett Sohl (converted to flowIn/Outs) The BearingAngle sensor model takes two parameter vectors and reports the angle between the two vectors at each I/O step. The angle will have a value between  $[0, 2\pi]$ . The BearingAngle model parameters consist of two vectors. One of the parameter vectors is an inertial vector called `celest_vec[3]`. The other parameter vector is a vector fixed to the node called `body_vec[3]`. As the inertial node attitude changes due to spacecraft motion or body articulation, the inertial orientation of node-fixed vector also changes. The sole output of the BearingAngle model is the angle between the `celest_vec[3]` and `body_vec[3]`.

from the GeneralSGModels models library for the xref:Dshellpp\_module simulation framework.

### Model BearingAngle class details

For more information on the members and functions of this model class, please see [GeneralSGModels::BearingAngle model class doxygen documentation](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/GeneralSGModels/html/group__BearingAngle__group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/GeneralSGModels/html/group\_\_BearingAngle\_\_group.html)

### Parameters

Name	Type	Size	Quantity	Units	Description
body_vec	double	3	Length		body fixed vector
celest_vec	double	3	Length		vector to celestial body in inertial frame

### Scratch

Name	Type	Size	Quantity	Units	Description
angle	double		Angle		angle between the body and the celestial vector

### Flow Outs

Name	Type	Size	Quantity	Units	Description
angle	double		Angle		bearing angle to celestial body

## 29.3. GeneralSGModels::DCMotorVin Motor Model



This model documentation is auto-generated. Documentation updates should be made to the model's `mdl` file to avoid the changes being lost.

[Doxygen link to all GeneralSGModels models of Motor class](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/GeneralSGModels/html/group__Motor__group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/GeneralSGModels/html/group\_\_Motor\_\_group.html)

### Description

Model of a DC motor with voltage input.

Model an idealized DC motor with voltage control. The Motor current dynamics are assumed to be instantaneous (inductance is assumed to be zero). Input to this model is a voltage across the motor. Outputs of this model are motor torque and current draw. Motor braking is not yet implemented by this model.

from the GeneralSGModels models library for the xref:Dshellpp\_module simulation framework.

### Model DCMotorVin class details

For more information on the members and functions of this model class, please see [GeneralSGModels::DCMotorVin model class doxygen documentation](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/GeneralSGModels/html/group__DCMotorVin__group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/GeneralSGModels/html/group\_\_DCMotorVin\_\_group.html)

### Parameters

Name	Type	Size	Quantity	Units	Description
K_t	double		MotorTorqueConstant		Motor torque constant (N-m/Amp) = back emf constant (V-sec/rad)
R	double		Resistance		Motor resistance
damping	double		RotationalSpringDamping		Mechanical damping coefficient (N-m-s/rad)

### Scratch

Name	Type	Size	Quantity	Units	Description
gearRatio	double		Dimensionless		Motor gear ratio
inbOmega	double		AngularVelocity		Angular velocity of motor input shaft
outOmega	double		AngularVelocity		Angular velocity of motor output shaft (after gear ratio)

### Discrete States

Name	Type	Size	Quantity	Units	Description
brake	int				flag to turn brake on(1) or off(0)

### Flow Ins

Name	Type	Size	Quantity	Units	Description
voltage	double		Voltage		Motor voltage

### Flow Outs

Name	Type	Size	Quantity	Units	Description
current	double		Current		Motor current draw
power	double		Power		Motor power draw
torque	double		Torque		Motor torque

## 29.4. GeneralSGModels::DCMotorVin2 Motor Model



This model documentation is auto-generated. Documentation updates should be made to the model's `mdl` file to avoid the changes being lost.

[Doxygen link to all GeneralSGModels models of Motor class](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/GeneralSGModels/html/group__Motor__group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/GeneralSGModels/html/group\_\_Motor\_\_group.html)

### Description

Model of a DC motor with voltage input.

Model an idealized DC motor with voltage control, damping, and resistance. The Motor current dynamics are assumed to be instantaneous (inductance is assumed to be zero). Input to this model is a voltage across the motor. Outputs of this model are motor torque and current draw. Motor braking is not yet implemented by this model. REFERENCE: Mechatronics and Measurement Systems by Michael B. Hstand and David G. Alciatore, McGraw-Hill, 1999.

from the GeneralSGModels models library for the xref:Dshellpp\_module simulation framework.

### Model DCMotorVin2 class details

For more information on the members and functions of this model class, please see [GeneralSGModels::DCMotorVin2 model class doxygen documentation](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/GeneralSGModels/html/group__DCMotorVin2__group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/GeneralSGModels/html/group\_\_DCMotorVin2\_\_group.html)

### Parameters

Name	Type	Size	Quantity	Units	Description
K_e	double		MotorBackEmfConstant		Voltage/speed constant (V-sec/rad)
stall_current	double		Current		Stall current (amps)

## 29.5. GeneralSGModels::ExternalSingleDofDisturbance Actuator Model



This model documentation is auto-generated. Documentation updates should be made to the model's `mdl` file to avoid the changes being lost.

[Doxygen link to all GeneralSGModels models of Actuator class](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/GeneralSGModels/html/group__Actuator__group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/GeneralSGModels/html/group\_\_Actuator\_\_group.html)

### Keywords Doxygen groups

- [Disturbance](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/GeneralSGModels/html/group__Disturbance__keyword__group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/GeneralSGModels/html/group\_\_Disturbance\_\_keyword\_\_group.html)
- [model](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/GeneralSGModels/html/group__model__keyword__group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/GeneralSGModels/html/group\_\_model\_\_keyword\_\_group.html)

### Description

Actuates a single degree of freedom on a node

Author: David Henriquez <br> / Garrett Sohl (modified to use flowIns) The ExternalSingleDofDisturbance actuator model is similar to the ExternalDisturbance actuator model. However, it also applies a pure force or pure torque about only one axis. The ExternalSingleDofDisturbance parameter Axis specifies the active axis and whether a force or torque is applied. The options for Axis are the following enumerated list: MX (torque about the local X axis), MY (torque about the local Y axis), MZ (torque about the local Z axis), FX (force about the local X axis), FY (force about the local Y axis), FZ (force about the local Y axis) and IGNORE; by default, the Axis parameter is set to IGNORE. The ExternalSingleDofDisturbance model flowIn specifies the magnitude of the force or torque. This model has no flowOuts.

from the GeneralSGModels models library for the xref:Dshellpp\_module simulation framework.

### Model ExternalSingleDofDisturbance class details

For more information on the members and functions of this model class, please see [GeneralSGModels::ExternalSingleDofDisturbance model class doxygen documentation](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/GeneralSGModels/html/group__ExternalSingleDofDisturbance__group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/GeneralSGModels/html/group\_\_ExternalSingleDofDisturbance\_\_group.html)

### Enums

#### Axis Enum

In the ExternalSingleDofDisturbance model definition, the original enum Axis is defined as:

```
enum Axis
{
    MX = 0,
    MY = 1,
    MZ = 2,
    FX = 3,
    FY = 4,
    FZ = 5,
    IGNORE = 6
};
```

In the C model code, these enum values can be accessed by using the follow C enum definitions:

```
/// ExternalSingleDofDisturbanceAxis enum
enum ExternalSingleDofDisturbanceAxis
{
    EXTERNAL_SINGLE_DOF_DISTURBANCE_AXIS_MX = 0,
    EXTERNAL_SINGLE_DOF_DISTURBANCE_AXIS_MY = 1,
    EXTERNAL_SINGLE_DOF_DISTURBANCE_AXIS_MZ = 2,
    EXTERNAL_SINGLE_DOF_DISTURBANCE_AXIS_FX = 3,
    EXTERNAL_SINGLE_DOF_DISTURBANCE_AXIS_FY = 4,
    EXTERNAL_SINGLE_DOF_DISTURBANCE_AXIS_FZ = 5,
    EXTERNAL_SINGLE_DOF_DISTURBANCE_AXIS_IGNORE = 6
};
```

#### Parameters

Name	Type	Size	Quantity	Units	Description
activeDof	Axis				

#### Flow Ins

Name	Type	Size	Quantity	Units	Description
dof	double		Unspecified		External single dof disturbance force or torque

## 29.6. GeneralSGModels::GearedPinAccel Motor Model



This model documentation is auto-generated. Documentation updates should be made to the model's `mdl` file to avoid the changes being lost.

[Doxygen link to all GeneralSGModels models of Motor class](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/GeneralSGModels/html/group__Motor__group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/GeneralSGModels/html/group\_\_Motor\_\_group.html)

#### Keywords Doxygen groups

- [Hinge](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/GeneralSGModels/html/group__Hinge__keyword__group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/GeneralSGModels/html/group\_\_Hinge\_\_keyword\_\_group.html)
- [Hinge!Pin](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/GeneralSGModels/html/group__Hinge__Pin__keyword__group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/GeneralSGModels/html/group\_\_Hinge\_\_Pin\_\_keyword\_\_group.html)
- [Hinge!Prescribed](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/GeneralSGModels/html/group__Hinge__Prescribed__keyword__group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/GeneralSGModels/html/group\_\_Hinge\_\_Prescribed\_\_keyword\_\_group.html)
- [Rover](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/GeneralSGModels/html/group__Rover__keyword__group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/GeneralSGModels/html/group\_\_Rover\_\_keyword\_\_group.html)
- [Rover!Controller](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/GeneralSGModels/html/group__Rover__Controller__keyword__group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/GeneralSGModels/html/group\_\_Rover\_\_Controller\_\_keyword\_\_group.html)
- [Rover!Locomotion](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/GeneralSGModels/html/group__Rover__Locomotion__keyword__group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/GeneralSGModels/html/group\_\_Rover\_\_Locomotion\_\_keyword\_\_group.html)
- [Rover!Wheel](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/GeneralSGModels/html/group__Rover__Wheel__keyword__group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/GeneralSGModels/html/group\_\_Rover\_\_Wheel\_\_keyword\_\_group.html)
- [Wheel](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/GeneralSGModels/html/group__Wheel__keyword__group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/GeneralSGModels/html/group\_\_Wheel\_\_keyword\_\_group.html)
- [Wheel!Controller](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/GeneralSGModels/html/group__Wheel__Controller__keyword__group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/GeneralSGModels/html/group\_\_Wheel\_\_Controller\_\_keyword\_\_group.html)
- [Wheel!Steering](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/GeneralSGModels/html/group__Wheel__Steering__keyword__group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/GeneralSGModels/html/group\_\_Wheel\_\_Steering\_\_keyword\_\_group.html)

#### Description

Prescribes accelerations to PIN joints with gear ratios. This motor MUST be on a body with a PRESCRIBED, single DOF joint.

from the GeneralSGModels models library for the xref:Dshellpp\_module simulation framework.

#### Model GearedPinAccel class details

For more information on the members and functions of this model class, please see [GeneralSGModels::GearedPinAccel model class doxygen documentation](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/GeneralSGModels/html/group__GearedPinAccel__group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/GeneralSGModels/html/group\_\_GearedPinAccel\_\_group.html)

## Scratch

Name	Type	Size	Quantity	Units	Description
gearRatio	double		Dimensionless		gear ratio

## Flow Ins

Name	Type	Size	Quantity	Units	Description
accel	double		AngularAcceleration		Prescribed acceleration

## 29.7. GeneralSGModels::GearedPinAngle Motor Model



This model documentation is auto-generated. Documentation updates should be made to the model's `mdl` file to avoid the changes being lost.

[Doxygen link to all GeneralSGModels models of Motor class](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/GeneralSGModels/html/group__Motor_group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/GeneralSGModels/html/group\_\_Motor\_group.html)

### Keywords Doxygen groups

- [Hinge](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/GeneralSGModels/html/group__Hinge_keyword_group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/GeneralSGModels/html/group\_\_Hinge\_keyword\_group.html)
- [Hinge!Pin](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/GeneralSGModels/html/group__Hinge_Pin_keyword_group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/GeneralSGModels/html/group\_\_Hinge\_Pin\_keyword\_group.html)
- [Hinge!Prescribed](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/GeneralSGModels/html/group__Hinge_Prescribed_keyword_group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/GeneralSGModels/html/group\_\_Hinge\_Prescribed\_keyword\_group.html)
- [Rover](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/GeneralSGModels/html/group__Rover_keyword_group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/GeneralSGModels/html/group\_\_Rover\_keyword\_group.html)
- [Rover!Controller](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/GeneralSGModels/html/group__Rover_Controller_keyword_group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/GeneralSGModels/html/group\_\_Rover\_Controller\_keyword\_group.html)
- [Rover!Locomotion](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/GeneralSGModels/html/group__Rover_Locomotion_keyword_group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/GeneralSGModels/html/group\_\_Rover\_Locomotion\_keyword\_group.html)
- [Rover!Wheel](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/GeneralSGModels/html/group__Rover_Wheel_keyword_group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/GeneralSGModels/html/group\_\_Rover\_Wheel\_keyword\_group.html)
- [Wheel](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/GeneralSGModels/html/group__Wheel_keyword_group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/GeneralSGModels/html/group\_\_Wheel\_keyword\_group.html)
- [Wheel!Controller](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/GeneralSGModels/html/group__Wheel_Controller_keyword_group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/GeneralSGModels/html/group\_\_Wheel\_Controller\_keyword\_group.html)
- [Wheel!Steering](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/GeneralSGModels/html/group__Wheel_Steering_keyword_group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/GeneralSGModels/html/group\_\_Wheel\_Steering\_keyword\_group.html)

### Description

Prescribes angles for PIN joints with gear ratios. This motor **MUST** be on a body with a PRESCRIBED, single DOF joint.

This model also assumes that the Q and U variables are MASKED, even though this is a prescribed hinge. The purpose of the masking is to force the subhinge's Q & U to exactly follow the model's Q and U values. So this model is meant to be used for dofs that do not have a controller, but we want the dof to follow a desired profile. The alternative is to add a high gain controller for the steering dof. Just removing the masking does not work because without a controller, residual velocities in the integrated U bleed into Q causing it to grow steadily.

from the GeneralSGModels models library for the xref:Dshellpp\_module simulation framework.

### Model GearedPinAngle class details

For more information on the members and functions of this model class, please see [GeneralSGModels::GearedPinAngle model class doxygen documentation](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/GeneralSGModels/html/group__GearedPinAngle_group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/GeneralSGModels/html/group\_\_GearedPinAngle\_group.html)

## Flow Ins

Name	Type	Size	Quantity	Units	Description
angle	double		Angle		Prescribed angle
rate	double		AngularVelocity		Prescribed rate

## 29.8. GeneralSGModels::GeneralAccelGimbal Motor Model



This model documentation is auto-generated. Documentation updates should be made to the model's `mdl` file to avoid the changes being lost.

[Doxygen link to all GeneralSGModels models of Motor class](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/GeneralSGModels/html/group__Motor_group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/GeneralSGModels/html/group\_\_Motor\_group.html)

### Keywords Doxygen groups

- [Hinge](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/GeneralSGModels/html/group__Hinge_keyword_group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/GeneralSGModels/html/group\_\_Hinge\_keyword\_group.html)

- [Hinge!Gimbal](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/GeneralSGModels/html/group__Hinge__Gimbal_keyword_group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/GeneralSGModels/html/group\_\_Hinge\_\_Gimbal\_keyword\_group.html)
- [Hinge!Prescribed](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/GeneralSGModels/html/group__Hinge__Prescribed_keyword_group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/GeneralSGModels/html/group\_\_Hinge\_\_Prescribed\_keyword\_group.html)

#### Description

Applies a generalized acceleration to a 3 DOF hinge

Author: David Henriquez <br> / Garrett Sohl (modified to use flowIns) The GeneralAccelGimbal motor model applies a generalized acceleration to the 3 DOF hinge on which the model is attached. The GeneralAccelGimbal model flowIn is used to specify the applied generalized acceleration. This model has no output.

from the GeneralSGModels models library for the xref:Dshellpp\_module simulation framework.

#### Model GeneralAccelGimbal class details

For more information on the members and functions of this model class, please see [GeneralSGModels::GeneralAccelGimbal model class doxygen documentation](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/GeneralSGModels/html/group__GeneralAccelGimbal_group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/GeneralSGModels/html/group\_\_GeneralAccelGimbal\_group.html)

#### Flow Ins

Name	Type	Size	Quantity	Units	Description
input	double	3	AngularAcceleration		prescribed gimbal hinge acceleration values

## 29.9. GeneralSGModels::GeneralAccelUjoint Motor Model



This model documentation is auto-generated. Documentation updates should be made to the model's `mdl` file to avoid the changes being lost.

[Doxygen link to all GeneralSGModels models of Motor class](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/GeneralSGModels/html/group__Motor_group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/GeneralSGModels/html/group\_\_Motor\_group.html)

#### Keywords Doxygen groups

- [Hinge](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/GeneralSGModels/html/group__Hinge_keyword_group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/GeneralSGModels/html/group\_\_Hinge\_keyword\_group.html)
- [Hinge!Prescribed](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/GeneralSGModels/html/group__Hinge__Prescribed_keyword_group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/GeneralSGModels/html/group\_\_Hinge\_\_Prescribed\_keyword\_group.html)
- [Hinge!Ujoint](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/GeneralSGModels/html/group__Hinge__Ujoint_keyword_group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/GeneralSGModels/html/group\_\_Hinge\_\_Ujoint\_keyword\_group.html)

#### Description

Applies a generalized acceleration to a 2 DOF hinge

Author: David Henriquez <br> / Garrett Sohl (modified to use flowIns) The GeneralAccelUjoint motor model applies a generalized acceleration to the 2 DOF hinge on which the model is attached. The GeneralAccelUjoint model flowIn is used to specify the applied generalized acceleration. This model has no output.

from the GeneralSGModels models library for the xref:Dshellpp\_module simulation framework.

#### Model GeneralAccelUjoint class details

For more information on the members and functions of this model class, please see [GeneralSGModels::GeneralAccelUjoint model class doxygen documentation](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/GeneralSGModels/html/group__GeneralAccelUjoint_group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/GeneralSGModels/html/group\_\_GeneralAccelUjoint\_group.html)

#### Flow Ins

Name	Type	Size	Quantity	Units	Description
input	double	2	AngularAcceleration		prescribed u-joint hinge acceleration values

## 29.10. GeneralSGModels::GeneralForceFulldofs Motor Model



This model documentation is auto-generated. Documentation updates should be made to the model's `mdl` file to avoid the changes being lost.

[Doxygen link to all GeneralSGModels models of Motor class](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/GeneralSGModels/html/group__Motor_group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/GeneralSGModels/html/group\_\_Motor\_group.html)

#### Keywords Doxygen groups

- [6dof](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/GeneralSGModels/html/group__6dof_keyword_group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/GeneralSGModels/html/group\_\_6dof\_keyword\_group.html)



- [Hinge](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/GeneralSGModels/html/group__Hinge_keyword_group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/GeneralSGModels/html/group\_\_Hinge\_keyword\_group.html)
- [Hinge!Full](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/GeneralSGModels/html/group__Hinge_Full_keyword_group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/GeneralSGModels/html/group\_\_Hinge\_Full\_keyword\_group.html)
- [Torque](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/GeneralSGModels/html/group__Torque_keyword_group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/GeneralSGModels/html/group\_\_Torque\_keyword\_group.html)
- [Torque!Commanded](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/GeneralSGModels/html/group__Torque_Commanded_keyword_group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/GeneralSGModels/html/group\_\_Torque\_Commanded\_keyword\_group.html)

#### Description

Applies a generalized force to a 6 DOF hinge

Author: David Henriquez <br> / Garrett Sohl (modified to use flowIns) The GeneralForceFullDOFs motor model applies a generalized force to the 6 DOF hinge on which the model is attached. The GeneralForceFullDOFs model flowIn is used to specify the applied generalized force. This model has no output.

from the GeneralSGModels models library for the xref:Dshellpp\_module simulation framework.

#### Model GeneralForceFullDOFs class details

For more information on the members and functions of this model class, please see [GeneralSGModels::GeneralForceFullDOFs model class doxygen documentation](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/GeneralSGModels/html/group__GeneralForceFullDOFs_group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/GeneralSGModels/html/group\_\_GeneralForceFullDOFs\_group.html)

#### Flow Ins

Name	Type	Size	Quantity	Units	Description
input	double	6	Mixed		generalized force value

## 29.11. GeneralSGModels::GeneralForceGimbal Motor Model



This model documentation is auto-generated. Documentation updates should be made to the model's `mdl` file to avoid the changes being lost.

[Doxygen link to all GeneralSGModels models of Motor class](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/GeneralSGModels/html/group__Motor_group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/GeneralSGModels/html/group\_\_Motor\_group.html)

#### Keywords Doxygen groups

- [Hinge](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/GeneralSGModels/html/group__Hinge_keyword_group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/GeneralSGModels/html/group\_\_Hinge\_keyword\_group.html)
- [Hinge!Gimbal](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/GeneralSGModels/html/group__Hinge_Gimbal_keyword_group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/GeneralSGModels/html/group\_\_Hinge\_Gimbal\_keyword\_group.html)
- [Torque](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/GeneralSGModels/html/group__Torque_keyword_group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/GeneralSGModels/html/group\_\_Torque\_keyword\_group.html)
- [Torque!Commanded](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/GeneralSGModels/html/group__Torque_Commanded_keyword_group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/GeneralSGModels/html/group\_\_Torque\_Commanded\_keyword\_group.html)

#### Description

Applies a generalized force to a 3 DOF hinge

Author: David Henriquez <br> / Garrett Sohl (modified to use flowIns) The GeneralForceGimbal motor model applies a generalized force to the 3 DOF hinge on which the model is attached. The GeneralForceGimbal model flowIn is used to specify the applied generalized force. This model has no output.

from the GeneralSGModels models library for the xref:Dshellpp\_module simulation framework.

#### Model GeneralForceGimbal class details

For more information on the members and functions of this model class, please see [GeneralSGModels::GeneralForceGimbal model class doxygen documentation](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/GeneralSGModels/html/group__GeneralForceGimbal_group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/GeneralSGModels/html/group\_\_GeneralForceGimbal\_group.html)

#### Flow Ins

Name	Type	Size	Quantity	Units	Description
input	double	3	Torque		gimbal hinge generalized force value

## 29.12. GeneralSGModels::GeneralForceUjoint Motor Model



This model documentation is auto-generated. Documentation updates should be made to the model's `mdl` file to avoid the changes being lost.

[Doxygen link to all GeneralSGModels models of Motor class](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/GeneralSGModels/html/group__Motor__group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/GeneralSGModels/html/group\_\_Motor\_\_group.html)

**Keywords Doxygen groups**

- [Hinge](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/GeneralSGModels/html/group__Hinge__keyword__group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/GeneralSGModels/html/group\_\_Hinge\_\_keyword\_\_group.html)
- [Hinge!Ujoint](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/GeneralSGModels/html/group__Hinge__Ujoint__keyword__group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/GeneralSGModels/html/group\_\_Hinge\_\_Ujoint\_\_keyword\_\_group.html)
- [Torque](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/GeneralSGModels/html/group__Torque__keyword__group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/GeneralSGModels/html/group\_\_Torque\_\_keyword\_\_group.html)
- [Torque!Commanded](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/GeneralSGModels/html/group__Torque__Commanded__keyword__group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/GeneralSGModels/html/group\_\_Torque\_\_Commanded\_\_keyword\_\_group.html)

**Description**

Applies a generalized force to a 2 DOF hinge

Author: David Henriquez <br> / Garrett Sohl (modified to use flowIns) The GeneralForceUjoint motor model applies a generalized force to the 2 DOF hinge on which the model is attached. The GeneralForceUjoint model flowIn is used to specify the applied generalized force. This model has no output.

from the GeneralSGModels models library for the xref:Dshellpp\_module simulation framework.

**Model GeneralForceUjoint class details**

For more information on the members and functions of this model class, please see [GeneralSGModels::GeneralForceUjoint model class doxygen documentation](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/GeneralSGModels/html/group__GeneralForceUjoint__group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/GeneralSGModels/html/group\_\_GeneralForceUjoint\_\_group.html)

**Flow Ins**

Name	Type	Size	Quantity	Units	Description
input	double	2	Torque		u-joint hinge generalized force value

### 29.13. GeneralSGModels::GeneralizedSpringDamperMotor Motor Model

 This model documentation is auto-generated. Documentation updates should be made to the model's mdl file to avoid the changes being lost.

[Doxygen link to all GeneralSGModels models of Motor class](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/GeneralSGModels/html/group__Motor__group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/GeneralSGModels/html/group\_\_Motor\_\_group.html)

**Keywords Doxygen groups**

- [Rover](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/GeneralSGModels/html/group__Rover__keyword__group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/GeneralSGModels/html/group\_\_Rover\_\_keyword\_\_group.html)
- [Rover!Dynamics](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/GeneralSGModels/html/group__Rover__Dynamics__keyword__group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/GeneralSGModels/html/group\_\_Rover\_\_Dynamics\_\_keyword\_\_group.html)
- [Spring/Damper](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/GeneralSGModels/html/group__Spring-Damper__keyword__group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/GeneralSGModels/html/group\_\_Spring-Damper\_\_keyword\_\_group.html)

**Description**

Generalized Spring Damper Motor Model for any of hinge

Given any hinge, compute generalized forces for the hinge dofs given the spring and damping coefficients for each of the dofs

from the GeneralSGModels models library for the xref:Dshellpp\_module simulation framework.

**Model GeneralizedSpringDamperMotor class details**

For more information on the members and functions of this model class, please see [GeneralSGModels::GeneralizedSpringDamperMotor model class doxygen documentation](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/GeneralSGModels/html/group__GeneralizedSpringDamperMotor__group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/GeneralSGModels/html/group\_\_GeneralizedSpringDamperMotor\_\_group.html)

**Parameters**

Name	Type	Size	Quantity	Units	Description
C	double	*	Unspecified		Damping term for a hinge dof (size of this value is number of hinge dofs)
K	double	*	Unspecified		Spring coefficient for a hinge dof (size of this value is number of hinge dofs)

Name	Type	Size	Quantity	Units	Description
zero_gen	double	*	Unspecified		Nominal zero value for a hinge dof (size of this value is number of hinge dofs)

#### Scratch

Name	Type	Size	Quantity	Units	Description
damping_f	double	*	Unspecified		Generalized damping forces
gen_coord	double	*	Unspecified		Hinge Coordinates
gen_f	double	*	Unspecified		Generalized hinge forces
gen_vel	double	*	Unspecified		Hinge velocities
spring_f	double	*	Unspecified		Generalized spring forces

## 29.14. GeneralSGModels::GimbalEncoder Encoder Model



This model documentation is auto-generated. Documentation updates should be made to the model's `mdl` file to avoid the changes being lost.

[Doxygen link to all GeneralSGModels models of Encoder class](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/GeneralSGModels/html/group__Encoder__group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/GeneralSGModels/html/group\_\_Encoder\_\_group.html)

#### Keywords Doxygen groups

- [Hinge](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/GeneralSGModels/html/group__Hinge__keyword__group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/GeneralSGModels/html/group\_\_Hinge\_\_keyword\_\_group.html)
- [Hinge!Gimbal](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/GeneralSGModels/html/group__Hinge__Gimbal__keyword__group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/GeneralSGModels/html/group\_\_Hinge\_\_Gimbal\_\_keyword\_\_group.html)

#### Description

Outputs the gimbal angular displacements

Author: David Henriquez <br> / Garrett Sohl (converted to flowIns/Outs) The GimbalEncoder encoder reports the angular displacements a gimbal hinge has undergone (i.e. angle[3]). If GimbalEncoder is attached to a hinge with less than three degrees of freedom, the extra degree(s) of freedom will be set to zero. If GimbalEncoder is attached to a hinge with more than three degrees of freedom, only the first three will be reported; however, such a scenario may cause a system error (i.e. segmentation fault).

from the GeneralSGModels models library for the xref:Dshellpp\_module simulation framework.

#### Model GimbalEncoder class details

For more information on the members and functions of this model class, please see [GeneralSGModels::GimbalEncoder model class doxygen documentation](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/GeneralSGModels/html/group__GimbalEncoder__group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/GeneralSGModels/html/group\_\_GimbalEncoder\_\_group.html)

#### Flow Outs

Name	Type	Size	Quantity	Units	Description
angle	double	3	Angle		gimbal hinge angle values

## 29.15. GeneralSGModels::JointForceTorqueSensor Sensor Model



This model documentation is auto-generated. Documentation updates should be made to the model's `mdl` file to avoid the changes being lost.

[Doxygen link to all GeneralSGModels models of Sensor class](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/GeneralSGModels/html/group__Sensor__group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/GeneralSGModels/html/group\_\_Sensor\_\_group.html)

#### Keywords Doxygen groups

- [Force](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/GeneralSGModels/html/group__Force__keyword__group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/GeneralSGModels/html/group\_\_Force\_\_keyword\_\_group.html)
- [Hinge](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/GeneralSGModels/html/group__Hinge__keyword__group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/GeneralSGModels/html/group\_\_Hinge\_\_keyword\_\_group.html)
- [Hinge!Force](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/GeneralSGModels/html/group__Hinge__Force__keyword__group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/GeneralSGModels/html/group\_\_Hinge\_\_Force\_\_keyword\_\_group.html)
- [Hinge!Torque](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/GeneralSGModels/html/group__Hinge__Torque__keyword__group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/GeneralSGModels/html/group\_\_Hinge\_\_Torque\_\_keyword\_\_group.html)
- [torque](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/GeneralSGModels/html/group__torque__keyword__group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/GeneralSGModels/html/group\_\_torque\_\_keyword\_\_group.html)

### Description

Outputs the forces and torques at the joint connecting the body to its parent body

Author: Jonathan M. Cameron This model senses the forces and torques at the joint connecting the body to its parent body. The forces and torques are normally in the 'p-node' (outboard) node frame of the joint. However, the user may specify a frame to transform the forces and torques into. WARNING: This will be the constraint forces at the joint projected into the 'p-node' frame. Therefore, this sensor will sense zero forces and torques if the joint is unconstrained. To sense the full forces and torques at the joint, ensure that it is a 'LOCKED' joint.

from the GeneralSGModels models library for the xref:Dshellpp\_module simulation framework.

### Model JointForceTorqueSensor class details

For more information on the members and functions of this model class, please see [GeneralSGModels::JointForceTorqueSensor model class doxygen documentation](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/GeneralSGModels/html/group__JointForceTorqueSensor__group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/GeneralSGModels/html/group\_\_JointForceTorqueSensor\_\_group.html)

### Enums

#### FTSenseMode Enum

In the JointForceTorqueSensor model definition, the original enum FTSenseMode is defined as:

```
enum FTSenseMode
{
    BODY = 0,
    PARENT = 1
};
```

In the C model code, these enum values can be accessed by using the follow C enum definitions:

```
/// JointForceTorqueSensorFTSenseMode enum
enum JointForceTorqueSensorFTSenseMode
{
    JOINT_FORCE_TORQUE_SENSOR_FTSENSE_MODE_BODY = 0,
    JOINT_FORCE_TORQUE_SENSOR_FTSENSE_MODE_PARENT = 1
};
```

### Parameters

Name	Type	Size	Quantity	Units	Description
referenceFrame	unsigned long (ulong)				UUID of Frame in which the input external disturbance force and torque are applied.
senseMode	FTSenseMode				Selects how the forces and torques are felt: by the body(BODY) or by the inboard body(PARENT)

### Scratch

Name	Type	Size	Quantity	Units	Description
force	double	3	Force		Joint force in the reference frame
torque	double	3	Torque		Joint force in the reference frame

### Flow Outs

Name	Type	Size	Quantity	Units	Description
force	double	3	Force		joint force
torque	double	3	Torque		joint torque



This model documentation is auto-generated. Documentation updates should be made to the model's `mdl` file to avoid the changes being lost.

[Doxygen link to all GeneralSGModels models of Sensor class](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/GeneralSGModels/html/group__Sensor__group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/GeneralSGModels/html/group\_\_Sensor\_\_group.html)

#### Keywords Doxygen groups

- [Idealized](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/GeneralSGModels/html/group__Idealized__keyword__group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/GeneralSGModels/html/group\_\_Idealized\_\_keyword\_\_group.html)
- [Node](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/GeneralSGModels/html/group__Node__keyword__group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/GeneralSGModels/html/group\_\_Node\_\_keyword\_\_group.html)
- [Node!Frame](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/GeneralSGModels/html/group__Node__Frame__keyword__group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/GeneralSGModels/html/group\_\_Node\_\_Frame\_\_keyword\_\_group.html)
- [Node!Uuid](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/GeneralSGModels/html/group__Node__Uuid__keyword__group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/GeneralSGModels/html/group\_\_Node\_\_Uuid\_\_keyword\_\_group.html)

#### Description

Outputs the UUID for the node this frame is attached to

The NodeFrame2FrameUuid sensor model outputs UUID for the root frame to the sensor node frame to frame. It is a simpler alternative to the NodePosAttitude model and does not put out the attitude/position information.

from the GeneralSGModels models library for the xref:Dshellpp\_module simulation framework.

#### Model NodeFrame2FrameUuid class details

For more information on the members and functions of this model class, please see [GeneralSGModels::NodeFrame2FrameUuid model class doxygen documentation](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/GeneralSGModels/html/group__NodeFrame2FrameUuid__group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/GeneralSGModels/html/group\_\_NodeFrame2FrameUuid\_\_group.html)

#### Parameters

Name	Type	Size	Quantity	Units	Description
rootFrame	unsigned long (ulong)				UUID for the root frame against which position and orientation are calculated

#### Scratch

Name	Type	Size	Quantity	Units	Description
pos	double	3	Length		The sensor node position wrt mbody root frame
quat	double	4	Quaternion		The sensor node quaternion wrt mbody root frame

#### Flow Outs

Name	Type	Size	Quantity	Units	Description
root2node_f2f_uuid	unsigned long (ulong)				The UUID for the mbody root to the sensor frame

## 29.17. GeneralSGModels::NoisyAttitude Sensor Model



This model documentation is auto-generated. Documentation updates should be made to the model's `mdl` file to avoid the changes being lost.

[Doxygen link to all GeneralSGModels models of Sensor class](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/GeneralSGModels/html/group__Sensor__group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/GeneralSGModels/html/group\_\_Sensor\_\_group.html)

#### Keywords Doxygen groups

- [Node](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/GeneralSGModels/html/group__Node__keyword__group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/GeneralSGModels/html/group\_\_Node\_\_keyword\_\_group.html)
- [Node!Attitude](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/GeneralSGModels/html/group__Node__Attitude__keyword__group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/GeneralSGModels/html/group\_\_Node\_\_Attitude\_\_keyword\_\_group.html)
- [Random](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/GeneralSGModels/html/group__Random__keyword__group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/GeneralSGModels/html/group\_\_Random\_\_keyword\_\_group.html)
- [noise](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/GeneralSGModels/html/group__noise__keyword__group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/GeneralSGModels/html/group\_\_noise\_\_keyword\_\_group.html)

#### Description

Adds noise a node's attitude

Author: David Henriquez <br> The NoisyAttitude sensor model gets the node's attitude quaternion and multiplies it with an error quaternion. The error quaternion noise added to its four elements and is then normalized to guarantee that the norm of error quaternion is equal to unity. The NoisyAttitude model parameters determine the noise characteristics of the error quaternion. `noise_switch` is a flag to turn the noise on or off; 0 = no noise; 1 = noise included. If noise is turned off, the output of the model is the true attitude of the node. If the noise is turned on, the model initializes its error quaternion in `setup()`. The initial error quaternion adds uniformly distributed noise (i.e.  $\text{sig\_init\_error} \cdot [-1, 1]$ ) to each element of the identity quaternion. `seed[2]` are the random number seeds used to generate ONLY for generating the initial error quaternion. For every I/O step in the simulation, the error quaternion has uniformly distributed noise (i.e.  $\text{sig\_att\_noise} \cdot [-1, 1]$ ) added to each of its elements. Also a random walk term is added (i.e.  $\text{sig\_gyro\_drift} \cdot dt \cdot [-1, 1] + \text{gyro\_random\_walk} \cdot \sqrt{dt} \cdot [-1, 1]$ ) to each of the error quaternion elements. The NoisyAttitude model states keep the current value of the random number seeds (i.e. `seed_state[2]`) and next value for the random walk term (i.e. `drift_rand_state[4]`) for the error quaternion. The NoisyAttitude model output is the error quaternion (i.e. `noise_level[4]`) and the noisy attitude (i.e. `noisy_attitude[4]`). If the noise is turned off, the error quaternion will be the identity quaternion and the noisy attitude will be the true attitude of the node.

from the GeneralSGModels models library for the xref:Dshellpp\_module simulation framework.

#### Model NoisyAttitude class details

For more information on the members and functions of this model class, please see [GeneralSGModels::NoisyAttitude model class doxygen documentation](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/GeneralSGModels/html/group__NoisyAttitude__group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/GeneralSGModels/html/group\_\_NoisyAttitude\_\_group.html)

#### Parameters

Name	Type	Size	Quantity	Units	Description
gyro_random_walk	double		Unspecified		gyro random walk
noise_switch	int				switch for noise; 0 = no noise; 1 = noise included
seed	double	2	Dimensionless		random number seed value
sig_att_noise	double		Angle		attitude noise
sig_gyro_drift	double		AngularVelocity		gyro drift
sig_init_error	double		Angle		initial attitude error

#### Scratch

Name	Type	Size	Quantity	Units	Description
firstFlag	int				triggers initialization behavior on the first io step

#### Discrete States

Name	Type	Size	Quantity	Units	Description
drift_rand_state	double	4	Quaternion		
error_out	double	4	Quaternion		The noise quaternion to be applied to true attitude quaternion
seed_state	double	2	Dimensionless		

#### Flow Outs

Name	Type	Size	Quantity	Units	Description
noise_level	double	4	Quaternion		
noisy_attitude	double	4	Quaternion		

## 29.18. GeneralSGModels::PinRate Encoder Model



This model documentation is auto-generated. Documentation updates should be made to the model's `mdl` file to avoid the changes being lost.

[Doxygen link to all GeneralSGModels models of Encoder class](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/GeneralSGModels/html/group__Encoder__group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/GeneralSGModels/html/group\_\_Encoder\_\_group.html)

**Keywords** *Doxygen groups*

- [Hinge](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/GeneralSGModels/html/group__Hinge__keyword__group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/GeneralSGModels/html/group\_\_Hinge\_\_keyword\_\_group.html)

- [Hinge!Angular](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/GeneralSGModels/html/group__Hinge_Angular_keyword_group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/GeneralSGModels/html/group\_\_Hinge\_Angular\_keyword\_group.html)
- [Hinge!Pin](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/GeneralSGModels/html/group__Hinge_Pin_keyword_group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/GeneralSGModels/html/group\_\_Hinge\_Pin\_keyword\_group.html)
- [velocity](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/GeneralSGModels/html/group__velocity_keyword_group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/GeneralSGModels/html/group\_\_velocity\_keyword\_group.html)

### Description

Outputs the pin joint angular velocity

Author: Garrett Sohl <br> The PinRate sensor reports the angular velocity of a pin joint hinge. If PinRate is attached to a hinge with more than one degree of freedom, only the first degree of freedom will be reported; however, such a scenario may cause a system error (i.e. segmentation fault).

from the GeneralSGModels models library for the xref:Dshellpp\_module simulation framework.

### Model PinRate class details

For more information on the members and functions of this model class, please see [GeneralSGModels::PinRate model class doxygen documentation](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/GeneralSGModels/html/group__PinRate_group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/GeneralSGModels/html/group\_\_PinRate\_group.html)

### Discrete States

Name	Type	Size	Quantity	Units	Description
rate	double		AngularVelocity		

### Flow Outs

Name	Type	Size	Quantity	Units	Description
rate	double		AngularVelocity		Pin rate

## 29.19. GeneralSGModels::PrescribedUjoint Motor Model



This model documentation is auto-generated. Documentation updates should be made to the model's `mdl` file to avoid the changes being lost.

[Doxygen link to all GeneralSGModels models of Motor class](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/GeneralSGModels/html/group__Motor_group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/GeneralSGModels/html/group\_\_Motor\_group.html)

### Keywords Doxygen groups

- [Hinge](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/GeneralSGModels/html/group__Hinge_keyword_group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/GeneralSGModels/html/group\_\_Hinge\_keyword\_group.html)
- [Hinge!Prescribed](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/GeneralSGModels/html/group__Hinge_Prescribed_keyword_group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/GeneralSGModels/html/group\_\_Hinge\_Prescribed\_keyword\_group.html)
- [Hinge!Ujoint](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/GeneralSGModels/html/group__Hinge_Ujoint_keyword_group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/GeneralSGModels/html/group\_\_Hinge\_Ujoint\_keyword\_group.html)

### Description

Set the displacement, velocity and acceleration of a u-joint hinge

Author: David Henriquez <br> The PrescribedUjoint motor model, upon receiving a command, sets the u-joint's hinge displacement, hinge velocity and hinge acceleration. NOTE: angle[0] goes to subhinge(0), angle[1] to subhinge(1), etc Refactored flowIns by Jonathan M. Cameron, November 2015.

from the GeneralSGModels models library for the xref:Dshellpp\_module simulation framework.

### Model PrescribedUjoint class details

For more information on the members and functions of this model class, please see [GeneralSGModels::PrescribedUjoint model class doxygen documentation](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/GeneralSGModels/html/group__PrescribedUjoint_group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/GeneralSGModels/html/group\_\_PrescribedUjoint\_group.html)

### Enums

#### MaskType Enum

In the PrescribedUjoint model definition, the original enum MaskType is defined as:

```
enum MaskType
{
    MASK_NONE = 0,
    MASK_Q = 1,
    MASK_U = 2,
    MASK_Q_U = 3
};
```

In the C model code, these enum values can be accessed by using the follow C enum definitions:

```

/// PrescribedUjointMaskType enum
enum PrescribedUjointMaskType
{
    PRESCRIBED_UJOINT_MASK_TYPE_MASK_NONE = 0,
    PRESCRIBED_UJOINT_MASK_TYPE_MASK_Q = 1,
    PRESCRIBED_UJOINT_MASK_TYPE_MASK_U = 2,
    PRESCRIBED_UJOINT_MASK_TYPE_MASK_Q_U = 3
};

```

#### Parameters

Name	Type	Size	Quantity	Units	Description
MaskMode	MaskType	2			Masking mode for the two subhinges

#### Flow Ins

Name	Type	Size	Quantity	Units	Description
angAccel	double	2	AngularAcceleration		prescribed hinge angular acceleraties (radians/sec**2)
angVel	double	2	AngularVelocity		prescribed hinge angular velocities (radians/sec)
angle	double	2	Angle		prescribed hinge angles (radians)

## 29.20. GeneralSGModels::SecondOrderResponse Sensor Model



This model documentation is auto-generated. Documentation updates should be made to the model's `mdl` file to avoid the changes being lost.

[Doxygen link to all GeneralSGModels models of Sensor class](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/GeneralSGModels/html/group__Sensor_group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/GeneralSGModels/html/group\_\_Sensor\_group.html)

#### Keywords Doxygen groups

- [Order](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/GeneralSGModels/html/group__Order_keyword_group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/GeneralSGModels/html/group\_\_Order\_keyword\_group.html)
- [Response](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/GeneralSGModels/html/group__Response_keyword_group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/GeneralSGModels/html/group\_\_Response\_keyword\_group.html)
- [Second](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/GeneralSGModels/html/group__Second_keyword_group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/GeneralSGModels/html/group\_\_Second\_keyword\_group.html)

### Description

from the GeneralSGModels models library for the xref:Dshellpp\_module simulation framework.

#### Model SecondOrderResponse class details

For more information on the members and functions of this model class, please see [GeneralSGModels::SecondOrderResponse model class doxygen documentation](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/GeneralSGModels/html/group__SecondOrderResponse_group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/GeneralSGModels/html/group\_\_SecondOrderResponse\_group.html)

#### Parameters

Name	Type	Size	Quantity	Units	Description
Fc	double		Frequency		Roll-off frequency (Hz)
InputMax	double		Unspecified		Maximum value of input (u)
InputMin	double		Unspecified		Minimum value of input (u)
State2ndDerivMax	double		Unspecified		Maximum value of state second derivative
State2ndDerivMin	double		Unspecified		Minimum value of state second derivative
StateDerivMax	double		Unspecified		Maximum value of state derivative
StateDerivMin	double		Unspecified		Minimum value of state derivative



Name	Type	Size	Quantity	Units	Description
StateMax	double		Unspecified		Maximum value of state
StateMin	double		Unspecified		Minimum value of state
damping	double		Dimensionless		Damping coefficient (0-1)

#### Scratch

Name	Type	Size	Quantity	Units	Description
Wn	double		AngularVelocity		Natural frequency (rad/sec)

#### Continuous States

Name	Type	Size	Quantity	Units	Description
x1	double		Unspecified		State
x2	double		Unspecified		State derivative

#### Flow Ins

Name	Type	Size	Quantity	Units	Description
u	double		Unspecified		Input command

#### Flow Outs

Name	Type	Size	Quantity	Units	Description
state	double		Unspecified		state
state2ndDeriv	double		Unspecified		state second derivative
stateDeriv	double		Unspecified		state derivative

## 29.21. GeneralSGModels::SignalMux Sensor Model



This model documentation is auto-generated. Documentation updates should be made to the model's `mdl` file to avoid the changes being lost.

[Doxygen link to all GeneralSGModels models of Sensor class](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/GeneralSGModels/html/group__Sensor_group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/GeneralSGModels/html/group\_\_Sensor\_group.html)

#### Description

Multiplexes output signal based on input signal and provided mux table.

Multiplexes output signal based on input signal and provided mux table. The provided variable sized parameter integer index table has to be the same size as the flow-output muxed signal. The flow-out is updated at PREDERIV granularity as  $flow\_out[i] = flow\_in[index[i]]$ . The indices within this table need to be in the range of the input to-be-muxed signal. The model can handle input and output signals of different variable sizes.

from the GeneralSGModels models library for the xref:Dshellpp\_module simulation framework.

#### Model SignalMux class details

For more information on the members and functions of this model class, please see [GeneralSGModels::SignalMux model class doxygen documentation](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/GeneralSGModels/html/group__SignalMux_group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/GeneralSGModels/html/group\_\_SignalMux\_group.html)

#### Parameters

Name	Type	Size	Quantity	Units	Description
muxTable	int	*			Index table

#### Flow Ins

Name	Type	Size	Quantity	Units	Description
------	------	------	----------	-------	-------------

Name	Type	Size	Quantity	Units	Description
in	double	*	Unspecified		Input signal

#### Flow Outs

Name	Type	Size	Quantity	Units	Description
out	double	*	Unspecified		Output signal

## 29.22. GeneralSGModels::SingleTrapezoidalProfile Motor Model



This model documentation is auto-generated. Documentation updates should be made to the model's `mdl` file to avoid the changes being lost.

[Doxygen link to all GeneralSGModels models of Motor class](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/GeneralSGModels/html/group__Motor__group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/GeneralSGModels/html/group\_\_Motor\_\_group.html)

#### Keywords Doxygen groups

- [Motion](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/GeneralSGModels/html/group__Motion__keyword__group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/GeneralSGModels/html/group\_\_Motion\_\_keyword\_\_group.html)
- [profile](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/GeneralSGModels/html/group__profile__keyword__group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/GeneralSGModels/html/group\_\_profile\_\_keyword\_\_group.html)

#### Description

Generates a trapezoidal velocity profile motion for a specified change in coordinates.

Parameters define the maximum velocity and acceleration for the wheel. This model generates a motor velocity profile consisting of a linear acceleration/deceleration phases plus a possible constant velocity (coast) phase. The input commands for this model consist of a triplet of maximum acceleration, maximum velocity and the desired change in coordinates. The model computes the velocity profile based on the input velocity/acceleration constraints and to meet the desired motion. This model can handle new commands that come in while in the midst of generating a profile - the profile is continued based on the new velocity/acceleration constraints and the desired angular motion. If the input max velocity is zero, then a brake to rest motion is triggered. The motion can consist of three phases: accel to coast, coast, and decel to rest phase. The first two phase may not be needed based on the commanded motion.

from the GeneralSGModels models library for the xref:Dshellpp\_module simulation framework.

#### Model SingleTrapezoidalProfile class details

For more information on the members and functions of this model class, please see [GeneralSGModels::SingleTrapezoidalProfile model class doxygen documentation](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/GeneralSGModels/html/group__SingleTrapezoidalProfile__group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/GeneralSGModels/html/group\_\_SingleTrapezoidalProfile\_\_group.html)

#### Parameters

Name	Type	Size	Quantity	Units	Description
relativeMode	bool				True means to use 2*PI wrap around for input coords to determine shortest motion.

#### Discrete States

Name	Type	Size	Quantity	Units	Description
currentStatus	int				current status (0→moving on profile, 1→done with profile)
end_coord	double		Unspecified		The absolute end coord value
prev_iostep_coord	double		Unspecified		The output coordinate value at the end of the previous I/O step
prev_iostep_vel	double		Unspecified		The output velocity value at the end of the previous I/O step
prev_profile_coord	double		Unspecified		The output coordinate value at the end of the last profile

## Flow Ins

Name	Type	Size	Quantity	Units	Description
accel	double		Unspecified		Absolute value of acceleration
coord	double		Unspecified		Desired coord change
maxVel	double		Unspecified		Absolute value of maximum velocity.

## Flow Outs

Name	Type	Size	Quantity	Units	Description
accel	double		Unspecified		The current acceleration value for the motion profile.
coord	double		Unspecified		The current absolute coordinate value for the motion profile.
rate	double		Unspecified		The current velocity value for the motion profile.
status	int				0→still moving, 1→done with profile

## 29.23. GeneralSGModels::SoftJointStop Motor Model



This model documentation is auto-generated. Documentation updates should be made to the model's `mdl` file to avoid the changes being lost.

[Doxygen link to all GeneralSGModels models of Motor class](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/GeneralSGModels/html/group__Motor_group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/GeneralSGModels/html/group\_\_Motor\_group.html)

### Keywords Doxygen groups

- [Constraint](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/GeneralSGModels/html/group__Constraint_keyword_group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/GeneralSGModels/html/group\_\_Constraint\_keyword\_group.html)
- [Constraint!Soft](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/GeneralSGModels/html/group__Constraint_Soft_keyword_group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/GeneralSGModels/html/group\_\_Constraint\_Soft\_keyword\_group.html)
- [Hinge](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/GeneralSGModels/html/group__Hinge_keyword_group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/GeneralSGModels/html/group\_\_Hinge\_keyword\_group.html)
- [Hinge!Constraint](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/GeneralSGModels/html/group__Hinge_Constraint_keyword_group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/GeneralSGModels/html/group\_\_Hinge\_Constraint\_keyword\_group.html)
- [Rover](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/GeneralSGModels/html/group__Rover_keyword_group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/GeneralSGModels/html/group\_\_Rover\_keyword\_group.html)
- [Rover!Dynamics](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/GeneralSGModels/html/group__Rover_Dynamics_keyword_group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/GeneralSGModels/html/group\_\_Rover\_Dynamics\_keyword\_group.html)
- [Spring/Damper](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/GeneralSGModels/html/group__Spring-Damper_keyword_group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/GeneralSGModels/html/group\_\_Spring-Damper\_keyword\_group.html)

### Description

Models a joint stop using a soft constraint.

A soft constraint penalty function is used to prevent motion past joint stops by applying an opposing torque. A nonlinear, Hunt-Crossley spring model is used.

from the GeneralSGModels models library for the xref:Dshellpp\_module simulation framework.

### Model SoftJointStop class details

For more information on the members and functions of this model class, please see [GeneralSGModels::SoftJointStop model class doxygen documentation](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/GeneralSGModels/html/group__SoftJointStop_group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/GeneralSGModels/html/group\_\_SoftJointStop\_group.html)

### Parameters

Name	Type	Size	Quantity	Units	Description
alpha	double		RotationalSpringDamping		Damping coefficient
k	double		RotationalSpringStiffness		Spring constant
lower_bound	double		Angle		Lower angle bound for joint

Name	Type	Size	Quantity	Units	Description
n	double		Dimensionless		Exponential for deflection term
upper_bound	double		Angle		Upper angle bound for joint

#### Scratch

Name	Type	Size	Quantity	Units	Description
angle	double		Angle		Hinge angle
delta	double		Angle		Deflection past joint stop
gearRatio	double		Dimensionless		Gear ratio of hinge
omega	double		AngularVelocity		Hinge angular velocity
outb_angle	double		Angle		Angle of outboard body
outb_omega	double		AngularVelocity		Angular velocity of outboard body
outb_tau	double		Torque		Constraint torque on outboard body
tau	double		Torque		Constraint torque on hinge

## 29.24. GeneralSGModels::SpringDamper Actuator Model



This model documentation is auto-generated. Documentation updates should be made to the model's `mdl` file to avoid the changes being lost.

[Doxygen link to all GeneralSGModels models of Actuator class](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/GeneralSGModels/html/group_Actuator_group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/GeneralSGModels/html/group\_Actuator\_group.html)

#### Keywords Doxygen groups

- [Constraint](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/GeneralSGModels/html/group_Constraint_keyword_group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/GeneralSGModels/html/group\_Constraint\_keyword\_group.html)
- [Constraint!Soft](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/GeneralSGModels/html/group_Constraint_Soft_keyword_group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/GeneralSGModels/html/group\_Constraint\_Soft\_keyword\_group.html)
- [Rover](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/GeneralSGModels/html/group_Rover_keyword_group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/GeneralSGModels/html/group\_Rover\_keyword\_group.html)
- [Rover!Dynamics](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/GeneralSGModels/html/group_Rover_Dynamics_keyword_group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/GeneralSGModels/html/group\_Rover\_Dynamics\_keyword\_group.html)
- [Spring/Damper](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/GeneralSGModels/html/group_Spring-Damper_keyword_group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/GeneralSGModels/html/group\_Spring-Damper\_keyword\_group.html)

#### Description

Simple spring-damper model

This model computes and applies constraint forces for a simple spring-damper that connects two actuator nodes on two different Darts bodies. The names of these nodes are input to the model with the `node1` and `node2` parameters. The connection force is always double-sided (tension and/or compression). This model has no `flowIns` or `flowOuts`. It makes direct Darts calls to compute the positions of each attachment point (`node1` and `node2`) and applies the constraint forces. This model is not ModelMex compliant.

from the GeneralSGModels models library for the `xref:Dshellpp_module` simulation framework.

#### Model SpringDamper class details

For more information on the members and functions of this model class, please see [GeneralSGModels::SpringDamper model class doxygen documentation](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/GeneralSGModels/html/group_SpringDamper_group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/GeneralSGModels/html/group\_SpringDamper\_group.html)

#### Parameters

Name	Type	Size	Quantity	Units	Description
C	double		LinearSpringDamping		damping constant
K	double		LinearSpringStiffness		spring force constant
from_node	object				frame of node on the first end of the spring-damper

Name	Type	Size	Quantity	Units	Description
to_node	object				frame of node on the second end of the spring-damper
unsprung_length	double		Length		Nominal unsprung length of spring

#### Scratch

Name	Type	Size	Quantity	Units	Description
dampner_force	double		Force		Magnitude of force exerted by the damper on node1
deflection	double		Length		Deflection of the spring length from nominal
force	double	3	Force		Total constraint force exerted by the spring-damper on node 1
force1_local	double	3	Force		Total constraint force exerted by the spring-damper on node 1
force2_local	double	3	Force		Total constraint force exerted by the spring-damper on node 1
length	double		Length		Length of the deflected spring
pos1	double	3	Length		Position of first node
pos2	double	3	Length		Position of second node
quat1	double	4	Quaternion		Quaternion of first node's local body coordinates
quat2	double	4	Quaternion		Quaternion of second node's local body coordinates
rel_velocity	double		Velocity		Relative speed of the end nodes
rvec	double	3	Length		Position vector from node1 to node2
rvec_unit	double	3	Dimensionless		Unit vector from node1 towards node2
spring_force	double		Force		Magnitude of force exerted by the spring on node1
vel1	double	3	Velocity		Velocity of first node
vel2	double	3	Velocity		Velocity of second node

## 29.25. GeneralSGModels::SpringDamperMotor Motor Model



This model documentation is auto-generated. Documentation updates should be made to the model's `mdl` file to avoid the changes being lost.

[Doxygen link to all GeneralSGModels models of Motor class](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/GeneralSGModels/html/group__Motor_group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/GeneralSGModels/html/group\_\_Motor\_group.html)

#### Keywords Doxygen groups

- [Rover](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/GeneralSGModels/html/group__Rover_keyword_group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/GeneralSGModels/html/group\_\_Rover\_keyword\_group.html)
- [Rover!Dynamics](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/GeneralSGModels/html/group__Rover_Dynamics_keyword_group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/GeneralSGModels/html/group\_\_Rover\_Dynamics\_keyword\_group.html)
- [Spring/Damper](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/GeneralSGModels/html/group__Spring-Damper_keyword_group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/GeneralSGModels/html/group\_\_Spring-Damper\_keyword\_group.html)

#### Description

Simple spring-damper model

This model computes and applies a generalized forces for a simple spring-damper that connects two different Darts bodies by a hinge. The connection force is always double-sided (tension and/or compression). This model has no flowIns or flowOuts. It makes direct Darts calls to compute the positions of each attachment point (node1 and node2) and applies the constraint forces. This model is not ModelMex compliant.

from the GeneralSGModels models library for the xref:Dshellpp\_module simulation framework.

#### Model SpringDamperMotor class details

For more information on the members and functions of this model class, please see [GeneralSGModels::SpringDamperMotor model class doxygen documentation](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/GeneralSGModels/html/group__SpringDamperMotor_group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/GeneralSGModels/html/group\_\_SpringDamperMotor\_group.html)

#### Parameters

Name	Type	Size	Quantity	Units	Description
C	double		Unspecified		damping constant
K	double		Unspecified		spring force constant
unsprung_length	double		Unspecified		Nominal unsprung length of spring

#### Scratch

Name	Type	Size	Quantity	Units	Description
gen_coord	double		Unspecified		Hinge coordinate
gen_f	double		Unspecified		Generalized force on hinge
gen_vel	double		Unspecified		Hinge velocity

## 29.26. GeneralSGModels::SpringDamperMotor6dof Motor Model



This model documentation is auto-generated. Documentation updates should be made to the model's `mdl` file to avoid the changes being lost.

[Doxygen link to all GeneralSGModels models of Motor class](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/GeneralSGModels/html/group__Motor_group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/GeneralSGModels/html/group\_\_Motor\_group.html)

#### Keywords Doxygen groups

- [Rover](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/GeneralSGModels/html/group__Rover_keyword_group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/GeneralSGModels/html/group\_\_Rover\_keyword\_group.html)
- [Rover!Dynamics](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/GeneralSGModels/html/group__Rover_Dynamics_keyword_group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/GeneralSGModels/html/group\_\_Rover\_Dynamics\_keyword\_group.html)
- [Spring/Damper](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/GeneralSGModels/html/group__Spring-Damper_keyword_group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/GeneralSGModels/html/group\_\_Spring-Damper\_keyword\_group.html)

#### Description

Simple spring-damper model

This model computes and applies a generalized forces for a simple 6 dof spring-damper that connects two different Darts bodies by a hinge. The connection force is always double-sided (tension and/or compression). This model has no flowIns or flowOuts. It makes direct Darts calls to compute the positions of each attachment point (node1 and node2) and applies the constraint forces. This model is not ModelMex compliant.

from the GeneralSGModels models library for the xref:Dshellpp\_module simulation framework.

#### Model SpringDamperMotor6dof class details

For more information on the members and functions of this model class, please see [GeneralSGModels::SpringDamperMotor6dof model class doxygen documentation](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/GeneralSGModels/html/group__SpringDamperMotor6dof_group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/GeneralSGModels/html/group\_\_SpringDamperMotor6dof\_group.html)

#### Parameters

Name	Type	Size	Quantity	Units	Description
C	double	36	Mixed		damping constant (6x6 matrix)

Name	Type	Size	Quantity	Units	Description
K	double	36	Mixed		spring force constant (6x6 matrix)
unsprung_length	double	6	Mixed		Nominal unsprung length of spring (6-vector; first 3 values=translational, last 3 values=rotational)

#### Scratch

Name	Type	Size	Quantity	Units	Description
gen_coord	double	7	Mixed		Hinge coordinates (7 vec; first 3=translation, last 4=quaternion)
gen_f	double	6	Mixed		Generalized force on hinge (6 vec)
gen_vel	double	6	Mixed		Hinge velocity (6 vec)

## 29.27. GeneralSGModels::TiltVector Sensor Model



This model documentation is auto-generated. Documentation updates should be made to the model's `mdl` file to avoid the changes being lost.

[Doxygen link to all GeneralSGModels models of Sensor class](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/GeneralSGModels/html/group__Sensor_group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/GeneralSGModels/html/group\_\_Sensor\_group.html)

#### Keywords Doxygen groups

- [Rover](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/GeneralSGModels/html/group__Rover_keyword_group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/GeneralSGModels/html/group\_\_Rover\_keyword\_group.html)
- [Rover!Sensor](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/GeneralSGModels/html/group__Rover_Sensor_keyword_group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/GeneralSGModels/html/group\_\_Rover\_Sensor\_keyword\_group.html)
- [Rover!Tilt](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/GeneralSGModels/html/group__Rover_Tilt_keyword_group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/GeneralSGModels/html/group\_\_Rover\_Tilt\_keyword\_group.html)
- [vector](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/GeneralSGModels/html/group__vector_keyword_group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/GeneralSGModels/html/group\_\_vector\_keyword\_group.html)

#### Description

Provides the pointing direction of the vehicle's tilt vector

This is an abstract of the accelerometer that is used for the pointing of rover's downward direction.

from the GeneralSGModels models library for the xref:Dshellpp\_module simulation framework.

#### Model TiltVector class details

For more information on the members and functions of this model class, please see [GeneralSGModels::TiltVector model class doxygen documentation](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/GeneralSGModels/html/group__TiltVector_group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/GeneralSGModels/html/group\_\_TiltVector\_group.html)

#### Parameters

Name	Type	Size	Quantity	Units	Description
tvec	double	3	Dimensionless		tilt vector on the local frame

#### Flow Outs

Name	Type	Size	Quantity	Units	Description
tvec_global	double	3	Dimensionless		tilt vector in the global frame

## 29.28. GeneralSGModels::UjointSpringDamper Motor Model



This model documentation is auto-generated. Documentation updates should be made to the model's `mdl` file to avoid the changes being lost.

[Doxygen link to all GeneralSGModels models of Motor class](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/GeneralSGModels/html/group__Motor__group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/GeneralSGModels/html/group\_\_Motor\_\_group.html)

#### Keywords Doxygen groups

- [Hinge](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/GeneralSGModels/html/group__Hinge__keyword__group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/GeneralSGModels/html/group\_\_Hinge\_\_keyword\_\_group.html)
- [HingeUjoint](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/GeneralSGModels/html/group__Hinge__Ujoint__keyword__group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/GeneralSGModels/html/group\_\_Hinge\_\_Ujoint\_\_keyword\_\_group.html)

#### Description

Spring Damper for U-joints

This model computes and applies generalized forces representing spring dampers in a U-joint.

from the GeneralSGModels models library for the xref:Dshellpp\_module simulation framework.

#### Model UjointSpringDamper class details

For more information on the members and functions of this model class, please see [GeneralSGModels::UjointSpringDamper model class doxygen documentation](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/GeneralSGModels/html/group__UjointSpringDamper__group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/GeneralSGModels/html/group\_\_UjointSpringDamper\_\_group.html)

#### Parameters

Name	Type	Size	Quantity	Units	Description
C	double	2	RotationalSpringDamping		damping constants
K	double	2	RotationalSpringStiffness		spring force constants

#### Scratch

Name	Type	Size	Quantity	Units	Description
gen_coord	double	2	Angle		Hinge coordinates
gen_f	double	2	Torque		Generalized force on hinge
gen_vel	double	2	AngularVelocity		Hinge velocity



## 30. GravitySGModels Dshell model library

### Doxygen reference to Models in GravitySGModels by Type

- [Actuators](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/GravitySGModels/html/group__Actuator__group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/GravitySGModels/html/group\_\_Actuator\_\_group.html)
- [Sensors](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/GravitySGModels/html/group__Sensor__group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/GravitySGModels/html/group\_\_Sensor\_\_group.html)
- [Motors](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/GravitySGModels/html/group__Motor__group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/GravitySGModels/html/group\_\_Motor\_\_group.html)
- [Encoders](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/GravitySGModels/html/group__Encoder__group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/GravitySGModels/html/group\_\_Encoder\_\_group.html)
- [Flows](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/GravitySGModels/html/group__Flow__group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/GravitySGModels/html/group\_\_Flow\_\_group.html)
- [Continuous](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/GravitySGModels/html/group__ContStates__group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/GravitySGModels/html/group\_\_ContStates\_\_group.html)

### 30.1. GravitySGModels::SphericalHarmonicGravity Actuator Model



This model documentation is auto-generated. Documentation updates should be made to the model's `mdl` file to avoid the changes being lost.

[Doxygen link to all GravitySGModels models of Actuator class](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/GravitySGModels/html/group__Actuator__group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/GravitySGModels/html/group\_\_Actuator\_\_group.html)

#### Description

SphericalHarmonicGravity - a NxM gravity Model

Author: NASA/JSC COMPASS Team

from the GravitySGModels models library for the xref:Dshellpp\_module simulation framework.

#### Model SphericalHarmonicGravity class details

For more information on the members and functions of this model class, please see [GravitySGModels::SphericalHarmonicGravity model class doxygen documentation](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/GravitySGModels/html/group__SphericalHarmonicGravity__group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/GravitySGModels/html/group\_\_SphericalHarmonicGravity\_\_group.html)

#### Parameters

Name	Type	Size	Quantity	Units	Description
PCI_FRAME_UUID	int				<b>Not for user input</b> - UUID for the planet-centered rotating frame.
PCR_FRAME_UUID	int				<b>Not for user input</b> - UUID for the planet-centered rotating frame.
apply	bool				Flag to indicate if computed gravitational acceleration should be applied to node.
cCoefficients	double	*	Unspecified		Gravitational C Coefficients
degree	int				Gravity Model Degree (tesserals)
equatorialRadius	double		Length		Central body equatorial radius
gravityParameter	double		GmGravitationalParameter		Central body gravitational parameter - GM
normalized	bool				Flag to indicate if coefficients (C and S) are normalized and need to be un-normalized after input.
order	int				Gravity Model Order (zonals)
sCoefficients	double	*	Unspecified		Gravitational S Coefficients
useGravityGradient	bool				Use Gravity Gradient Flag

**Scratch**

Name	Type	Size	Quantity	Units	Description
angAccelBody	double	3	AngularAcceleration		Angular acceleration vector in the body frame
angAccelInertial	double	3	AngularAcceleration		Angular acceleration vector in the PCI frame
angularAccelMag	double		AngularAcceleration		Magnitude of the angular acceleration vector in node/body frame
gravGradientTorque	double	3	Torque		Gravity gradient torque vector in the node/body frame
gravGradientTorqueMag	double		Torque		Magnitude of the gravity gradient torque vector
linearAccelMag	double		Acceleration		Magnitude of the linear acceleration vector in PCR frame
linearAccelPCR	double	3	Acceleration		Linear gravitational acceleration in PCR frame
relativePosition	double	3	Length		Relative position in Planet-Centered Rotating (PCR) frame
relativePositionBody	double	3	Length		Relative position in body frame

**Flow Outs**

Name	Type	Size	Quantity	Units	Description
angularAccel	double	3	AngularAcceleration		Angular/Rotational gravitational acceleration in PCI frame
linearAccel	double	3	Acceleration		Linear gravitational acceleration in PCI frame
potential	double		SpecificEnergy		Gravitational potential

## 31. RoverNavModels Dshell model library

### Doxygen reference to Models in RoverNavModels by Type

- [Actuators](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/RoverNavModels/html/group__Actuator__group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/RoverNavModels/html/group\_\_Actuator\_\_group.html)
- [Sensors](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/RoverNavModels/html/group__Sensor__group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/RoverNavModels/html/group\_\_Sensor\_\_group.html)
- [Motors](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/RoverNavModels/html/group__Motor__group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/RoverNavModels/html/group\_\_Motor\_\_group.html)
- [Encoders](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/RoverNavModels/html/group__Encoder__group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/RoverNavModels/html/group\_\_Encoder\_\_group.html)
- [Flows](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/RoverNavModels/html/group__Flow__group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/RoverNavModels/html/group\_\_Flow\_\_group.html)
- [Continuous](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/RoverNavModels/html/group__ContStates__group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/RoverNavModels/html/group\_\_ContStates\_\_group.html)

### 31.1. RoverNavModels::ArcTraj Flow Model



This model documentation is auto-generated. Documentation updates should be made to the model's `mdl` file to avoid the changes being lost.

[Doxygen link to all RoverNavModels models of Flow class](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/RoverNavModels/html/group__Flow__group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/RoverNavModels/html/group\_\_Flow\_\_group.html)

#### Keywords Doxygen groups

- [Rover](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/RoverNavModels/html/group__Rover__keyword__group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/RoverNavModels/html/group\_\_Rover\_\_keyword\_\_group.html)
- [Rover!Controller](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/RoverNavModels/html/group__Rover__Controller__keyword__group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/RoverNavModels/html/group\_\_Rover\_\_Controller\_\_keyword\_\_group.html)
- [Rover!Locomotion](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/RoverNavModels/html/group__Rover__Locomotion__keyword__group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/RoverNavModels/html/group\_\_Rover\_\_Locomotion\_\_keyword\_\_group.html)

#### Description

Generate vehicle chassis linear and angular velocity from radius and mode produced by the RoverNavDyn.

from the {RoverNavModels\_module\_uri}[RoverNavModels] models library for the xref:Dshellpp\_module simulation framework.

#### Model ArcTraj class details

For more information on the members and functions of this model class, please see [RoverNavModels::ArcTraj model class doxygen documentation](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/RoverNavModels/html/group__ArcTraj__group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/RoverNavModels/html/group\_\_ArcTraj\_\_group.html)

#### Parameters

Name	Type	Size	Quantity	Units	Description
nominal_angular_velocity	double		AngularVelocity		Nominal angular velocity to use
nominal_linear_velocity	double		Velocity		Nominal linear speed to use

#### Flow Ins

Name	Type	Size	Quantity	Units	Description
mode	int				0 → turn in place, 1 → move forward, -1 → move reverse
radius	double		Length		turning radius (0 → straight, positive → left turn, negative → right turn)

#### Flow Outs

Name	Type	Size	Quantity	Units	Description
angular_velocity	double		AngularVelocity		Chassis angular velocity, +ve → left turn, -ve → right turn
linear_velocity	double		Velocity		Chassis linear velocity, +ve → forward, -ve → backward

## 31.2. RoverNavModels::ControlStatus Flow Model



This model documentation is auto-generated. Documentation updates should be made to the model's `mdl` file to avoid the changes being lost.

[Doxygen link to all RoverNavModels models of Flow class](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/RoverNavModels/html/group_Flow_group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/RoverNavModels/html/group\_Flow\_group.html)

### Keywords Doxygen groups

- [Rover](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/RoverNavModels/html/group_Rover_keyword_group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/RoverNavModels/html/group\_Rover\_keyword\_group.html)
- [Rover!Controller](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/RoverNavModels/html/group_Rover_Controller_keyword_group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/RoverNavModels/html/group\_Rover\_Controller\_keyword\_group.html)
- [Rover!Locomotion](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/RoverNavModels/html/group_Rover_Locomotion_keyword_group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/RoverNavModels/html/group\_Rover\_Locomotion\_keyword\_group.html)

### Description

Model to determine if commanded angles have been achieved.

from the {RoverNavModels\_module\_uri}[RoverNavModels] models library for the xref:Dshellpp\_module simulation framework.

### Model ControlStatus class details

For more information on the members and functions of this model class, please see [RoverNavModels::ControlStatus model class doxygen documentation](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/RoverNavModels/html/group_ControlStatus_group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/RoverNavModels/html/group\_ControlStatus\_group.html)

### Parameters

Name	Type	Size	Quantity	Units	Description
dimension	int				Dimension of input vectors
epsilon	double		Angle		tolerance (radians) to determine if motion is complete

### Flow Ins

Name	Type	Size	Quantity	Units	Description
angle	double	*	Angle		Current angles
angle_d	double	*	Angle		Desired angles

### Flow Outs

Name	Type	Size	Quantity	Units	Description
status	int				status (false → still controlling to desired values, true → done)

## 31.3. RoverNavModels::DriveTrain4x4 Sensor Model



This model documentation is auto-generated. Documentation updates should be made to the model's `mdl` file to avoid the changes being lost.

[Doxygen link to all RoverNavModels models of Sensor class](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/RoverNavModels/html/group_Sensor_group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/RoverNavModels/html/group\_Sensor\_group.html)

### Keywords Doxygen groups

- [Rover](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/RoverNavModels/html/group_Rover_keyword_group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/RoverNavModels/html/group\_Rover\_keyword\_group.html)
- [Rover!Locomotion](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/RoverNavModels/html/group_Rover_Locomotion_keyword_group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/RoverNavModels/html/group\_Rover\_Locomotion\_keyword\_group.html)
- [Rover!Odometry](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/RoverNavModels/html/group_Rover_Odometry_keyword_group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/RoverNavModels/html/group\_Rover\_Odometry\_keyword\_group.html)
- [Rover!n-wheel](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/RoverNavModels/html/group_Rover_n-wheel_keyword_group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/RoverNavModels/html/group\_Rover\_n-wheel\_keyword\_group.html)

### Description

Model used as part of drive train modeling for a 4x4 vehicle for distributing engine torque to wheels.

from the {RoverNavModels\_module\_uri}[RoverNavModels] models library for the xref:Dshellpp\_module simulation framework.

### Model DriveTrain4x4 class details

For more information on the members and functions of this model class, please see [RoverNavModels::DriveTrain4x4 model class doxygen documentation](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/RoverNavModels/html/group_DriveTrain4x4_group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/RoverNavModels/html/group\_DriveTrain4x4\_group.html)

### Parameters

Name	Type	Size	Quantity	Units	Description
max_steer_angle	double		Angle		The maximum absolute angle for the steer angle.
pitman_frame_uuid	unsigned long (ulong)				The UUID for the Pitman steering frame.
zero_axis	double	3	Dimensionless		The effective wheel rotation axis for zero steer.

### Scratch

Name	Type	Size	Quantity	Units	Description
desired_yaw_angle	double		Angle		The angle wrt forward direction for the pitman axis.
motion_direction	int				Forward/backward motion direction of the vehicle.
raw_yaw_angle	double		Angle		The unclamped, raw value for the pitman steering angle.
turn_center2pitman_vec	double	3	Length		The turn center to pitman frame vector in the chassis frame.
turn_radius	double		Length		The turning radius.

### Flow Ins

Name	Type	Size	Quantity	Units	Description
chassis_motion2turn_center_f2f_uuid	unsigned long (ulong)				Chassis motion to turn center frame location frame2frame UUID.
desired_x_y_h_rate	double	3	Mixed		The desired instantaneous chassis x,y,h velocity

### Flow Outs

Name	Type	Size	Quantity	Units	Description
current_chassis_speed	double		Velocity		The current speed of the chassis
desired_chassis_speed	double		Velocity		The desired speed of the chassis
desired_steer_angle	double		Angle		Instantaneous desired steer angle for the chassis

## 31.4. RoverNavModels::DriveTrainAccel Sensor Model



This model documentation is auto-generated. Documentation updates should be made to the model's `mdl` file to avoid the changes being lost.

[Doxygen link to all RoverNavModels models of Sensor class](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/RoverNavModels/html/group_Sensor_group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/RoverNavModels/html/group\_Sensor\_group.html)

### Keywords Doxygen groups

- [Rover](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/RoverNavModels/html/group__Rover_keyword_group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/RoverNavModels/html/group\_\_Rover\_keyword\_group.html)
- [Rover!Locomotion](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/RoverNavModels/html/group__Rover_Locomotion_keyword_group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/RoverNavModels/html/group\_\_Rover\_Locomotion\_keyword\_group.html)
- [Rover!Odometry](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/RoverNavModels/html/group__Rover_Odometry_keyword_group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/RoverNavModels/html/group\_\_Rover\_Odometry\_keyword\_group.html)
- [Rover!n-wheel](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/RoverNavModels/html/group__Rover_n-wheel_keyword_group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/RoverNavModels/html/group\_\_Rover\_n-wheel\_keyword\_group.html)

### Description

Model used as part of drive train modeling for a 4x4 vehicle's accelerator/brake.

from the {RoverNavModels\_module\_uri}[RoverNavModels] models library for the xref:Dshellpp\_module simulation framework.

### Model DriveTrainAccel class details

For more information on the members and functions of this model class, please see [RoverNavModels::DriveTrainAccel model class doxygen documentation](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/RoverNavModels/html/group__DriveTrainAccel_group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/RoverNavModels/html/group\_\_DriveTrainAccel\_group.html)

### Flow Ins

Name	Type	Size	Quantity	Units	Description
desired_chassis_accel	double		Acceleration		The desired instantaneous accelerator/brake acceleration

### Flow Outs

Name	Type	Size	Quantity	Units	Description
current_chassis_speed	double		Velocity		The current speed of the chassis
desired_chassis_speed	double		Velocity		The desired speed of the chassis

## 31.5. RoverNavModels::DriveTrainSteering Flow Model



This model documentation is auto-generated. Documentation updates should be made to the model's `mdl` file to avoid the changes being lost.

[Doxygen link to all RoverNavModels models of Flow class](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/RoverNavModels/html/group__Flow_group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/RoverNavModels/html/group\_\_Flow\_group.html)

### Keywords Doxygen groups

- [Rover](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/RoverNavModels/html/group__Rover_keyword_group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/RoverNavModels/html/group\_\_Rover\_keyword\_group.html)
- [Rover!Locomotion](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/RoverNavModels/html/group__Rover_Locomotion_keyword_group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/RoverNavModels/html/group\_\_Rover\_Locomotion\_keyword\_group.html)
- [Rover!Odometry](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/RoverNavModels/html/group__Rover_Odometry_keyword_group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/RoverNavModels/html/group\_\_Rover\_Odometry\_keyword\_group.html)
- [Rover!n-wheel](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/RoverNavModels/html/group__Rover_n-wheel_keyword_group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/RoverNavModels/html/group\_\_Rover\_n-wheel\_keyword\_group.html)

### Description

Model used as part of drive train modeling for a 4x4 vehicle for distributing engine torque to wheels.

from the {RoverNavModels\_module\_uri}[RoverNavModels] models library for the xref:Dshellpp\_module simulation framework.

### Model DriveTrainSteering class details

For more information on the members and functions of this model class, please see [RoverNavModels::DriveTrainSteering model class doxygen documentation](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/RoverNavModels/html/group__DriveTrainSteering_group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/RoverNavModels/html/group\_\_DriveTrainSteering\_group.html)

### Flow Ins

Name	Type	Size	Quantity	Units	Description
desired_chassis_angular_velocity	double		AngularVelocity		The desired instantaneous chassis angular velocity

Name	Type	Size	Quantity	Units	Description
desired_chassis_speed	double		Velocity		The desired speed of the chassis

#### Flow Outs

Name	Type	Size	Quantity	Units	Description
desired_steer_angle	double		Angle		Instantaneous desired steer angle for the chassis

## 31.6. RoverNavModels::Locomotion Flow Model



This model documentation is auto-generated. Documentation updates should be made to the model's `mdl` file to avoid the changes being lost.

[Doxygen link to all RoverNavModels models of Flow class](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/RoverNavModels/html/group_Flow_group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/RoverNavModels/html/group\_Flow\_group.html)

#### Keywords Doxygen groups

- [Rover](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/RoverNavModels/html/group_Rover_keyword_group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/RoverNavModels/html/group\_Rover\_keyword\_group.html)
- [Rover!Locomotion](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/RoverNavModels/html/group_Rover_Locomotion_keyword_group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/RoverNavModels/html/group\_Rover\_Locomotion\_keyword\_group.html)

#### Description

Model of rover locomotion. The input is a commanded motion. This model splits a motion into smaller submotions - commanding a new (x,y,heading) for each small sub-move. The distance between sub-motions is commanded by the `stepDist` (for linear) and `stepAngle` (for turn-in-place) parameters.

from the {RoverNavModels\_module\_uri}[RoverNavModels] models library for the xref:Dshellpp\_module simulation framework.

#### Model Locomotion class details

For more information on the members and functions of this model class, please see [RoverNavModels::Locomotion model class doxygen documentation](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/RoverNavModels/html/group_Locomotion_group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/RoverNavModels/html/group\_Locomotion\_group.html)

#### Parameters

Name	Type	Size	Quantity	Units	Description
look_move_direction	double	2	Dimensionless		x/y direction of forward motion for rover when heading is zero
pivotOffsetX	double		Length		x-direction offset from chassis frame of pivot point. (body frame)
stepAngle	double		Angle		Sub-motion angle change (for turn-in-place)
stepDist	double		Length		Sub-motion travel distance

#### Scratch

Name	Type	Size	Quantity	Units	Description
deltaAngle	double		Angle		The heading angle delta (nominally <code>stepAngle</code> , except when close to completion of turn)

#### Discrete States

Name	Type	Size	Quantity	Units	Description
accumAngle	double		Angle		Accumulated angle change since last command

Name	Type	Size	Quantity	Units	Description
accumDist	double		Length		Accumulated distance since last command
status	int				Current status (0 → moving, 1 → ready for new command)

#### Flow Ins

Name	Type	Size	Quantity	Units	Description
angle	double		Angle		angle to tranverse when turning in place
distance	double		Length		total distance to traverse when moving forward/backward
heading_in	double		Angle		Current heading
mode	int				mode (0 → turn in place, 1 → move forward, -1 → move reverse)
radius	double		Length		turning radius (0 → straight, positive → left turn, negative → right turn)
xpos	double		Length		Current x position
ypos	double		Length		Current y position

#### Flow Outs

Name	Type	Size	Quantity	Units	Description
direction	int				motion direction
heading	double		Angle		Commanded heading
status	int				0 → still moving, 1 → done moving (ready for new command)
x	double		Length		Commanded x position
y	double		Length		Commanded y position

### 31.7. RoverNavModels::NavOdometry Actuator Model



This model documentation is auto-generated. Documentation updates should be made to the model's `mdl` file to avoid the changes being lost.

[Doxygen link to all RoverNavModels models of Actuator class](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/RoverNavModels/html/group__Actuator_group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/RoverNavModels/html/group\_\_Actuator\_group.html)

#### Keywords Doxygen groups

- [Rover](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/RoverNavModels/html/group__Rover_keyword_group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/RoverNavModels/html/group\_\_Rover\_keyword\_group.html)
- [Rover!6wheel](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/RoverNavModels/html/group__Rover_6wheel_keyword_group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/RoverNavModels/html/group\_\_Rover\_6wheel\_keyword\_group.html)
- [Rover!Locomotion](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/RoverNavModels/html/group__Rover_Locomotion_keyword_group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/RoverNavModels/html/group\_\_Rover\_Locomotion\_keyword\_group.html)
- [Rover!Odometry](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/RoverNavModels/html/group__Rover_Odometry_keyword_group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/RoverNavModels/html/group\_\_Rover\_Odometry\_keyword\_group.html)

#### Description

Model of rover odometry for a 6 wheeled rover where all 6 wheels can be steered.

from the {RoverNavModels\_module\_uri}[RoverNavModels] models library for the xref:Dshellpp\_module simulation framework.



### Model NavOdometry class details

For more information on the members and functions of this model class, please see [RoverNavModels::NavOdometry model class doxygen documentation](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/RoverNavModels/html/group__NavOdometry_group.html) ([https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/RoverNavModels/html/group\\_\\_NavOdometry\\_group.html](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/RoverNavModels/html/group__NavOdometry_group.html))

### Parameters

Name	Type	Size	Quantity	Units	Description
bogeyFront	double		Flag		+1 → bogey is on forward side of rover. -1 → rocker is on forward side of rover
direction	double		Flag		+1 → rover moves towards bogey when wheels rotate positive. -1 → rover moves towards rocker when wheels rotate positive
epsilon	double		Angle		Acceptable wheel encoder error
maxAngularVelocity	double		AngularVelocity		Maximum angular velocity for wheels (default = 0)
minAngularVelocity	double		AngularVelocity		Minimum angular velocity for wheels (default = 0)
wheelCenterBias	double		Length		Bias of center wheel (positive implies closer to front side wheels, zero implies in center of front and back wheels)
wheelFrontRearDist	double		Length		Distance between front and back side wheels
wheelLeftRightDist	double		Length		Distance between left and right side wheels
wheelRadius	double		Length		Wheel radius

### Discrete States

Name	Type	Size	Quantity	Units	Description
leftBogeySideAngle	double		Angle		Wheel encoder angles needed to complete motion for left front wheel
leftBogeySideSpeed	double		Dimensionless		Speed multiplier for left front wheel (ranges from 0 (minimum speed) to 1 (max))
leftBogeySideStartAngle	double		Angle		Wheel encoder angle at start of motion
leftCenterAngle	double		Angle		Wheel encoder angles needed to complete motion for left center wheel
leftCenterSpeed	double		Dimensionless		Speed multiplier for left center wheel (ranges from 0 (minimum speed) to 1 (max))
leftCenterStartAngle	double		Angle		Wheel encoder angle at start of motion
leftRockerSideAngle	double		Angle		Wheel encoder angles needed to complete motion for left back wheel
leftRockerSideSpeed	double		Dimensionless		Speed multiplier for left back wheel (ranges from 0 (minimum speed) to 1 (max))

Name	Type	Size	Quantity	Units	Description
leftRockerSideStartAngle	double		Angle		Wheel encoder angle at start of motion
rightBogeySideAngle	double		Angle		Wheel encoder angles needed to complete motion for right front wheel
rightBogeySideSpeed	double		Dimensionless		Speed multiplier for right front wheel (ranges from 0 (minimum speed) to 1 (max))
rightBogeySideStartAngle	double		Angle		Wheel encoder angle at start of motion
rightCenterAngle	double		Angle		Wheel encoder angles needed to complete motion for right center wheel
rightCenterSpeed	double		Dimensionless		Speed multiplier for right center wheel (ranges from 0 (minimum speed) to 1 (max))
rightCenterStartAngle	double		Angle		Wheel encoder angle at start of motion
rightRockerSideAngle	double		Angle		Wheel encoder angles needed to complete motion for right back wheel
rightRockerSideSpeed	double		Dimensionless		Speed multiplier for right back wheel (ranges from 0 (minimum speed) to 1 (max))
rightRockerSideStartAngle	double		Angle		Wheel encoder angle at start of motion
startTime	double		Time		Start time of current motion (used to interpolate output angles for position control)
status	int				Current status (0 → moving, 1 → ready for new command)

#### Continuous States

Name	Type	Size	Quantity	Units	Description
leftBogeySideAngle	double	1	Angle		Left front desired angle
leftCenterAngle	double	1	Angle		Left center desired angle
leftRockerSideAngle	double	1	Angle		Left rear desired angle
rightBogeySideAngle	double	1	Angle		Right front desired angle
rightCenterAngle	double	1	Angle		Right center desired angle
rightRockerSideAngle	double	1	Angle		Right rear desired angle

#### Flow Ins

Name	Type	Size	Quantity	Units	Description
BogeySideLeftEnc	double		Angle		Current value of left front wheel encoder
BogeySideRightEnc	double		Angle		Current value of right front wheel encoder
CenterLeftEnc	double		Angle		Current value of left center wheel encoder
CenterRightEnc	double		Angle		Current value of right center wheel encoder

Name	Type	Size	Quantity	Units	Description
RockerSideLeftEnc	double		Angle		Current value of left back wheel encoder
RockerSideRightEnc	double		Angle		Current value of right back wheel encoder
angle	double		Angle		angle to tranverse when turning in place
distance	double		Length		distance to traverse when moving forward/backward
mode	int				mode (0 → turn in place, 1 → move forward, -1 → move reverse)
radius	double		Length		turning radius (0 → straight, positive → left turn, negative → right turn)
speed	double		Dimensionless		0 → minimum speed, 1 → maximum speed
steerStatus	double		Flag		0 → steering not done, 1 → steering complete

#### Flow Outs

Name	Type	Size	Quantity	Units	Description
BogeySideLeftAngle	double		Angle		Desired wheel angle for left front wheel
BogeySideLeftRate	double		AngularVelocity		Desired wheel rate for left front wheel
BogeySideRightAngle	double		Angle		Desired wheel angle for right front wheel
BogeySideRightRate	double		AngularVelocity		Desired wheel rate for right front wheel
CenterLeftAngle	double		Angle		Desired wheel angle for left center wheel
CenterLeftRate	double		AngularVelocity		Desired wheel rate for left center wheel
CenterRightAngle	double		Angle		Desired wheel angle for right center wheel
CenterRightRate	double		AngularVelocity		Desired wheel rate for right center wheel
RockerSideLeftAngle	double		Angle		Desired wheel angle for left back wheel
RockerSideLeftRate	double		AngularVelocity		Desired wheel rate for left back wheel
RockerSideRightAngle	double		Angle		Desired wheel angle for right back wheel
RockerSideRightRate	double		AngularVelocity		Desired wheel rate for right back wheel
status	int				0 → still moving, 1 → done moving (ready for new command)



This model documentation is auto-generated. Documentation updates should be made to the model's `mdl` file to avoid the changes being lost.

[Doxygen link to all RoverNavModels models of Actuator class](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/RoverNavModels/html/group__Actuator_group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/RoverNavModels/html/group\_\_Actuator\_group.html)

#### Keywords Doxygen groups

- [Rover](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/RoverNavModels/html/group__Rover_keyword_group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/RoverNavModels/html/group\_\_Rover\_keyword\_group.html)
- [Rover!2](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/RoverNavModels/html/group__Rover_2_keyword_group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/RoverNavModels/html/group\_\_Rover\_2\_keyword\_group.html)
- [Rover!Locomotion](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/RoverNavModels/html/group__Rover_Locomotion_keyword_group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/RoverNavModels/html/group\_\_Rover\_Locomotion\_keyword\_group.html)
- [Rover!Odometry](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/RoverNavModels/html/group__Rover_Odometry_keyword_group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/RoverNavModels/html/group\_\_Rover\_Odometry\_keyword\_group.html)
- [wheel](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/RoverNavModels/html/group__wheel_keyword_group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/RoverNavModels/html/group\_\_wheel\_keyword\_group.html)

#### Description

Model of rover odometry for a 6 wheeled rover where only 2 wheels are actively driven/steered (as in Rocky7).

from the {RoverNavModels\_module\_uri}[RoverNavModels] models library for the xref:Dshellpp\_module simulation framework.

#### Model NavOdometry2W class details

For more information on the members and functions of this model class, please see [RoverNavModels::NavOdometry2W model class doxygen documentation](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/RoverNavModels/html/group__NavOdometry2W_group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/RoverNavModels/html/group\_\_NavOdometry2W\_group.html)

#### Parameters

Name	Type	Size	Quantity	Units	Description
direction	double		Flag		+1 → implies rover moves forward when wheels turn forward. -1 → rover moves backwards when wheels move forward
epsilon	double		Angle		Acceptable wheel encoder error
maxAngularVelocity	double		AngularVelocity		Maximum angular velocity for wheels (default = 0)
minAngularVelocity	double		AngularVelocity		Minimum angular velocity for wheels (default = 0)
wheelCenterBias	double		Length		Bias of center wheel (positive implies closer to front side wheels, zero implies in center of front and back wheels)
wheelFrontBackDist	double		Length		Distance between front and back side wheels
wheelLeftRightDist	double		Length		Distance between left and right side wheels
wheelRadius	double		Length		Wheel radius

#### Discrete States

Name	Type	Size	Quantity	Units	Description
leftAngle	double		Angle		Wheel encoder angles needed to complete motion for left side driven wheel
leftSpeed	double		Dimensionless		Speed multiplier for left side wheel (ranges from 0 (minimum speed) to 1 (max))
rightAngle	double		Angle		Wheel encoder angles needed to complete motion for right side driven wheel

Name	Type	Size	Quantity	Units	Description
rightSpeed	double		Dimensionless		Speed multiplier for right side wheel (ranges from 0 (minimum speed) to 1 (max))
status	int				Current status (0 → moving, 1 → ready for new command)

#### Flow Ins

Name	Type	Size	Quantity	Units	Description
angle	double		Angle		angle to tranverse when turning in place
distance	double		Length		distance to traverse when moving forward/backward
leftEnc	double		Angle		Current value of left side wheel encoder
mode	double		Flag		mode (0 → turn in place, 1 → move forward, -1 → move reverse)
radius	double		Length		turning radius (0 → straight, positive → left turn, negative → right turn)
rightEnc	double		Angle		Current value of right side wheel encoder
speed	double		Dimensionless		0 → minimum speed, 1 → maximum speed
steerLeft	double		Angle		Current steering angle for left side wheel
steerRight	double		Angle		Current steering angle for right side wheel

#### Flow Outs

Name	Type	Size	Quantity	Units	Description
leftRate	double		AngularVelocity		Desired wheel rate for left side wheel
rightRate	double		AngularVelocity		Desired wheel rate for right side wheel
status	double		Flag		0 → still moving, 1 → done moving (ready for new command)
steerLeft	double		Angle		Desired steering angle for left side wheel
steerRight	double		Angle		Desired steering angle for right side wheel

### 31.9. RoverNavModels::RoverNavDyn Flow Model



This model documentation is auto-generated. Documentation updates should be made to the model's `mdl` file to avoid the changes being lost.

[Doxygen link to all RoverNavModels models of Flow class](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/RoverNavModels/html/group_Flow_group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/RoverNavModels/html/group\_Flow\_group.html)

**Keywords Doxygen groups**

- [Rover](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/RoverNavModels/html/group_Rover_keyword_group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/RoverNavModels/html/group\_Rover\_keyword\_group.html)

- [Rover!Navigation!Sojourner](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/RoverNavModels/html/group__Rover__Navigation__Sojourner__keyword__group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/RoverNavModels/html/group\_\_Rover\_\_Navigation\_\_Sojourner\_\_keyword\_\_group.html)

### Description

Model of autonomous rover navigation

Given the obstacle detection camera view, this model navigates the rover to a given goal. This model was built to interface to the NavOdometry model which provides odometry estimates on the motion needed to complete the motion commanded by this model.

from the {RoverNavModels\_module\_uri}[RoverNavModels] models library for the xref:Dshellpp\_module simulation framework.

### Model RoverNavDyn class details

For more information on the members and functions of this model class, please see [RoverNavModels::RoverNavDyn model class doxygen documentation](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/RoverNavModels/html/group__RoverNavDyn__group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/RoverNavModels/html/group\_\_RoverNavDyn\_\_group.html)

### Enums

#### ObsLocation Enum

In the RoverNavDyn model definition, the original enum ObsLocation is defined as:

```
enum ObsLocation
{
    None = 0,
    Left = 1,
    Center = 2,
    Right = 3,
    LeftCenter = 4,
    LeftRight = 5,
    CenterRight = 6,
    All = 7
};
```

In the C model code, these enum values can be accessed by using the follow C enum definitions:

```
/// RoverNavDynObsLocation enum
enum RoverNavDynObsLocation
{
    ROVER_NAV_DYN_OBS_LOCATION_NONE = 0,
    ROVER_NAV_DYN_OBS_LOCATION_LEFT = 1,
    ROVER_NAV_DYN_OBS_LOCATION_CENTER = 2,
    ROVER_NAV_DYN_OBS_LOCATION_RIGHT = 3,
    ROVER_NAV_DYN_OBS_LOCATION_LEFT_CENTER = 4,
    ROVER_NAV_DYN_OBS_LOCATION_LEFT_RIGHT = 5,
    ROVER_NAV_DYN_OBS_LOCATION_CENTER_RIGHT = 6,
    ROVER_NAV_DYN_OBS_LOCATION_ALL = 7
};
```

#### NavigationMode Enum

In the RoverNavDyn model definition, the original enum NavigationMode is defined as:

```
enum NavigationMode
{
    TurnToGoal = 0,
    LoopToGoal = 1,
    TurnInPlace = 2,
    ThreadNeedle = 3,
    Backup = 4
};
```

In the C model code, these enum values can be accessed by using the follow C enum definitions:

```
/// RoverNavDynNavigationMode enum
enum RoverNavDynNavigationMode
{
    ROVER_NAV_DYN_NAVIGATION_MODE_TURN_TO_GOAL = 0,
    ROVER_NAV_DYN_NAVIGATION_MODE_LOOP_TO_GOAL = 1,
    ROVER_NAV_DYN_NAVIGATION_MODE_TURN_IN_PLACE = 2,
    ROVER_NAV_DYN_NAVIGATION_MODE_THREAD_NEEDLE = 3,
    ROVER_NAV_DYN_NAVIGATION_MODE_BACKUP = 4
};
```

### Parameters

Name	Type	Size	Quantity	Units	Description
large_turn_radius	double		Length		large turning radius
look_move_direction	double	2	Dimensionless		x/y direction which camera looks: unit length

Name	Type	Size	Quantity	Units	Description
medium_pos_err	double		Length		medium position error
medium_rot_err	double		Angle		medium rotation error
nominal_rotation	double		Angle		nominal rotation
nominal_trans	double		Length		nominal straight motion (25 cm)
small_pos_err	double		Length		small position error
small_rot_err	double		Angle		small rotation error
small_turn_radius	double		Length		small turning radius
very_small_pos_err	double		Length		very small position error

#### Discrete States

Name	Type	Size	Quantity	Units	Description
alley_length	double		Length		length of needle alley
curr_mode	NavigationMode				Current navigation mode
curr_obstacle	ObsLocation				Current obstacle location
moveLength	double		Length		Total length of forward movement commands since start of simulation
prev_obstacle	ObsLocation				Previous obstacle location
turnLength	double		Angle		Total angle of turn in place commands since start of simulation

#### Flow Ins

Name	Type	Size	Quantity	Units	Description
ObstacleC	int				center obstacle if > 0
ObstacleL	int				left side obstacle if > 0
ObstacleR	int				right side obstacle if > 0
driveStatus	int				status of rover (0 → rover still competing previous drive command, 1 → rover ready for new command)
enable	int				List of flags to enable generation of new trajectory. This allows blocking of new trajectory generation to allow a variety of actions (all 1's → enable new trajectory, 0 → disable new trajectory)
heading_d	double		Angle		desired heading
heading_in	double		Angle		current heading
steerStatus	double		Flag		status of rover (0 → rover still competing previous steer command, 1 → rover may still be completing drive command)
xpos	double		Length		current x position
xpos_d	double		Length		desired x position

Name	Type	Size	Quantity	Units	Description
ypos	double		Length		current y position
ypos_d	double		Length		desired y position

#### Flow Outs

Name	Type	Size	Quantity	Units	Description
angle	double		Angle		angle to traverse when turning in place
at_goal	int				Flag for whether the rover has reached the goal location (not heading) (true/1 if at goal, false/0 otherwise)
at_heading	int				Flag for whether the rover has achieved the correct heading (true/1 if at heading, false/0 otherwise)
distance	double		Length		distance to travel (when moving forward/backward)
mode	int				0 → turn in place, 1 → move forward, -1 → move reverse
radius	double		Length		turning radius (0 → straight, positive → left turn, negative → right turn)
speed	double		Dimensionless		0 → minimum speed, 1 → maximum speed

### 31.10. RoverNavModels::RoverPosNavigation Flow Model



This model documentation is auto-generated. Documentation updates should be made to the model's `mdl` file to avoid the changes being lost.

[Doxygen link to all RoverNavModels models of Flow class](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/RoverNavModels/html/group_Flow_group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/RoverNavModels/html/group\_Flow\_group.html)

#### Keywords Doxygen groups

- [Rover](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/RoverNavModels/html/group_Rover_keyword_group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/RoverNavModels/html/group\_Rover\_keyword\_group.html)
- [Rover!Navigation!Sojourner](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/RoverNavModels/html/group_Rover_Navigation_Sojourner_keyword_group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/RoverNavModels/html/group\_Rover\_Navigation\_Sojourner\_keyword\_group.html)

#### Description

Model of autonomous rover navigation

Given the obstacle detection camera view, this model navigates the rover to a given goal. This model was built to interface to the WheelDriveMotion and WheelSteerDriveMotion model which provides computes the motion needed to complete the motion commanded by this model.

from the {RoverNavModels\_module\_uri}[RoverNavModels] models library for the xref:Dshellpp\_module simulation framework.

#### Model RoverPosNavigation class details

For more information on the members and functions of this model class, please see [RoverNavModels::RoverPosNavigation model class doxygen documentation](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/RoverNavModels/html/group_RoverPosNavigation_group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/RoverNavModels/html/group\_RoverPosNavigation\_group.html)

#### Enums

##### Obstacle Enum

In the RoverPosNavigation model definition, the original enum Obstacle is defined as:



```
enum Obstacle
{
    None = 0,
    Left = 1,
    Center = 2,
    Right = 3,
    LeftCenter = 4,
    LeftRight = 5,
    CenterRight = 6,
    All = 7
};
```

In the C model code, these enum values can be accessed by using the follow C enum definitions:

```
/// RoverPosNavigationObstacle enum
enum RoverPosNavigationObstacle
{
    ROVER_POS_NAVIGATION_OBSTACLE_NONE = 0,
    ROVER_POS_NAVIGATION_OBSTACLE_LEFT = 1,
    ROVER_POS_NAVIGATION_OBSTACLE_CENTER = 2,
    ROVER_POS_NAVIGATION_OBSTACLE_RIGHT = 3,
    ROVER_POS_NAVIGATION_OBSTACLE_LEFT_CENTER = 4,
    ROVER_POS_NAVIGATION_OBSTACLE_LEFT_RIGHT = 5,
    ROVER_POS_NAVIGATION_OBSTACLE_CENTER_RIGHT = 6,
    ROVER_POS_NAVIGATION_OBSTACLE_ALL = 7
};
```

### Mode Enum

In the RoverPosNavigation model definition, the original enum Mode is defined as:

```
enum Mode
{
    Disabled = 0,
    Enabled = 1,
    TurnToGoal = 2,
    TurnInPlaceToGoal = 3,
    SteerInPlaceToGoal = 4,
    DriveToGoal = 5,
    DriveLine = 6,
    DriveArc = 7,
    SteerArc = 8,
    TurnToHeading = 9,
    TurnInPlaceToHeading = 10,
    SteerInPlaceToHeading = 11,
    HandleObstacle = 12,
    SteerTurnInPlaceObstacle = 13,
    SteerLineInPlaceObstacle = 14,
    TurnInPlaceObstacle = 15,
    DriveForwardObstacle = 16,
    DriveBackwardObstacle = 17,
    UndefinedState = 18
};
```

In the C model code, these enum values can be accessed by using the follow C enum definitions:

```
/// RoverPosNavigationMode enum
enum RoverPosNavigationMode
{
    ROVER_POS_NAVIGATION_MODE_DISABLED = 0,
    ROVER_POS_NAVIGATION_MODE_ENABLED = 1,
    ROVER_POS_NAVIGATION_MODE_TURN_TO_GOAL = 2,
    ROVER_POS_NAVIGATION_MODE_TURN_IN_PLACE_TO_GOAL = 3,
    ROVER_POS_NAVIGATION_MODE_STEER_IN_PLACE_TO_GOAL = 4,
    ROVER_POS_NAVIGATION_MODE_DRIVE_TO_GOAL = 5,
    ROVER_POS_NAVIGATION_MODE_DRIVE_LINE = 6,
    ROVER_POS_NAVIGATION_MODE_DRIVE_ARC = 7,
    ROVER_POS_NAVIGATION_MODE_STEER_ARC = 8,
    ROVER_POS_NAVIGATION_MODE_TURN_TO_HEADING = 9,
    ROVER_POS_NAVIGATION_MODE_TURN_IN_PLACE_TO_HEADING = 10,
    ROVER_POS_NAVIGATION_MODE_STEER_IN_PLACE_TO_HEADING = 11,
    ROVER_POS_NAVIGATION_MODE_HANDLE_OBSTACLE = 12,
    ROVER_POS_NAVIGATION_MODE_STEER_TURN_IN_PLACE_OBSTACLE = 13,
    ROVER_POS_NAVIGATION_MODE_STEER_LINE_IN_PLACE_OBSTACLE = 14,
    ROVER_POS_NAVIGATION_MODE_TURN_IN_PLACE_OBSTACLE = 15,
    ROVER_POS_NAVIGATION_MODE_DRIVE_FORWARD_OBSTACLE = 16,
    ROVER_POS_NAVIGATION_MODE_DRIVE_BACKWARD_OBSTACLE = 17,
    ROVER_POS_NAVIGATION_MODE_UNDEFINED_STATE = 18
};
```

### Parameters

Name	Type	Size	Quantity	Units	Description
------	------	------	----------	-------	-------------

Name	Type	Size	Quantity	Units	Description
chassis_motion_frame_uuid	unsigned int (uint)				The UUID for the frame located at the chassis frame but oriented so X axis is along desired linear velocity.
ck_mode	int				A flag to indicate if using CK mode (=1) or not (=0).
large_turn_radius	double		Length		large turning radius
look_move_direction	double	2	Dimensionless		x/y direction which camera looks: unit length
medium_pos_err	double		Length		medium position error
medium_rot_err	double		Angle		medium rotation error
nominal_angular_velocity	double		AngularVelocity		Nominal angular velocity to use
nominal_linear_velocity	double		Velocity		Nominal linear speed to use
nominal_rotation	double		Angle		nominal rotation
nominal_trans	double		Length		nominal straight motion (25 cm)
obstacle_avoidance_motion_counter	double		Dimensionless		The number of cycles to continue turning or driving in an obstacle avoidance turn or drive.
obstacle_avoidance_motion_timeout	double		Dimensionless		The number of cycles before timing out on a motion state.
small_pos_err	double		Length		small position error
small_rot_err	double		Angle		small rotation error
small_turn_radius	double		Length		small turning radius
turn_center_frame_uuid	unsigned int (uint)				The UUID for the frame located at the arc turn center.
vehicle_type	int				Vehicle type: -1 = skid steered, 0 = partially steered, 1 = fully-steered
very_small_pos_err	double		Length		very small position error
very_small_rot_err	double		Angle		very_small rotation error

#### Discrete States

Name	Type	Size	Quantity	Units	Description
alley_length	double		Length		length of needle alley
curr_mode	Mode				Current navigation mode
curr_obstacle	Obstacle				Current obstacle location
motion_cycle_counter	double		Dimensionless		Updates the motion command if the counter exceeds a set number of cycles
moveLength	double		Length		Total length of forward movement commands since start of simulation

Name	Type	Size	Quantity	Units	Description
nextState	Mode				The next statechart state to transition to
obstacle_avoidance_heading	double		Angle		The desired heading to avoid an obstacle
obstacle_drive_location_x	double		Length		The x-component of the distance to drive when avoiding an obstacle
obstacle_drive_location_y	double		Length		The y-component of the distance forward to drive when avoiding an obstacle
prev_d_error	double		Length		previous distance error
prev_h_error	double		Angle		previous heading error
prev_obstacle	Obstacle				Previous obstacle location
prev_x_y_h	double	3	Mixed		previous x, y, h
turnLength	double		Angle		Total angle of turn in place commands since start of simulation

#### Flow Ins

Name	Type	Size	Quantity	Units	Description
ObstacleC	int				center obstacle if > 0
ObstacleL	int				left side obstacle if > 0
ObstacleR	int				right side obstacle if > 0
desired_x_y_h	double	3	Mixed		desired x,y,h position
driveStatus	int				status of rover (0 → rover still competing previous drive command, 1 → rover ready for new command)
enable	int				List of flags to enable generation of new trajectory. This allows blocking of new trajectory generation to allow a variety of actions (all 1's → enable new trajectory, 0 → disable new trajectory)
root2localization_f2f_uuid	unsigned long (ulong)				Root to localization frame location frame2frame UUID.
steerStatus	int				status of rover (0 → rover still competing previous steer command, 1 → rover may still be completing drive command)
x_y_h	double	3	Mixed		current x, y, h position

#### Flow Outs

Name	Type	Size	Quantity	Units	Description
angle	double		Angle		angle to traverse when turning in place
angular_velocity	double		AngularVelocity		angular velocity for arc motion

Name	Type	Size	Quantity	Units	Description
at_goal	int				Flag for whether the rover has reached the goal location (not heading) (true/1 if at goal, false/0 otherwise)
at_heading	int				Flag for whether the rover has achieved the correct heading (true/1 if at heading, false/0 otherwise)
chassis_motion2turn_center_f2f_uuid	unsigned long (ulong)				Chassis motion to turn center frame location frame2frame UUID.
direction	double		Dimensionless		The direction to drive in for vehicles that are capable to - i.e. all wheel steered. =0
distance	double		Length		distance to travel (when moving forward/backward)
linear_velocity	double		Velocity		linear velocity for arc motion
localization2chassis_motion_f2f_uuid	unsigned long (ulong)				Localization to chassis motion frame location frame2frame UUID.
mode	int				0 → turn in place, 1 → move forward, -1 → move reverse
nav_drive_state	int				Flag for indicating is wheels can drive (true/1 if steering completed, false/0 otherwise)
radius	double		Length		turning radius (0 → straight, positive → left turn, negative → right turn)
speed	double		Dimensionless		0 → minimum speed, 1 → maximum speed
x_y_h_out	double	3	Mixed		For CK mode, next x, y, h output

### 31.11. RoverNavModels::RoverPosNavigationFsm Flow Model



This model documentation is auto-generated. Documentation updates should be made to the model's `mdl` file to avoid the changes being lost.

[Doxygen link to all RoverNavModels models of Flow class](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/RoverNavModels/html/group_Flow_group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/RoverNavModels/html/group\_Flow\_group.html)

#### Keywords Doxygen groups

- [Rover](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/RoverNavModels/html/group_Rover_keyword_group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/RoverNavModels/html/group\_Rover\_keyword\_group.html)
- [Rover!Navigation!Sojourner](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/RoverNavModels/html/group_Rover_Navigation_Sojourner_keyword_group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/RoverNavModels/html/group\_Rover\_Navigation\_Sojourner\_keyword\_group.html)

#### Description

Model of autonomous rover navigation

Given the obstacle detection camera view, this model navigates the rover to a given goal. This model was built to interface to the WheelDriveMotion and WheelSteerDriveMotion model which provides computes the motion needed to complete the motion commanded by this model.

from the {RoverNavModels\_module\_uri}[RoverNavModels] models library for the xref:Dshellpp\_module simulation framework.

#### Model RoverPosNavigationFsm class details

For more information on the members and functions of this model class, please see [RoverNavModels::RoverPosNavigationFsm model class doxygen documentation](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/RoverNavModels/html/group__RoverPosNavigationFsm_group.html)  
([https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/RoverNavModels/html/group\\_\\_RoverPosNavigationFsm\\_group.html](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/RoverNavModels/html/group__RoverPosNavigationFsm_group.html))

## Parameters

Name	Type	Size	Quantity	Units	Description
chassis_motion_frame_uuid	unsigned int (uint)				The UUID for the frame located at the chassis frame but oriented so X axis is along desired linear velocity.
ck_mode	int				A flag to indicate if using CK mode (=1) or not (=0).
large_turn_radius	double		Length		large turning radius
look_move_direction	double	2	Dimensionless		x/y direction which camera looks: unit length
medium_pos_err	double		Length		medium position error
medium_rot_err	double		Angle		medium rotation error
nominal_angular_velocity	double		AngularVelocity		Nominal angular velocity to use
nominal_linear_velocity	double		Velocity		Nominal linear speed to use
nominal_rotation	double		Angle		nominal rotation
nominal_trans	double		Length		nominal straight motion (25 cm)
obstacle_avoidance_motion_counter	double		Dimensionless		The number of cycles to continue turning or driving in an obstacle avoidance turn or drive.
obstacle_avoidance_motion_timeout	double		Dimensionless		The number of cycles before timing out on a motion state.
small_pos_err	double		Length		small position error
small_rot_err	double		Angle		small rotation error
small_turn_radius	double		Length		small turning radius
turn_center_frame_uuid	unsigned int (uint)				The UUID for the frame located at the arc turn center.
vehicle_type	int				Vehicle type: -1 = skid steered, 0 = partially steered, 1 = fully-steered
very_small_pos_err	double		Length		very small position error
very_small_rot_err	double		Angle		very_small rotation error

## Flow Ins

Name	Type	Size	Quantity	Units	Description
ObstacleC	int				center obstacle if > 0
ObstacleL	int				left side obstacle if > 0
ObstacleR	int				right side obstacle if > 0
desired_x_y_h	double	3	Mixed		desired xy,h position

Name	Type	Size	Quantity	Units	Description
driveStatus	int				status of rover (0 → rover still competing previous drive command, 1 → rover ready for new command)
enable	int				List of flags to enable generation of new trajectory. This allows blocking of new trajectory generation to allow a variety of actions (all 1's → enable new trajectory, 0 → disable new trajectory)
root2localization_f2f_uuid	unsigned long (ulong)				Root to localization frame location frame2frame UUID.
steerStatus	int				status of rover (0 → rover still competing previous steer command, 1 → rover may still be completing drive command)
x_y_h	double	3	Mixed		current x, y, h position

#### Flow Outs

Name	Type	Size	Quantity	Units	Description
angle	double		Angle		angle to traverse when turning in place
angular_velocity	double		AngularVelocity		angular velocity for arc motion
at_goal	int				Flag for whether the rover has reached the goal location (not heading) (true/1 if at goal, false/0 otherwise)
at_heading	int				Flag for whether the rover has achieved the correct heading (true/1 if at heading, false/0 otherwise)
chassis_motion2turn_center_f2f_uuid	unsigned long (ulong)				Chassis motion to turn center frame location frame2frame UUID.
direction	double		Dimensionless		The direction to drive in for vehicles that are capable to - i.e. all wheel steered. =0
distance	double		Length		distance to travel (when moving forward/backward)
linear_velocity	double		Velocity		linear velocity for arc motion
localization2chassis_motion_f2f_uuid	unsigned long (ulong)				Localization to chassis motion frame location frame2frame UUID.
mode	int				0 → turn in place, 1 → move forward, -1 → move reverse
nav_drive_state	int				Flag for indicating is wheels can drive (true/1 if steering completed, false/0 otherwise)

Name	Type	Size	Quantity	Units	Description
radius	double		Length		turning radius (0 → straight, positive → left turn, negative → right turn)
speed	double		Dimensionless		0 → minimum speed, 1 → maximum speed
x_y_h_out	double	3	Mixed		For CK mode, next x, y, h output

### 31.12. RoverNavModels::RoverTurningRadiusWayPointsNav Flow Model



This model documentation is auto-generated. Documentation updates should be made to the model's `mdl` file to avoid the changes being lost.

[Doxygen link to all RoverNavModels models of Flow class](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/RoverNavModels/html/group_Flow_group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/RoverNavModels/html/group\_Flow\_group.html)

#### Keywords Doxygen groups

- [Rover](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/RoverNavModels/html/group_Rover_keyword_group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/RoverNavModels/html/group\_Rover\_keyword\_group.html)
- [Rover!Navigation](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/RoverNavModels/html/group_Rover_Navigation_keyword_group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/RoverNavModels/html/group\_Rover\_Navigation\_keyword\_group.html)

#### Description

Model to generate waypoints for non-zero turning radius vehicles

For vehicles with non-zero turning radius, this model generated way points that provide a path that is compatible with the minimum turning radius for getting the vehicle to the desired goal.

from the {RoverNavModels\_module\_uri}[RoverNavModels] models library for the xref:Dshellpp\_module simulation framework.

#### Model RoverTurningRadiusWayPointsNav class details

For more information on the members and functions of this model class, please see [RoverNavModels::RoverTurningRadiusWayPointsNav model class doxygen documentation](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/RoverNavModels/html/group_RoverTurningRadiusWayPointsNav_group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/RoverNavModels/html/group\_RoverTurningRadiusWayPointsNav\_group.html)

#### Parameters

Name	Type	Size	Quantity	Units	Description
h_waypoint_tol	double		Angle		Waypoint heading error threshold to use when switching to next waypoint value as goal
min_approach_offset	double		Length		The minimum approach distance to goal.
min_turning_radius	double		Length		The minimum turning radius for the vehicle
xy_waypoint_tol	double		Length		Waypoint xy error threshold to use when switching to next waypoint value as goal

#### Discrete States

Name	Type	Size	Quantity	Units	Description
num_waypoints	unsigned int (uint)				The number of current waypoints.
waypoint_index	unsigned int (uint)				The current active waypoint index.
waypoints	double	9	Mixed		Buffer for the list of waypoints (3 times number of waypoints)

## Flow Ins

Name	Type	Size	Quantity	Units	Description
desired_x_y_h	double	3	Mixed		desired xy,h position
x_y_h	double	3	Mixed		current xy,h position

## Flow Outs

Name	Type	Size	Quantity	Units	Description
waypoint_x_y_h	double	3	Mixed		next waypoint xy,h position

## 31.13. RoverNavModels::RoverVelNavigation Flow Model



This model documentation is auto-generated. Documentation updates should be made to the model's `mdl` file to avoid the changes being lost.

[Doxygen link to all RoverNavModels models of Flow class](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/RoverNavModels/html/group__Flow__group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/RoverNavModels/html/group\_\_Flow\_\_group.html)

### Keywords Doxygen groups

- [Rover](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/RoverNavModels/html/group__Rover__keyword__group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/RoverNavModels/html/group\_\_Rover\_\_keyword\_\_group.html)
- [Rover!Navigation!Velocity](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/RoverNavModels/html/group__Rover__Navigation__Velocity__keyword__group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/RoverNavModels/html/group\_\_Rover\_\_Navigation\_\_Velocity\_\_keyword\_\_group.html)

### Description

Model of autonomous rover navigation

Given the obstacle detection camera view, this model navigates the rover to a given goal using velocity commands instead of position drive commands as is used by RoverPosNavigation. This model was built to interface to the WheelDriveVelocity and WheelSteerDriveVelocity model which provides computes the motion needed to complete the motion commanded by this model.

from the {RoverNavModels\_module\_uri}[RoverNavModels] models library for the xref:Dshellpp\_module simulation framework.

### Model RoverVelNavigation class details

For more information on the members and functions of this model class, please see [RoverNavModels::RoverVelNavigation model class doxygen documentation](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/RoverNavModels/html/group__RoverVelNavigation__group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/RoverNavModels/html/group\_\_RoverVelNavigation\_\_group.html)

### Enums

#### Obstacle Enum

In the RoverVelNavigation model definition, the original enum Obstacle is defined as:

```
enum Obstacle
{
    None = 0,
    Left = 1,
    Center = 2,
    Right = 3,
    LeftCenter = 4,
    LeftRight = 5,
    CenterRight = 6,
    All = 7
};
```

In the C model code, these enum values can be accessed by using the follow C enum definitions:

```
/// RoverVelNavigationObstacle enum
enum RoverVelNavigationObstacle
{
    ROVER_VEL_NAVIGATION_OBSTACLE_NONE = 0,
    ROVER_VEL_NAVIGATION_OBSTACLE_LEFT = 1,
    ROVER_VEL_NAVIGATION_OBSTACLE_CENTER = 2,
    ROVER_VEL_NAVIGATION_OBSTACLE_RIGHT = 3,
    ROVER_VEL_NAVIGATION_OBSTACLE_LEFT_CENTER = 4,
    ROVER_VEL_NAVIGATION_OBSTACLE_LEFT_RIGHT = 5,
    ROVER_VEL_NAVIGATION_OBSTACLE_CENTER_RIGHT = 6,
    ROVER_VEL_NAVIGATION_OBSTACLE_ALL = 7
};
```

#### Mode Enum

In the RoverVelNavigation model definition, the original enum Mode is defined as:



```
enum Mode
{
    Disabled = 0,
    Enabled = 1,
    ApproachGoal = 2,
    TurnToGoal = 3,
    DriveToGoal = 4,
    ApproachHeading = 5,
    TurnToHeading = 6,
    ObstacleTurn = 7,
    ObstacleDriveForward = 8,
    ObstacleDriveBackward = 9,
    UndefinedState = 10
};
```

In the C model code, these enum values can be accessed by using the follow C enum definitions:

```
/// RoverVelNavigationMode enum
enum RoverVelNavigationMode
{
    ROVER_VEL_NAVIGATION_MODE_DISABLED = 0,
    ROVER_VEL_NAVIGATION_MODE_ENABLED = 1,
    ROVER_VEL_NAVIGATION_MODE_APPROACH_GOAL = 2,
    ROVER_VEL_NAVIGATION_MODE_TURN_TO_GOAL = 3,
    ROVER_VEL_NAVIGATION_MODE_DRIVE_TO_GOAL = 4,
    ROVER_VEL_NAVIGATION_MODE_APPROACH_HEADING = 5,
    ROVER_VEL_NAVIGATION_MODE_TURN_TO_HEADING = 6,
    ROVER_VEL_NAVIGATION_MODE_OBSTACLE_TURN = 7,
    ROVER_VEL_NAVIGATION_MODE_OBSTACLE_DRIVE_FORWARD = 8,
    ROVER_VEL_NAVIGATION_MODE_OBSTACLE_DRIVE_BACKWARD = 9,
    ROVER_VEL_NAVIGATION_MODE_UNDEFINED_STATE = 10
};
```

## Parameters

Name	Type	Size	Quantity	Units	Description
CK_obstacle_avoid_drive_steps	int				Number of steps to continue driving to avoid an obstacle
angular_velocity	double		AngularVelocity		Rotational speed to drive at
chassis_motion_frame_uuid	unsigned int (uint)				The UUID for the frame located at the vehicle frame and defining the desired instantaneous chassis spatial velocity.
ck_mode	int				A flag to indicate if using CK mode (=1) or not (=0).
drive_pos_err	double		Length		position error between vehicle location and goal location during drive
goal_rot_err	double		Angle		rotation error between vehicle heading and goal drive direction
heading_rot_err	double		Angle		rotation error between vehicle heading and goal drive direction
linear_velocity	double		Velocity		Linear speed to drive at
look_move_direction	double	2	Dimensionless		x/y direction which camera looks: unit length
obstacle_avoid_drive_steps	int				Number of steps to continue driving to avoid an obstacle
obstacle_avoid_on	int				Flag to indicate use of obstacle avoidance: 0 don't use, 1 use obstacle avoidance

Name	Type	Size	Quantity	Units	Description
obstacle_avoid_turn_steps	int				Number of steps to continue turning to avoid an obstacle
small_goal_rot_err	double		Angle		small rotation error between vehicle heading and goal drive direction
threshold_scale	double		Dimensionless		Threshold parameter. When exceeded start setting outputs at max value
turn_center_frame_uuid	unsigned int (uint)				The UUID for the frame located at the arc turn center.
turn_pos_err	double		Length		position error between vehicle location and goal location during turn

#### Discrete States

Name	Type	Size	Quantity	Units	Description
curr_obstacle	Obstacle				Current obstacle location
nextState	Mode				The next statechart state to transition to
obstacle_avoid_drive_steps_completed	int				Number of steps to driven to avoid an obstacle
obstacle_avoid_turn_steps_completed	int				Number of steps to turned to avoid an obstacle
turn_direction	int				Turn direction: left = -1, no turn = 0, right = 1

#### Flow Ins

Name	Type	Size	Quantity	Units	Description
ObstacleC	int				center obstacle if > 0
ObstacleL	int				left side obstacle if > 0
ObstacleR	int				right side obstacle if > 0
desired_x_y_h	double	3	Mixed		desired xy,h position
enable	int				List of flags to enable generation of new trajectory. This allows blocking of new trajectory generation to allow a variety of actions (all 1's→enable new trajectory, 0→disable new trajectory)
root2localization_f2f_uuid	unsigned long (ulong)				Root to localization frame location frame2frame UUID.
x_y_h	double	3	Mixed		current xy,h position

#### Flow Outs

Name	Type	Size	Quantity	Units	Description
chassis_motion2turn_center_f2f_uuid	unsigned long (ulong)				Chassis motion to turn center frame location frame2frame UUID.

Name	Type	Size	Quantity	Units	Description
desired_x_y_h_rate	double	3	Mixed		Desired chassis x,y,h rates
localization2chassis_motion_f2f_uuid	unsigned long (ulong)				Localization to chassis motion frame location frame2frame UUID.
x_y_h_out	double	3	Mixed		next x,y,h position in CK mode

### 31.14. RoverNavModels::RoverVelNavigationFsm Flow Model



This model documentation is auto-generated. Documentation updates should be made to the model's `mdl` file to avoid the changes being lost.

[Doxygen link to all RoverNavModels models of Flow class](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/RoverNavModels/html/group_Flow_group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/RoverNavModels/html/group\_Flow\_group.html)

#### Keywords Doxygen groups

- [Rover](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/RoverNavModels/html/group_Rover_keyword_group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/RoverNavModels/html/group\_Rover\_keyword\_group.html)
- [Rover!Navigation!Velocity](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/RoverNavModels/html/group_Rover_Navigation_Velocity_keyword_group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/RoverNavModels/html/group\_Rover\_Navigation\_Velocity\_keyword\_group.html)

#### Description

Model of autonomous rover navigation

Given the obstacle detection camera view, this model navigates the rover to a given goal using velocity commands instead of position drive commands as is used by RoverPosNavigation. This model was built to interface to the WheelDriveVelocity and WheelSteerDriveVelocity model which provides computes the motion needed to complete the motion commanded by this model.

from the {RoverNavModels\_module\_uri}[RoverNavModels] models library for the xref:Dshellpp\_module simulation framework.

#### Model RoverVelNavigationFsm class details

For more information on the members and functions of this model class, please see [RoverNavModels::RoverVelNavigationFsm model class doxygen documentation](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/RoverNavModels/html/group_RoverVelNavigationFsm_group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/RoverNavModels/html/group\_RoverVelNavigationFsm\_group.html)

#### Parameters

Name	Type	Size	Quantity	Units	Description
CK_obstacle_avoid_drive_steps	int				Number of steps to continue driving to avoid an obstacle
angular_velocity	double		AngularVelocity		Rotational speed to drive at
chassis_motion_frame_uuid	unsigned int (uint)				The UUID for the frame located at the vehicle frame and defining the desired instantaneous chassis spatial velocity.
ck_mode	int				A flag to indicate if using CK mode (=1) or not (=0).
drive_pos_err	double		Length		position error between vehicle location and goal location during drive
goal_rot_err	double		Angle		rotation error between vehicle heading and goal drive direction
heading_rot_err	double		Angle		rotation error between vehicle heading and goal drive direction
idle_angular_velocity	double		AngularVelocity		Forward 'in-gear' angular velocity of vehicle in idle

Name	Type	Size	Quantity	Units	Description
idle_drive_velocity	double		Velocity		Forward 'in-gear' driving velocity of vehicle in idle
linear_velocity	double		Velocity		Linear speed to drive at
look_move_direction	double	2	Dimensionless		x/y direction which camera looks: unit length
no_backup	bool				Disable backing up maneuvers for the vehicle
obstacle_avoid_drive_steps	int				Number of steps to continue driving to avoid an obstacle
obstacle_avoid_on	int				Flag to indicate use of obstacle avoidance: 0 don't use, 1 use obstacle avoidance
obstacle_avoid_turn_steps	int				Number of steps to continue turning to avoid an obstacle
rollover_threshold	double		Acceleration		Upper limit on the product of desired angular and linear velocity
small_goal_rot_err	double		Angle		small rotation error between vehicle heading and goal drive direction
threshold_scale	double		Dimensionless		Threshold parameter. When exceeded start setting outputs at max value
turn_center_frame_uuid	unsigned int (uint)				The UUID for the frame located at the arc turn center.
turn_pos_err	double		Length		position error between vehicle location and goal location during turn

#### Flow Ins

Name	Type	Size	Quantity	Units	Description
ObstacleC	int				center obstacle if > 0
ObstacleL	int				left side obstacle if > 0
ObstacleR	int				right side obstacle if > 0
desired_x_y_h	double	3	Mixed		desired xy,h position
enable	int				List of flags to enable generation of new trajectory. This allows blocking of new trajectory generation to allow a variety of actions (all 1's→enable new trajectory, 0→disable new trajectory)
root2localization_f2f_uuid	unsigned long (ulong)				Root to localization frame location frame2frame UUID.
x_y_h	double	3	Mixed		current xy,h position

#### Flow Outs

Name	Type	Size	Quantity	Units	Description
------	------	------	----------	-------	-------------

Name	Type	Size	Quantity	Units	Description
chassis_motion2turn_center_f2f_uuid	unsigned long (ulong)				Chassis motion to turn center frame location frame2frame UUID.
desired_x_y_h_rate	double	3	Mixed		Desired chassis x,y,h rates
localization2chassis_motion_f2f_uuid	unsigned long (ulong)				Localization to chassis motion frame location frame2frame UUID.
x_y_h_out	double	3	Mixed		next x,y,h position in CK mode

### 31.15. RoverNavModels::SimpleArcPlanner Flow Model



This model documentation is auto-generated. Documentation updates should be made to the model's `mdl` file to avoid the changes being lost.

[Doxygen link to all RoverNavModels models of Flow class](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/RoverNavModels/html/group__Flow__group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/RoverNavModels/html/group\_\_Flow\_\_group.html)

#### Keywords Doxygen groups

- [Rover](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/RoverNavModels/html/group__Rover__keyword__group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/RoverNavModels/html/group\_\_Rover\_\_keyword\_\_group.html)
- [Rover!Arc!Planner](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/RoverNavModels/html/group__Rover__Arc__Planner__keyword__group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/RoverNavModels/html/group\_\_Rover\_\_Arc\_\_Planner\_\_keyword\_\_group.html)

#### Description

Model of arc planner for vehicles with no turn in place capability

Rover will check a series of arcs for potential obstacles and choose the arc that is clear of obstacles and gets the orver closest to the goal

from the {RoverNavModels\_module\_uri}[RoverNavModels] models library for the xref:Dshellpp\_module simulation framework.

#### Model SimpleArcPlanner class details

For more information on the members and functions of this model class, please see [RoverNavModels::SimpleArcPlanner model class doxygen documentation](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/RoverNavModels/html/group__SimpleArcPlanner__group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/RoverNavModels/html/group\_\_SimpleArcPlanner\_\_group.html)

#### Enums

##### Mode Enum

In the SimpleArcPlanner model definition, the original enum Mode is defined as:

```
enum Mode
{
    CalculateArc = 0,
    PerformingArc = 1,
    GoalReached = 2,
    BackUp = 3
};
```

In the C model code, these enum values can be accessed by using the follow C enum definitions:

```
/// SimpleArcPlannerMode enum
enum SimpleArcPlannerMode
{
    SIMPLE_ARC_PLANNER_MODE_CALCULATE_ARC = 0,
    SIMPLE_ARC_PLANNER_MODE_PERFORMING_ARC = 1,
    SIMPLE_ARC_PLANNER_MODE_GOAL_REACHED = 2,
    SIMPLE_ARC_PLANNER_MODE_BACK_UP = 3
};
```

#### Parameters

Name	Type	Size	Quantity	Units	Description
arc_resolution	double		Time		Time step used to propagate arcs
arc_steps	int				How far arc should be expanded and checked

Name	Type	Size	Quantity	Units	Description
bounding_box_x	double		Length		X bounds for obstacle detection
bounding_box_y	double		Length		Y bounds for obstacle detection
demSpecName	string				The name of the top level DVar spec with the vehicle's TopoDem pointer.
goal_error	double		Length		Error for determines when goal has been reached
nominal_angular_velocity	double		AngularVelocity		Nominal angular speed to use
nominal_linear_velocity	double		Velocity		Nominal linear speed to use
num_arcs	int				Number of arcs to check
obstacle_height	double		Length		Height for an obstacle
recompute_steps	int				Number of steps before arc should be recomputed

#### Discrete States

Name	Type	Size	Quantity	Units	Description
nSteps	int				Number of steps taken from last arc compute
navState	int				State logic for navigation

#### Flow Ins

Name	Type	Size	Quantity	Units	Description
chassis_motion2turn_center_f2f_uuid	unsigned long (ulong)				Chassis motion to turn center frame location frame2frame UUID.
desired_x_y_h	double	3	Mixed		desired xy,h position
enable	int				List of flags to enable generation of new trajectory. This allows blocking of new trajectory generation to allow a variety of actions (all 1's→enable new trajectory, 0→disable new trajectory)
x_y_h	double	3	Mixed		current xy,h position

#### Flow Outs

Name	Type	Size	Quantity	Units	Description
desired_x_y_h_rate	double	3	Mixed		desired xy,h rate for arc motion

### 31.16. RoverNavModels::SimpleArcPlannerFsm Flow Model



This model documentation is auto-generated. Documentation updates should be made to the model's `mdl` file to avoid the changes being lost.

[Doxygen link to all RoverNavModels models of Flow class](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/RoverNavModels/html/group_Flow_group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/RoverNavModels/html/group\_Flow\_group.html)

Keywords Doxygen groups

- [Rover](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/RoverNavModels/html/group__Rover__keyword__group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/RoverNavModels/html/group\_\_Rover\_\_keyword\_\_group.html)
- [Rover!Arc!Planner](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/RoverNavModels/html/group__Rover__Arc__Planner__keyword__group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/RoverNavModels/html/group\_\_Rover\_\_Arc\_\_Planner\_\_keyword\_\_group.html)

### Description

Model of arc planner for vehicles with no turn in place capability

Rover will check a series of arcs for potential obstacles and choose the arc that is clear of obstacles and gets the orver closest to the goal

from the {RoverNavModels\_module\_uri}[RoverNavModels] models library for the xref:Dshellpp\_module simulation framework.

### Model SimpleArcPlannerFsm class details

For more information on the members and functions of this model class, please see [RoverNavModels::SimpleArcPlannerFsm model class doxygen documentation](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/RoverNavModels/html/group__SimpleArcPlannerFsm__group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/RoverNavModels/html/group\_\_SimpleArcPlannerFsm\_\_group.html)

### Enums

#### Mode Enum

In the SimpleArcPlannerFsm model definition, the original enum Mode is defined as:

```
enum Mode
{
    CalculateArc = 0,
    PerformingArc = 1,
    GoalReached = 2,
    BackUp = 3
};
```

In the C model code, these enum values can be accessed by using the follow C enum definitions:

```
/// SimpleArcPlannerFsmMode enum
enum SimpleArcPlannerFsmMode
{
    SIMPLE_ARC_PLANNER_FSM_MODE_CALCULATE_ARC = 0,
    SIMPLE_ARC_PLANNER_FSM_MODE_PERFORMING_ARC = 1,
    SIMPLE_ARC_PLANNER_FSM_MODE_GOAL_REACHED = 2,
    SIMPLE_ARC_PLANNER_FSM_MODE_BACK_UP = 3
};
```

### Parameters

Name	Type	Size	Quantity	Units	Description
arc_resolution	double		Time		Time step used to propagate arcs
arc_steps	int				How far arc should be expanded and checked
bounding_box_x	double		Length		X bounds for obstacle detection
bounding_box_y	double		Length		Y bounds for obstacle detection
demSpecName	string				The name of the top level DVar spec with the vehicle's TopoDem pointer.
goal_error	double		Length		Error for determines when goal has been reached
nominal_angular_velocity	double		AngularVelocity		Nominal angular speed to use
nominal_linear_velocity	double		Velocity		Nominal linear speed to use
num_arcs	int				Number of arcs to check
obstacle_height	double		Length		Height for an obstacle
recompute_steps	int				Number of steps before arc should be recomputed

### Discrete States

Name	Type	Size	Quantity	Units	Description
nSteps	int				Number of steps taken from last arc compute
navState	int				State logic for navigation

#### Flow Ins

Name	Type	Size	Quantity	Units	Description
chassis_motion2turn_center_f2f_uuid	unsigned long (ulong)				Chassis motion to turn center frame location frame2frame UUID.
desired_x_y_h	double	3	Mixed		desired xy,h position
enable	int				List of flags to enable generation of new trajectory. This allows blocking of new trajectory generation to allow a variety of actions (all 1's→enable new trajectory, 0→disable new trajectory)
x_y_h	double	3	Mixed		current xy,h position

#### Flow Outs

Name	Type	Size	Quantity	Units	Description
desired_x_y_h_rate	double	3	Mixed		desired xy,h rate for arc motion

### 31.17. RoverNavModels::Steering2W Actuator Model



This model documentation is auto-generated. Documentation updates should be made to the model's `mdl` file to avoid the changes being lost.

[Doxygen link to all RoverNavModels models of Actuator class](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/RoverNavModels/html/group__Actuator_group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/RoverNavModels/html/group\_\_Actuator\_group.html)

#### Keywords Doxygen groups

- [Rover](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/RoverNavModels/html/group__Rover_keyword_group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/RoverNavModels/html/group\_\_Rover\_keyword\_group.html)
- [Rover!2](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/RoverNavModels/html/group__Rover_2_keyword_group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/RoverNavModels/html/group\_\_Rover\_2\_keyword\_group.html)
- [Rover!Locomotion](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/RoverNavModels/html/group__Rover_Locomotion_keyword_group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/RoverNavModels/html/group\_\_Rover\_Locomotion\_keyword\_group.html)
- [wheel](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/RoverNavModels/html/group__wheel_keyword_group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/RoverNavModels/html/group\_\_wheel\_keyword\_group.html)

#### Description

Model to command steering angle for 2 wheel steering rovers. Rover has 6 wheels in a general configuration.

from the {RoverNavModels\_module\_uri}[RoverNavModels] models library for the xref:Dshellpp\_module simulation framework.

#### Model Steering2W class details

For more information on the members and functions of this model class, please see [RoverNavModels::Steering2W model class doxygen documentation](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/RoverNavModels/html/group__Steering2W_group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/RoverNavModels/html/group\_\_Steering2W\_group.html)

#### Parameters

Name	Type	Size	Quantity	Units	Description
epsilon	double		Angle		tolerance to determine if steering motion is complete



Name	Type	Size	Quantity	Units	Description
steeringSide	int				+1 → front wheel steering, -1 → rear wheel steering (assuming forward motion)
wheelCenterBias	double		Length		Bias of center wheel (positive implies closer to front side wheels, zero implies in center of front and back wheels)
wheelFrontBackDist	double		Length		Distance between front and back side wheels
wheelLeftRightDist	double		Length		Distance between left and right side wheels

#### Flow Ins

Name	Type	Size	Quantity	Units	Description
angle	double		Angle		angle to tranverse when turning in place
mode	int				mode (0 → turn in place, 1 → move forward, -1 → move reverse)
radius	double		Length		turning radius (0 → straight, positive → left turn, negative → right turn)
steerLeftIn	double		Angle		Current left steering angle
steerRightIn	double		Angle		Current right steering angle

#### Flow Outs

Name	Type	Size	Quantity	Units	Description
steerLeft	double		Angle		Desired steering angle for left side wheel
steerRight	double		Angle		Desired steering angle for right side wheel
steerStatus	double		Flag		Steering status (0 → still steering to desired values, 1 → done steering)

### 31.18. RoverNavModels::Steering2WFlow Flow Model



This model documentation is auto-generated. Documentation updates should be made to the model's `mdl` file to avoid the changes being lost.

[Doxygen link to all RoverNavModels models of Flow class](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/RoverNavModels/html/group__Flow__group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/RoverNavModels/html/group\_\_Flow\_\_group.html)

#### Keywords Doxygen groups

- [Rover](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/RoverNavModels/html/group__Rover__keyword__group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/RoverNavModels/html/group\_\_Rover\_\_keyword\_\_group.html)
- [Rover!2](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/RoverNavModels/html/group__Rover__2__keyword__group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/RoverNavModels/html/group\_\_Rover\_\_2\_\_keyword\_\_group.html)
- [Rover!Locomotion](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/RoverNavModels/html/group__Rover__Locomotion__keyword__group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/RoverNavModels/html/group\_\_Rover\_\_Locomotion\_\_keyword\_\_group.html)
- [wheel](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/RoverNavModels/html/group__wheel__keyword__group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/RoverNavModels/html/group\_\_wheel\_\_keyword\_\_group.html)

#### Description

Flow model to command steering angle for 2 wheel steering rovers. Rover has 6 wheels in a general configuration.

from the {RoverNavModels\_module\_uri}[RoverNavModels] models library for the xref:Dshellpp\_module simulation framework.

### Model Steering2WFlow class details

For more information on the members and functions of this model class, please see [RoverNavModels::Steering2WFlow model class doxygen documentation](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/RoverNavModels/html/group_Steering2WFlow_group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/RoverNavModels/html/group\_Steering2WFlow\_group.html)

### Parameters

Name	Type	Size	Quantity	Units	Description
steeringSide	int				+1 → front wheel steering. -1 → rear wheel steering (assuming forward motion)
wheelCenterBias	double		Length		Bias of center wheel (positive implies closer to front side wheels, zero implies in center of front and back wheels)
wheelFrontRearDist	double		Length		Distance between front and back side wheels
wheelLeftRightDist	double		Length		Distance between left and right side wheels

### Flow Ins

Name	Type	Size	Quantity	Units	Description
angle	double		Angle		angle to transverse when turning in place
mode	int				mode (0 → turn in place, 1 → move forward, -1 → move reverse)
radius	double		Length		turning radius (0 → straight, positive → left turn, negative → right turn)

### Flow Outs

Name	Type	Size	Quantity	Units	Description
steerRockerSideLeft	double		Angle		Desired steering angle for left side wheel
steerRockerSideRight	double		Angle		Desired steering angle for right side wheel

## 31.19. RoverNavModels::Steering4W Actuator Model



This model documentation is auto-generated. Documentation updates should be made to the model's `mdl` file to avoid the changes being lost.

[Doxygen link to all RoverNavModels models of Actuator class](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/RoverNavModels/html/group_Actuator_group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/RoverNavModels/html/group\_Actuator\_group.html)

### Keywords Doxygen groups

- [Rover](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/RoverNavModels/html/group_Rover_keyword_group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/RoverNavModels/html/group\_Rover\_keyword\_group.html)
- [Rover!4](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/RoverNavModels/html/group_Rover_4_keyword_group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/RoverNavModels/html/group\_Rover\_4\_keyword\_group.html)
- [Rover!Locomotion](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/RoverNavModels/html/group_Rover_Locomotion_keyword_group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/RoverNavModels/html/group\_Rover\_Locomotion\_keyword\_group.html)
- [wheel](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/RoverNavModels/html/group_wheel_keyword_group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/RoverNavModels/html/group\_wheel\_keyword\_group.html)

### Description

Model to command steering angles for rovers with 4 wheel steering. Center wheels (if any) are not steered. A line through the center wheels (real or phantom) goes through the center of steering.

from the {RoverNavModels\_module\_uri}[RoverNavModels] models library for the xref:Dshellpp\_module simulation framework.

### Model Steering4W class details

For more information on the members and functions of this model class, please see [RoverNavModels::Steering4W model class doxygen documentation](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/RoverNavModels/html/group_Steering4W_group.html) ([https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/RoverNavModels/html/group\\_Steering4W\\_group.html](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/RoverNavModels/html/group_Steering4W_group.html))

### Parameters

Name	Type	Size	Quantity	Units	Description
epsilon	double		Angle		tolerance to determine if steering motion is complete
wheelCenterBias	double		Length		Bias of center wheel axis or center line (positive implies closer to front side wheels, zero implies in center of front and back wheels)
wheelFrontBackDist	double		Length		Distance between front and back side wheels
wheelLeftRightDist	double		Length		Distance between left and right side wheels

### Flow Ins

Name	Type	Size	Quantity	Units	Description
angle	double		Angle		angle to tranverse when turning in place
mode	double		Flag		mode (0 → turn in place, 1 → move forward, -1 → move reverse)
radius	double		Length		turning radius (0 → straight, positive → left turn, negative → right turn)
steerLeftBackIn	double		Angle		Current left rear steering angle
steerLeftFrontIn	double		Angle		Current left front steering angle
steerRightBackIn	double		Angle		Current right rear steering angle
steerRightFrontIn	double		Angle		Current right front steering angle

### Flow Outs

Name	Type	Size	Quantity	Units	Description
steerLeftBack	double		Angle		Desired steering angle for left rear wheel
steerLeftFront	double		Angle		Desired steering angle for left front wheel
steerRightBack	double		Angle		Desired steering angle for right rear wheel
steerRightFront	double		Angle		Desired steering angle for right front wheel
steerStatus	double		Flag		Steering status (0 → still steering to desired values, 1 → done steering)

## 31.20. RoverNavModels::Steering4WFlow Flow Model



This model documentation is auto-generated. Documentation updates should be made to the model's `mdl` file to avoid the changes being lost.

[Doxygen link to all RoverNavModels models of Flow class](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/RoverNavModels/html/group_Flow_group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/RoverNavModels/html/group\_Flow\_group.html)

### Keywords Doxygen groups

- [Rover](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/RoverNavModels/html/group_Rover_keyword_group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/RoverNavModels/html/group\_Rover\_keyword\_group.html)
- [Rover!4](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/RoverNavModels/html/group_Rover_4_keyword_group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/RoverNavModels/html/group\_Rover\_4\_keyword\_group.html)
- [Rover!Locomotion](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/RoverNavModels/html/group_Rover_Locomotion_keyword_group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/RoverNavModels/html/group\_Rover\_Locomotion\_keyword\_group.html)
- [wheel](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/RoverNavModels/html/group_wheel_keyword_group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/RoverNavModels/html/group\_wheel\_keyword\_group.html)

### Description

Flow model to command steering angles for rovers with 4 wheel steering. Rover has wheels in a general configuration.

from the {RoverNavModels\_module\_uri}[RoverNavModels] models library for the xref:Dshellpp\_module simulation framework.

### Model Steering4WFlow class details

For more information on the members and functions of this model class, please see [RoverNavModels::Steering4WFlow model class doxygen documentation](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/RoverNavModels/html/group_Steering4WFlow_group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/RoverNavModels/html/group\_Steering4WFlow\_group.html)

### Parameters

Name	Type	Size	Quantity	Units	Description
bogeyFront	double		Flag		+1 → bogey side is front, -1 → rocker side is front
wheelCenterBias	double		Length		Bias of center wheel (positive implies closer to bogey side wheels, zero implies in center of front and back wheels)
wheelFrontRearDist	double		Length		Distance between front and back side wheels
wheelLeftRightDist	double		Length		Distance between left and right side wheels

### Flow Ins

Name	Type	Size	Quantity	Units	Description
angle	double		Angle		angle to transverse when turning in place
mode	int				mode (0 → turn in place, 1 → move forward, -1 → move reverse)
radius	double		Length		turning radius (0 → straight, positive → left turn, negative → right turn)

### Flow Outs

Name	Type	Size	Quantity	Units	Description
steerBogeySideLeft	double		Angle		Desired steering angle for left bogey wheel
steerBogeySideRight	double		Angle		Desired steering angle for right bogey wheel
steerRockerSideLeft	double		Angle		Desired steering angle for left rear wheel

Name	Type	Size	Quantity	Units	Description
steerRockerSideRight	double		Angle		Desired steering angle for right rear wheel

## 31.21. RoverNavModels::Steering6W Actuator Model



This model documentation is auto-generated. Documentation updates should be made to the model's `mdl` file to avoid the changes being lost.

[Doxygen link to all RoverNavModels models of Actuator class](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/RoverNavModels/html/group__Actuator_group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/RoverNavModels/html/group\_\_Actuator\_group.html)

### Keywords Doxygen groups

- [Rover](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/RoverNavModels/html/group__Rover_keyword_group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/RoverNavModels/html/group\_\_Rover\_keyword\_group.html)
- [Rover!6](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/RoverNavModels/html/group__Rover_6_keyword_group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/RoverNavModels/html/group\_\_Rover\_6\_keyword\_group.html)
- [Rover!Locomotion](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/RoverNavModels/html/group__Rover_Locomotion_keyword_group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/RoverNavModels/html/group\_\_Rover\_Locomotion\_keyword\_group.html)
- [wheel](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/RoverNavModels/html/group__wheel_keyword_group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/RoverNavModels/html/group\_\_wheel\_keyword\_group.html)

### Description

Model to command steering angles for rovers with 6 wheel steering. Rover has wheels in a general configuration.

from the {RoverNavModels\_module\_uri}{RoverNavModels} models library for the xref:Dshellpp\_module simulation framework.

### Model Steering6W class details

For more information on the members and functions of this model class, please see [RoverNavModels::Steering6W model class doxygen documentation](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/RoverNavModels/html/group__Steering6W_group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/RoverNavModels/html/group\_\_Steering6W\_group.html)

### Parameters

Name	Type	Size	Quantity	Units	Description
epsilon	double		Angle		tolerance to determine if steering motion is complete
wheelCenterBias	double		Length		Bias of center wheel (positive implies closer to front side wheels, zero implies in center of front and back wheels)
wheelFrontBackDist	double		Length		Distance between front and back side wheels
wheelLeftRightDist	double		Length		Distance between left and right side wheels

### Flow Ins

Name	Type	Size	Quantity	Units	Description
angle	double		Angle		angle to tranverse when turning in place
mode	double		Flag		mode (0 → turn in place, 1 → move forward, -1 → move reverse)
radius	double		Length		turning radius (0 → straight, positive → left turn, negative → right turn)
steerLeftBackIn	double		Angle		Current left rear steering angle
steerLeftCenterIn	double		Angle		Current left center steering angle

Name	Type	Size	Quantity	Units	Description
steerLeftFrontIn	double		Angle		Current left front steering angle
steerRightBackIn	double		Angle		Current right rear steering angle
steerRightCenterIn	double		Angle		Current right center steering angle
steerRightFrontIn	double		Angle		Current right front steering angle

#### Flow Outs

Name	Type	Size	Quantity	Units	Description
steerLeftBack	double		Angle		Desired steering angle for left rear wheel
steerLeftCenter	double		Angle		Desired steering angle for left center wheel
steerLeftFront	double		Angle		Desired steering angle for left front wheel
steerRightBack	double		Angle		Desired steering angle for right rear wheel
steerRightCenter	double		Angle		Desired steering angle for right center wheel
steerRightFront	double		Angle		Desired steering angle for right front wheel
steerStatus	double		Flag		Steering status (0 → still steering to desired values, 1 → done steering)

## 31.22. RoverNavModels::Steering6WFlow Flow Model



This model documentation is auto-generated. Documentation updates should be made to the model's `mdl` file to avoid the changes being lost.

[Doxygen link to all RoverNavModels models of Flow class](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/RoverNavModels/html/group_Flow_group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/RoverNavModels/html/group\_Flow\_group.html)

#### Keywords Doxygen groups

- [Rover](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/RoverNavModels/html/group_Rover_keyword_group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/RoverNavModels/html/group\_Rover\_keyword\_group.html)
- [Rover!6](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/RoverNavModels/html/group_Rover_6_keyword_group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/RoverNavModels/html/group\_Rover\_6\_keyword\_group.html)
- [Rover!Locomotion](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/RoverNavModels/html/group_Rover_Locomotion_keyword_group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/RoverNavModels/html/group\_Rover\_Locomotion\_keyword\_group.html)
- [wheel](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/RoverNavModels/html/group_wheel_keyword_group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/RoverNavModels/html/group\_wheel\_keyword\_group.html)

#### Description

Flow model to command steering angles for rovers with 6 wheel steering. Rover has wheels in a general configuration.

from the {RoverNavModels\_module\_uri}[RoverNavModels] models library for the xref:Dshellpp\_module simulation framework.

#### Model Steering6WFlow class details

For more information on the members and functions of this model class, please see [RoverNavModels::Steering6WFlow model class doxygen documentation](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/RoverNavModels/html/group_Steering6WFlow_group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/RoverNavModels/html/group\_Steering6WFlow\_group.html)

#### Parameters

Name	Type	Size	Quantity	Units	Description
------	------	------	----------	-------	-------------

Name	Type	Size	Quantity	Units	Description
bogeyFront	double		Flag		+1 → bogey is front side, -1 → rocker is front side
wheelCenterBias	double		Length		Bias of center wheel (positive implies closer to front side wheels, zero implies in center of front and back wheels)
wheelFrontRearDist	double		Length		Distance between front and back side wheels
wheelLeftRightDist	double		Length		Distance between left and right side wheels

#### Flow Ins

Name	Type	Size	Quantity	Units	Description
angle	double		Angle		angle to transverse when turning in place
mode	int				mode (0 → turn in place, 1 → move forward, -1 → move reverse)
radius	double		Length		turning radius (0 → straight, positive → left turn, negative → right turn)

#### Flow Outs

Name	Type	Size	Quantity	Units	Description
steerBogeySideLeft	double		Angle		Desired steering angle for left front wheel
steerBogeySideRight	double		Angle		Desired steering angle for right front wheel
steerCenterLeft	double		Angle		Desired steering angle for left center wheel
steerCenterRight	double		Angle		Desired steering angle for right center wheel
steerRockerSideLeft	double		Angle		Desired steering angle for left rear wheel
steerRockerSideRight	double		Angle		Desired steering angle for right rear wheel

### 31.23. RoverNavModels::SteeringStatus Flow Model



This model documentation is auto-generated. Documentation updates should be made to the model's `mdl` file to avoid the changes being lost.

[Doxygen link to all RoverNavModels models of Flow class](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/RoverNavModels/html/group_Flow_group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/RoverNavModels/html/group\_Flow\_group.html)

#### Keywords Doxygen groups

- [Rover](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/RoverNavModels/html/group_Rover_keyword_group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/RoverNavModels/html/group\_Rover\_keyword\_group.html)
- [Rover!Controller](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/RoverNavModels/html/group_Rover_Controller_keyword_group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/RoverNavModels/html/group\_Rover\_Controller\_keyword\_group.html)
- [Rover!Locomotion](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/RoverNavModels/html/group_Rover_Locomotion_keyword_group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/RoverNavModels/html/group\_Rover\_Locomotion\_keyword\_group.html)

#### Description

Model to determine if commanded steering angles have been achieved.

from the {RoverNavModels\_module\_uri}[RoverNavModels] models library for the xref:Dshellpp\_module simulation framework.

#### Model SteeringStatus class details

For more information on the members and functions of this model class, please see [RoverNavModels::SteeringStatus model class doxygen documentation](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/RoverNavModels/html/group_SteeringStatus_group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/RoverNavModels/html/group\_SteeringStatus\_group.html)

#### Parameters

Name	Type	Size	Quantity	Units	Description
dimension	int				Dimension of input vectors
epsilon	double		Angle		tolerance (radians) to determine if steering motion is complete

#### Flow Ins

Name	Type	Size	Quantity	Units	Description
steerAngle	double	*	Angle		Current steering angles
steerAngle_d	double	*	Angle		Desired steering angles

#### Flow Outs

Name	Type	Size	Quantity	Units	Description
steerStatus	int				Steering status (0 → still steering to desired values, 1 → done steering)

### 31.24. RoverNavModels::SwitchExample Flow Model



This model documentation is auto-generated. Documentation updates should be made to the model's mdl file to avoid the changes being lost.

[Doxygen link to all RoverNavModels models of Flow class](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/RoverNavModels/html/group_Flow_group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/RoverNavModels/html/group\_Flow\_group.html)

#### Keywords Doxygen groups

- [Rover](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/RoverNavModels/html/group_Rover_keyword_group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/RoverNavModels/html/group\_Rover\_keyword\_group.html)
- [Rover!Controller](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/RoverNavModels/html/group_Rover_Controller_keyword_group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/RoverNavModels/html/group\_Rover\_Controller\_keyword\_group.html)
- [Rover!Locomotion](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/RoverNavModels/html/group_Rover_Locomotion_keyword_group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/RoverNavModels/html/group\_Rover\_Locomotion\_keyword\_group.html)

#### Description

Example to show to use boost statechart.

from the {RoverNavModels\_module\_uri}[RoverNavModels] models library for the xref:Dshellpp\_module simulation framework.

#### Model SwitchExample class details

For more information on the members and functions of this model class, please see [RoverNavModels::SwitchExample model class doxygen documentation](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/RoverNavModels/html/group_SwitchExample_group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/RoverNavModels/html/group\_SwitchExample\_group.html)

#### Flow Ins

Name	Type	Size	Quantity	Units	Description
flip	int				Flip state

#### Flow Outs

Name	Type	Size	Quantity	Units	Description
state	double		Flag		Output state



## 31.25. RoverNavModels::SwitchMSMExample Flow Model



This model documentation is auto-generated. Documentation updates should be made to the model's `mdl` file to avoid the changes being lost.

[Doxygen link to all RoverNavModels models of Flow class](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/RoverNavModels/html/group_Flow_group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/RoverNavModels/html/group\_Flow\_group.html)

### Keywords Doxygen groups

- [Rover](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/RoverNavModels/html/group_Rover_keyword_group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/RoverNavModels/html/group\_Rover\_keyword\_group.html)
- [Rover!Controller](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/RoverNavModels/html/group_Rover_Controller_keyword_group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/RoverNavModels/html/group\_Rover\_Controller\_keyword\_group.html)
- [Rover!Locomotion](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/RoverNavModels/html/group_Rover_Locomotion_keyword_group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/RoverNavModels/html/group\_Rover\_Locomotion\_keyword\_group.html)

### Description

Example to show to use boost Meta state Machine (MSM).

from the {RoverNavModels\_module\_uri}[RoverNavModels] models library for the xref:Dshellpp\_module simulation framework.

### Model SwitchMSMExample class details

For more information on the members and functions of this model class, please see [RoverNavModels::SwitchMSMExample model class doxygen documentation](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/RoverNavModels/html/group_SwitchMSMExample_group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/RoverNavModels/html/group\_SwitchMSMExample\_group.html)

### Flow Ins

Name	Type	Size	Quantity	Units	Description
flip	int				Flip state

### Flow Outs

Name	Type	Size	Quantity	Units	Description
state	double		Flag		Output state

## 31.26. RoverNavModels::SwitchMSMFuncorExample Flow Model



This model documentation is auto-generated. Documentation updates should be made to the model's `mdl` file to avoid the changes being lost.

[Doxygen link to all RoverNavModels models of Flow class](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/RoverNavModels/html/group_Flow_group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/RoverNavModels/html/group\_Flow\_group.html)

### Keywords Doxygen groups

- [Rover](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/RoverNavModels/html/group_Rover_keyword_group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/RoverNavModels/html/group\_Rover\_keyword\_group.html)
- [Rover!Controller](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/RoverNavModels/html/group_Rover_Controller_keyword_group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/RoverNavModels/html/group\_Rover\_Controller\_keyword\_group.html)
- [Rover!Locomotion](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/RoverNavModels/html/group_Rover_Locomotion_keyword_group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/RoverNavModels/html/group\_Rover\_Locomotion\_keyword\_group.html)

### Description

Example to show to use boost Meta state Machine (MSM) specified using functors.

from the {RoverNavModels\_module\_uri}[RoverNavModels] models library for the xref:Dshellpp\_module simulation framework.

### Model SwitchMSMFuncorExample class details

For more information on the members and functions of this model class, please see [RoverNavModels::SwitchMSMFuncorExample model class doxygen documentation](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/RoverNavModels/html/group_SwitchMSMFuncorExample_group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/RoverNavModels/html/group\_SwitchMSMFuncorExample\_group.html)

### Flow Ins

Name	Type	Size	Quantity	Units	Description
flip	int				Flip state

## Flow Outs

Name	Type	Size	Quantity	Units	Description
state	double		Flag		Output state

### 31.27. RoverNavModels::WheelDriveMotion Flow Model



This model documentation is auto-generated. Documentation updates should be made to the model's `mdl` file to avoid the changes being lost.

[Doxygen link to all RoverNavModels models of Flow class](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/RoverNavModels/html/group__Flow_group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/RoverNavModels/html/group\_\_Flow\_group.html)

#### Keywords Doxygen groups

- [Rover](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/RoverNavModels/html/group__Rover_keyword_group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/RoverNavModels/html/group\_\_Rover\_keyword\_group.html)
- [Rover!Locomotion](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/RoverNavModels/html/group__Rover_Locomotion_keyword_group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/RoverNavModels/html/group\_\_Rover\_Locomotion\_keyword\_group.html)
- [Rover!Odometry](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/RoverNavModels/html/group__Rover_Odometry_keyword_group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/RoverNavModels/html/group\_\_Rover\_Odometry\_keyword\_group.html)
- [Rover!n-wheel](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/RoverNavModels/html/group__Rover_n-wheel_keyword_group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/RoverNavModels/html/group\_\_Rover\_n-wheel\_keyword\_group.html)

#### Description

Model of rover wheel kinematics for a non-steered wheel on a rover.

This model computes the drive distance this wheel should drive to traverse the arc it will follow when the vehicle drives along a circular arc specified by the flow-in desired vehicle motion.

from the {RoverNavModels\_module\_uri}[RoverNavModels] models library for the xref:Dshellpp\_module simulation framework.

#### Model WheelDriveMotion class details

For more information on the members and functions of this model class, please see [RoverNavModels::WheelDriveMotion model class doxygen documentation](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/RoverNavModels/html/group__WheelDriveMotion_group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/RoverNavModels/html/group\_\_WheelDriveMotion\_group.html)

#### Parameters

Name	Type	Size	Quantity	Units	Description
direction	double		Flag		+1 → rover moves towards bogey when wheels rotate positive. -1 → rover moves towards rocker when wheels rotate positive
localization_frame_uuid	unsigned int (uint)				The UUID for the vehicle's localization frame.
maxAngularVelocity	double		AngularVelocity		Maximum angular velocity for wheels (default = 0)
minAngularVelocity	double		AngularVelocity		Minimum angular velocity for wheels (default = 0)
speedScaleFactor	double		Dimensionless		Scale parameter for wheel turn rate.
wheelRadius	double		Length		Wheel radius
wheel_frame_uuid	unsigned int (uint)				The UUID for the wheel frame.
wheel_rate_factor	double		Dimensionless		The wheel rate factor (0 to 1.0) used to generate a smooth angle profile for the motors to get to the total desired angle change. The larger the number the faster the profile.

#### Discrete States

Name	Type	Size	Quantity	Units	Description
arc_radius	double		Length		Same as flowIns radius, the distance of the chassis frame from the turn center (0 means turn in place, + means left, - means right)
chassis_from_turn_center_x	double		Length		X component of the distance of the chassis frame from the turn center in the chassis frame. This value is 0.
chassis_from_turn_center_y	double		Length		Y component of the distance of the chassis frame from the turn center in the chassis fframe. This value is the input turn arc radius.
desired_wheel_angle	double		Angle		Delta wheel rotation (radians) needed to cover desired ground arc distance
prev_drive_angle	double		Angle		Drive angle at previous step (seems to be in wheel encoder angle) {not used?}
prev_drive_rate	double		AngularVelocity		Current value of desired wheel angle
reference_wheel_angle	double		Angle		The absolute wheel encoder angle at start of new sub-arc command. The commanded motion is wrt to this reference wheel angle.
status	int				Current status (0 → moving, 1 → ready) for new command
wheel_arc_distance	double		Length		Ground distance along the arc that the wheel needs to move.
wheel_scale_direction	double		Flag		When 1, the rotation of the wheel is in the same direction as the turn angle. The value is based on the wheel's position wrt the turn center
wheel_speed	double		Velocity		Wheel's linear ground speed along its arc path. It is the product of the wheel_turn_radius and the chassis angular velocity.
wheel_turn_radius	double		Length		Distance of the wheel from the turn center. It is the square root of the sum of the squares of the X & Y components.
wheel_x_position	double		Length		X component of the position of the wheel wrt the turn center in the chassis frame
wheel_y_position	double		Length		Y component of the position of the wheel wrt the turn center in the chassis frame

#### Flow Ins

Name	Type	Size	Quantity	Units	Description
angle	double		Angle		Chassis angle to transverse during turn in place
angular_velocity	double		AngularVelocity		Chassis angular velocity wrt inertial frame in the chassis frame
chassis_motion2turn_center_f2f_uuid	unsigned long (ulong)				Chassis motion to turn center frame location frame2frame UUID.
distance	double		Length		Ground distance to traverse for the chassis when moving forward/backward
linear_velocity	double		Velocity		Chassis's linear velocity wrt inertial frame in the chassis frame. Only used for straight drive motion.
mode	int				mode (0 → turn in place, 1 → move forward, -1 → move reverse, 2 → arc-drive). No used directly - only used to detect a new arc command has been issued.
nav_drive_state	int				Flag for indicating is wheels can drive (true/1 if steering completed, false/0 otherwise)
speed	double		Dimensionless		0 → minimum speed, 1 → maximum speed
wheel_enc	double		Angle		Current value of wheel encoder

#### Flow Outs

Name	Type	Size	Quantity	Units	Description
total_drive_angle	double		Angle		Total delta in wheel drive angle needed (in radians) for the wheel along the arc
wheel_drive_angle	double		Angle		Desired delta wheel drive angle (in radians) for wheel
wheel_drive_rate	double		AngularVelocity		Desired wheel drive rate for wheel

### 31.28. RoverNavModels::WheelDriveVelocity Flow Model



This model documentation is auto-generated. Documentation updates should be made to the model's `mdl` file to avoid the changes being lost.

[Doxygen link to all RoverNavModels models of Flow class](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/RoverNavModels/html/group_Flow_group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/RoverNavModels/html/group\_Flow\_group.html)

#### Keywords Doxygen groups

- [Rover](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/RoverNavModels/html/group_Rover_keyword_group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/RoverNavModels/html/group\_Rover\_keyword\_group.html)
- [Rover!Locomotion](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/RoverNavModels/html/group_Rover_Locomotion_keyword_group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/RoverNavModels/html/group\_Rover\_Locomotion\_keyword\_group.html)
- [Rover!Odometry](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/RoverNavModels/html/group_Rover_Odometry_keyword_group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/RoverNavModels/html/group\_Rover\_Odometry\_keyword\_group.html)
- [Rover!n-wheel](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/RoverNavModels/html/group_Rover_n-wheel_keyword_group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/RoverNavModels/html/group\_Rover\_n-wheel\_keyword\_group.html)

#### Description

Model of rover wheel velocity kinematics for a non-steered wheel on a rover.

This model computes the wheel drive velocity corresponding to a desired vehicle velocity.

from the {RoverNavModels\_module\_uri}[RoverNavModels] models library for the xref:Dshellpp\_module simulation framework.

#### Model WheelDriveVelocity class details

For more information on the members and functions of this model class, please see [RoverNavModels::WheelDriveVelocity model class doxygen documentation](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/RoverNavModels/html/group__WheelDriveVelocity_group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/RoverNavModels/html/group\_\_WheelDriveVelocity\_group.html)

#### Parameters

Name	Type	Size	Quantity	Units	Description
chassis_frame_uuid	unsigned int (uint)				The UUID for the chassis frame.
direction	double		Flag		+1 → rover moves towards bogey when wheels rotate positive. -1 → rover moves towards rocker when wheels rotate positive
maxAngularVelocity	double		AngularVelocity		Maximum angular velocity for wheels (default = 0)
minAngularVelocity	double		AngularVelocity		Minimum angular velocity for wheels (default = 0)
speedScaleFactor	double		Dimensionless		Scale parameter for wheel turn rate.
wheelRadius	double		Length		Wheel radius
wheel_frame_uuid	unsigned int (uint)				The UUID for the wheel frame.
wheel_rate_factor	double		Dimensionless		The wheel rate factor (0 to 1.0) used to generate a smooth angle profile for the motors to get to the total desired angle change. The larger the number the faster the profile.

#### Scratch

Name	Type	Size	Quantity	Units	Description
arc_radius	double		Length		The chassis turn radius based on the desired angular velocity and the desired XY velocity for the chassis.
chassis_from_turn_center_x	double		Length		X component of the distance of the chassis frame from the turn center
chassis_from_turn_center_y	double		Length		Y component of the distance of the chassis frame from the turn center
linear_velocity	double		Velocity		The magnitude of the desired XY linear velocity of the chassis
wheel_scale_direction	double		Flag		The direction the wheel should turn because of its position wrt the turn center

Name	Type	Size	Quantity	Units	Description
wheel_speed	double		Velocity		The transational speed of the wheel.
wheel_turn_radius	double		Length		The distance of the wheel from the turn center
wheel_x_position	double		Length		X component of the distance of the wheel from the turn center
wheel_y_position	double		Length		Y component of the distance of the wheel from the turn center

#### Discrete States

Name	Type	Size	Quantity	Units	Description
prev_drive_rate	double		AngularVelocity		Current value of desired wheel angle
status	int				Current status (0 → moving, 1 → ready for new command)

#### Flow Ins

Name	Type	Size	Quantity	Units	Description
chassis_motion2turn_center_f2f_uuid	unsigned long (ulong)				Chassis motion to turn center frame location frame2frame UUID.
desired_x_y_h_rate	double	3	Mixed		The desired instantaneous chassis x,y,h rates
initialized_pose	int				Flag to inticate pose is initialized; (1, initialized; 0 ,not initialized)
ready_status	int				Flag to indicate if can drive

#### Flow Outs

Name	Type	Size	Quantity	Units	Description
total_drive_angle	double		Angle		Total desired wheel drive angle for the wheel motion
wheel_drive_angle	double		Angle		Instantaneous desired wheel drive aangle for wheel
wheel_drive_rate	double		AngularVelocity		Desired wheel drive rate for wheel

### 31.29. RoverNavModels::WheelSteerDriveMotion Flow Model



This model documentation is auto-generated. Documentation updates should be made to the model's `mdl` file to avoid the changes being lost.

[Doxygen link to all RoverNavModels models of Flow class](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/RoverNavModels/html/group_Flow_group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/RoverNavModels/html/group\_Flow\_group.html)

#### Keywords Doxygen groups

- [Rover](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/RoverNavModels/html/group_Rover_keyword_group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/RoverNavModels/html/group\_Rover\_keyword\_group.html)
- [Rover!Locomotion](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/RoverNavModels/html/group_Rover_Locomotion_keyword_group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/RoverNavModels/html/group\_Rover\_Locomotion\_keyword\_group.html)
- [Rover!Odometry](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/RoverNavModels/html/group_Rover_Odometry_keyword_group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/RoverNavModels/html/group\_Rover\_Odometry\_keyword\_group.html)
- [RoverIn-wheel](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/RoverNavModels/html/group_Rover_n-wheel_keyword_group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/RoverNavModels/html/group\_Rover\_n-wheel\_keyword\_group.html)

#### Description

Model of rover wheel kinematics for a steered wheel on a rover.

This model computes the steer angle and the drive distance this wheel should steer/drive to traverse the arc it will follow when the vehicle drives along a circular arc specified by the flow-in desired vehicle motion. The steer angle computation is based on the following. At any given instance, the desired wheel rotation axis should ideally be along the turn center to wheel center vector. We compute this vector in the chassis frame. We also keep track of the wheel axis in the chassis frame when the steer angle is zero. The steer angle is the steer rotation needed to line up these two vectors, and thus is the angle between these two vectors.

from the {RoverNavModels\_module\_uri}[RoverNavModels] models library for the xref:Dshellpp\_module simulation framework.

#### Model WheelSteerDriveMotion class details

For more information on the members and functions of this model class, please see [RoverNavModels::WheelSteerDriveMotion model class doxygen documentation](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/RoverNavModels/html/group__WheelSteerDriveMotion_group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/RoverNavModels/html/group\_\_WheelSteerDriveMotion\_group.html)

#### Parameters

Name	Type	Size	Quantity	Units	Description
chassis_frame_uuid	unsigned int (uint)				The UUID for the chassis frame.
ck_mode	int				A flag to indicate if using CK mode (-1) or not (=0).
steerBodyId	unsigned int (uint)				The UUID for the steer body for this wheel.
steerOffsetAngle	double		Angle		The offset between the zero steer angle and the chassis x-axis
steerParentBodyId	unsigned int (uint)				The UUID for the parent body of the steer body for this wheel.
steer_rate_factor	double		Dimensionless		The steer rate factor to slow down the steer angle change
wheelBodyId	unsigned int (uint)				The UUID for the wheel body for this wheel.

#### Scratch

Name	Type	Size	Quantity	Units	Description
curr_steer_theta	double		Angle		Current steer hinge coordinate value
curr_yaw_angle	double		Angle		Current yaw angle for the wheel axis
desired_yaw_angle	double		Angle		Desired yaw angle for the wheel axis
motion_type	string				Type of motion being commanded

#### Discrete States

Name	Type	Size	Quantity	Units	Description
desired_wheel_steer_angle	double		Angle		Current value of desired wheel angle
first_run	int				Flag to indicate that the computeSteerMotion is run the first time
new_output_steer_angle	double		Angle		The steer angle output from this model

Name	Type	Size	Quantity	Units	Description
orig_steer_wrt_chassis_yaw	double		Angle		The initial yaw attitude of the steer body with respect to the chassis
performed_drive_motion	int				Flag to indicate if the wheel drive part was performed
reference_wheel_steer_angle	double		Angle		Actual wheel steer angle at start of new sub-arc
wheel_axis_angle	double		Angle		The angle of the wheel joint axis wrt the steering body frame (used as an offset in computing the steering angle)

#### Flow Ins

Name	Type	Size	Quantity	Units	Description
steerStatus	int				Current steer status (0 → not complete, 1 → complete)

#### Flow Outs

Name	Type	Size	Quantity	Units	Description
total_steer_angle	double		Angle		Total desired wheel steer angle for the wheel
wheel_steer_angle	double		Angle		Desired wheel steer angle for wheel

### 31.30. RoverNavModels::WheelSteerDriveVelocity Flow Model



This model documentation is auto-generated. Documentation updates should be made to the model's `mdl` file to avoid the changes being lost.

[Doxygen link to all RoverNavModels models of Flow class](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/RoverNavModels/html/group_Flow_group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/RoverNavModels/html/group\_Flow\_group.html)

#### Keywords Doxygen groups

- [Rover](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/RoverNavModels/html/group_Rover_keyword_group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/RoverNavModels/html/group\_Rover\_keyword\_group.html)
- [Rover!Locomotion](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/RoverNavModels/html/group_Rover_Locomotion_keyword_group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/RoverNavModels/html/group\_Rover\_Locomotion\_keyword\_group.html)
- [Rover!Odometry](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/RoverNavModels/html/group_Rover_Odometry_keyword_group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/RoverNavModels/html/group\_Rover\_Odometry\_keyword\_group.html)
- [Rover!n-wheel](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/RoverNavModels/html/group_Rover_n-wheel_keyword_group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/RoverNavModels/html/group\_Rover\_n-wheel\_keyword\_group.html)

#### Description

Model of rover wheel kinematics for a steered wheel on a rover.

This model computes the steer angle and the drive distance this wheel should steer/drive to traverse the arc it will follow when the vehicle drives along a circular arc specified by the flow-in desired vehicle motion.

from the {RoverNavModels\_module\_uri}[RoverNavModels] models library for the xref:Dshellpp\_module simulation framework.

#### Model WheelSteerDriveVelocity class details

For more information on the members and functions of this model class, please see [RoverNavModels::WheelSteerDriveVelocity model class doxygen documentation](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/RoverNavModels/html/group_WheelSteerDriveVelocity_group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/RoverNavModels/html/group\_WheelSteerDriveVelocity\_group.html)

#### Parameters

Name	Type	Size	Quantity	Units	Description
ck_mode	int				A flag to indicate if using CK mode (=1) or not (=0).



Name	Type	Size	Quantity	Units	Description
steerBodyId	unsigned int (uint)				The UUID for the steer body for this wheel.
steerOffsetAngle	double		Angle		The offset between the zero steer angle and the chassis x-axis
steerParentBodyId	unsigned int (uint)				The UUID for the parent body of the steer body for this wheel.
steer_rate_factor	double		Dimensionless		The steer rate factor to slow down the steer angle change
wheelBodyId	unsigned int (uint)				The UUID for the wheel body for this wheel.

#### Discrete States

Name	Type	Size	Quantity	Units	Description
desired_wheel_steer_angle	double		Angle		Current value of desired wheel angle
first_run	int				Flag to indicate that the computeSteerMotion is run the first time
new_output_steer_angle	double		Angle		The steer angle output from this model
orig_steer_wrt_chassis_yaw	double		Angle		The initial yaw attitude of the steer body with respect to the chassis
performed_drive_motion	int				Flag to indicate if the wheel drive part was performed
reference_wheel_steer_angle	double		Angle		Actual wheel steer angle at start of new sub-arc
wheel_axis_angle	double		Angle		The wheel drive joint axis wrt the chassis

#### Flow Outs

Name	Type	Size	Quantity	Units	Description
total_steer_angle	double		Angle		Total desired wheel steer angle for the wheel
wheel_steer_angle	double		Angle		Desired wheel steer angle for wheel

## 32. SurfaceContactModels Dshell model library

### Doxygen reference to Models in SurfaceContactModels by Type

- [Actuators](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group__Actuator_group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group\_\_Actuator\_group.html)
- [Sensors](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group__Sensor_group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group\_\_Sensor\_group.html)
- [Motors](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group__Motor_group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group\_\_Motor\_group.html)
- [Encoders](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group__Encoder_group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group\_\_Encoder\_group.html)
- [Flows](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group__Flow_group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group\_\_Flow\_group.html)
- [Continuous](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group__ContStates_group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group\_\_ContStates\_group.html)

### 32.1. SurfaceContactModels::BekkerWheelSoilContact Actuator Model



This model documentation is auto-generated. Documentation updates should be made to the model's `mdl` file to avoid the changes being lost.

[Doxygen link to all SurfaceContactModels models of Actuator class](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group__Actuator_group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group\_\_Actuator\_group.html)

#### Keywords Doxygen groups

- [Constraint](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group__Constraint_keyword_group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group\_\_Constraint\_keyword\_group.html)
- [Constraint!Soft](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group__Constraint_Soft_keyword_group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group\_\_Constraint\_Soft\_keyword\_group.html)
- [Contact](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group__Contact_keyword_group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group\_\_Contact\_keyword\_group.html)
- [Contact!Terrain](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group__Contact_Terrain_keyword_group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group\_\_Contact\_Terrain\_keyword\_group.html)
- [Contact!Wheel](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group__Contact_Wheel_keyword_group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group\_\_Contact\_Wheel\_keyword\_group.html)
- [Friction](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group__Friction_keyword_group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group\_\_Friction\_keyword\_group.html)
- [Rover](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group__Rover_keyword_group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group\_\_Rover\_keyword\_group.html)
- [Rover!Dynamics](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group__Rover_Dynamics_keyword_group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group\_\_Rover\_Dynamics\_keyword\_group.html)
- [Rover!Wheel/Soil](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group__Rover_Wheel-Soil_keyword_group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group\_\_Rover\_Wheel-Soil\_keyword\_group.html)
- [Soil](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group__Soil_keyword_group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group\_\_Soil\_keyword\_group.html)
- [Terrain](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group__Terrain_keyword_group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group\_\_Terrain\_keyword\_group.html)
- [Terrain!Contact](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group__Terrain_Contact_keyword_group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group\_\_Terrain\_Contact\_keyword\_group.html)
- [Wheel](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group__Wheel_keyword_group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group\_\_Wheel\_keyword\_group.html)
- [Wheel!Contact](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group__Wheel_Contact_keyword_group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group\_\_Wheel\_Contact\_keyword\_group.html)
- [Wheel!Soil](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group__Wheel_Soil_keyword_group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group\_\_Wheel\_Soil\_keyword\_group.html)
- [contact](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group__contact_keyword_group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group\_\_contact\_keyword\_group.html)
- [mechanics](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group__mechanics_keyword_group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group\_\_mechanics\_keyword\_group.html)

#### Description

Applies compliant contact force to a body based on a Bekker terramechanics model.

This model computes and applies wheel/soil contact forces.

from the SurfaceContactModels models library for the xref:Dshellpp\_module simulation framework.

#### Model BekkerWheelSoilContact class details

For more information on the members and functions of this model class, please see [SurfaceContactModels::BekkerWheelSoilContact model class doxygen documentation](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group__BekkerWheelSoilContact_group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group\_\_BekkerWheelSoilContact\_group.html)

#### Parameters

Name	Type	Size	Quantity	Units	Description
cohesion_modulus	double		Dimensionless		Bekker soil cohesion modulus
default_cohesion	double		Pressure		Soil cohesion
default_density	double		Density		soil density
default_phi	double		Angle		internal friction angle

Name	Type	Size	Quantity	Units	Description
demSpecName	string				The name of the top level DVar spec with the vehicle's TopoDem pointer.
exit_angle_coeffs	double	2	Mixed		Coefficient for determining exit angle from the entry angle
exponent	double		Dimensionless		Bekker normal force exponent
friction_modulus	double		Dimensionless		Bekker friction angle modulus
lateral_shear_modulus_coeffs	double	2	Mixed		The constant and linear coeffs used for computing the lateral shear modulus from the lateral slip angle.
longitudinal_shear_modulus_coeffs	double	2	Mixed		The constant and linear coeffs used for computing the longitudinal shear modulus from the lateral slip angle.
max_stress_angle_coeffs	double	2	Mixed		The constant and linear coeffs used for computing the max stress angle from the longitudinal stress
soil_damping	double		LinearSpringDamping		Soil damping coefficient
wheel_radius	double		Length		wheel radius (used to compute contact patch area)
wheel_width	double		Length		wheel width (used to compute contact patch area)

#### Scratch

Name	Type	Size	Quantity	Units	Description
cohesion	double		Pressure		Current soil cohesion
density	double		Density		Current soil density
force	double	3	Force		Contact constraint force (contact frame)
phi	double		Angle		Current internal friction angle
torque	double	3	Torque		Contact constraint torque (contact frame)
wheel_moment	double	3	Torque		Wheel moment (contact frame)

#### Flow Ins

Name	Type	Size	Quantity	Units	Description
gravity	double	3	Acceleration		Gravity vector
sinkage	double		Length		penetration distance



This model documentation is auto-generated. Documentation updates should be made to the model's `mdl` file to avoid the changes being lost.

[Doxygen link to all SurfaceContactModels models of Actuator class](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group__Actuator_group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group\_\_Actuator\_group.html)

### Keywords Doxygen groups

- [Constraint](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group__Constraint_keyword_group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group\_\_Constraint\_keyword\_group.html)
- [Constraint!Soft](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group__Constraint_Soft_keyword_group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group\_\_Constraint\_Soft\_keyword\_group.html)
- [Contact](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group__Contact_keyword_group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group\_\_Contact\_keyword\_group.html)
- [Contact!Terrain](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group__Contact_Terrain_keyword_group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group\_\_Contact\_Terrain\_keyword\_group.html)
- [Contact!Wheel](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group__Contact_Wheel_keyword_group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group\_\_Contact\_Wheel\_keyword\_group.html)
- [Friction](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group__Friction_keyword_group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group\_\_Friction\_keyword\_group.html)
- [Rover](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group__Rover_keyword_group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group\_\_Rover\_keyword\_group.html)
- [Rover!Dynamics](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group__Rover_Dynamics_keyword_group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group\_\_Rover\_Dynamics\_keyword\_group.html)
- [Rover!Wheel/Soil](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group__Rover_Wheel-Soil_keyword_group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group\_\_Rover\_Wheel-Soil\_keyword\_group.html)
- [Rover!Wheel/Variable](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group__Rover_Wheel-Variable_keyword_group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group\_\_Rover\_Wheel-Variable\_keyword\_group.html)
- [Soil](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group__Soil_keyword_group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group\_\_Soil\_keyword\_group.html)
- [Terrain](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group__Terrain_keyword_group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group\_\_Terrain\_keyword\_group.html)
- [Terrain!Contact](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group__Terrain_Contact_keyword_group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group\_\_Terrain\_Contact\_keyword\_group.html)
- [Wheel](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group__Wheel_keyword_group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group\_\_Wheel\_keyword\_group.html)
- [Wheel!Contact](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group__Wheel_Contact_keyword_group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group\_\_Wheel\_Contact\_keyword\_group.html)
- [Wheel!Soil](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group__Wheel_Soil_keyword_group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group\_\_Wheel\_Soil\_keyword\_group.html)
- [contact](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group__contact_keyword_group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group\_\_contact\_keyword\_group.html)
- [mechanics](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group__mechanics_keyword_group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group\_\_mechanics\_keyword\_group.html)
- [soil](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group__soil_keyword_group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group\_\_soil\_keyword\_group.html)

### Description

Applies compliant contact force to a body based on a Bekker terramechanics model on a terrain with variable soil properties.

This model computes and applies wheel/soil contact forces on a terrain with variable soil properties.

from the SurfaceContactModels models library for the xref:Dshellpp\_module simulation framework.

### Model BekkerWheelVariableSoilContact class details

For more information on the members and functions of this model class, please see [SurfaceContactModels::BekkerWheelVariableSoilContact model class doxygen documentation](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group__BekkerWheelVariableSoilContact_group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group\_\_BekkerWheelVariableSoilContact\_group.html)

### Parameters

Name	Type	Size	Quantity	Units	Description
bekker_soil_params	BekkerSoilParams	*			Variable size list of Struct containing Bekker soil parameter values
demSpecName	string				The name of the top level DVar spec with the vehicle's TopoDem pointer.
wheel_radius	double		Length		wheel radius (used to compute contact patch area)
wheel_width	double		Length		wheel width (used to compute contact patch area)

### Scratch

Name	Type	Size	Quantity	Units	Description
body_pos	double	3	Length		The location of the body in terrain frame
current_bekker_soil_params	BekkerSoilParams				Current soil parameters

Name	Type	Size	Quantity	Units	Description
force	double	3	Force		Contact constraint force (contact frame)
soil_type	int				Current soil type under the wheel.
torque	double	3	Torque		Contact constraint torque (contact frame)
wheel_moment	double	3	Torque		Wheel moment (contact frame)

#### Flow Ins

Name	Type	Size	Quantity	Units	Description
gravity	double	3	Acceleration		Gravity vector
sinkage	double		Length		penetration distance

### 32.3. SurfaceContactModels::CompliantContact Actuator Model



This model documentation is auto-generated. Documentation updates should be made to the model's `mdl` file to avoid the changes being lost.

[Doxygen link to all SurfaceContactModels models of Actuator class](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group_Actuator_group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group\_Actuator\_group.html)

#### Keywords Doxygen groups

- [Constraint](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group_Constraint_keyword_group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group\_Constraint\_keyword\_group.html)
- [Constraint!Soft](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group_Constraint_Soft_keyword_group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group\_Constraint\_Soft\_keyword\_group.html)
- [Contact](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group_Contact_keyword_group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group\_Contact\_keyword\_group.html)
- [Contact!Terrain](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group_Contact_Terrain_keyword_group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group\_Contact\_Terrain\_keyword\_group.html)
- [Contact!Wheel](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group_Contact_Wheel_keyword_group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group\_Contact\_Wheel\_keyword\_group.html)
- [Crossley](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group_Crossley_keyword_group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group\_Crossley\_keyword\_group.html)
- [Friction](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group_Friction_keyword_group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group\_Friction\_keyword\_group.html)
- [Rover](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group_Rover_keyword_group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group\_Rover\_keyword\_group.html)
- [Rover!Dynamics](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group_Rover_Dynamics_keyword_group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group\_Rover\_Dynamics\_keyword\_group.html)
- [Rover!Wheel/Soil](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group_Rover_Wheel-Soil_keyword_group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group\_Rover\_Wheel-Soil\_keyword\_group.html)
- [Soil](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group_Soil_keyword_group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group\_Soil\_keyword\_group.html)
- [Spring/Damper](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group_Spring-Damper_keyword_group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group\_Spring-Damper\_keyword\_group.html)
- [Spring/Damper!Hunt](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group_Spring-Damper_Hunt_keyword_group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group\_Spring-Damper\_Hunt\_keyword\_group.html)
- [Spring/Damper!Nonlinear](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group_Spring-Damper_Nonlinear_keyword_group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group\_Spring-Damper\_Nonlinear\_keyword\_group.html)
- [Terrain](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group_Terrain_keyword_group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group\_Terrain\_keyword\_group.html)
- [Terrain!Contact](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group_Terrain_Contact_keyword_group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group\_Terrain\_Contact\_keyword\_group.html)
- [Wheel](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group_Wheel_keyword_group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group\_Wheel\_keyword\_group.html)
- [Wheel!Contact](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group_Wheel_Contact_keyword_group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group\_Wheel\_Contact\_keyword\_group.html)
- [Wheel!Soil](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group_Wheel_Soil_keyword_group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group\_Wheel\_Soil\_keyword\_group.html)
- [contact](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group_contact_keyword_group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group\_contact\_keyword\_group.html)
- [mechanics](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group_mechanics_keyword_group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group\_mechanics\_keyword\_group.html)

#### Description

Applies compliant contact force to a body based on the non-linear Hunt Crossley spring/damper equations

This model computes and applies a compliant, frictional contact force for a single contact point. Current Limitations: Normal plane unit directions are written in the inertial frame. This does not allow for different slip/roll behaviour in the lateral and transverse directions. To overcome this, a technique for correctly displacing the contact point within a local reference frame (which may be rotating due to steering) must be developed.

from the SurfaceContactModels models library for the xref:Dshellpp\_module simulation framework.

## Model CompliantContact class details

For more information on the members and functions of this model class, please see [SurfaceContactModels::CompliantContact model class doxygen documentation](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group__CompliantContact_group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group\_\_CompliantContact\_group.html)

### Enums

#### ContactLocationFrame Enum

In the CompliantContact model definition, the original enum ContactLocationFrame is defined as:

```
enum ContactLocationFrame
{
    inertial = 0,
    body = 1
};
```

In the C model code, these enum values can be accessed by using the follow C enum definitions:

```
/// CompliantContactContactLocationFrame enum
enum CompliantContactContactLocationFrame
{
    COMPLIANT_CONTACT_CONTACT_LOCATION_FRAME_INERTIAL = 0,
    COMPLIANT_CONTACT_CONTACT_LOCATION_FRAME_BODY = 1
};
```

#### ForceModel Enum

In the CompliantContact model definition, the original enum ForceModel is defined as:

```
enum ForceModel
{
    nonlinear = 0,
    linear = 1
};
```

In the C model code, these enum values can be accessed by using the follow C enum definitions:

```
/// CompliantContactForceModel enum
enum CompliantContactForceModel
{
    COMPLIANT_CONTACT_FORCE_MODEL_NONLINEAR = 0,
    COMPLIANT_CONTACT_FORCE_MODEL_LINEAR = 1
};
```

#### ContactType Enum

In the CompliantContact model definition, the original enum ContactType is defined as:

```
enum ContactType
{
    NoContact = 0,
    Sliding = 1,
    Rolling = 2
};
```

In the C model code, these enum values can be accessed by using the follow C enum definitions:

```
/// CompliantContactContactType enum
enum CompliantContactContactType
{
    COMPLIANT_CONTACT_CONTACT_TYPE_NO_CONTACT = 0,
    COMPLIANT_CONTACT_CONTACT_TYPE_SLIDING = 1,
    COMPLIANT_CONTACT_CONTACT_TYPE_ROLLING = 2
};
```

### Parameters

Name	Type	Size	Quantity	Units	Description
alpha	double	2	LinearSpringDamping		alpha for normal and tangent directions
frame	ContactLocationFrame				Contact location frame (default is body frame)
kp	double	2	LinearSpringStiffness		Spring constant for compliant contact in normal and tangent directions

Name	Type	Size	Quantity	Units	Description
maxForce	double		Force		Maximum allowed applied force - prevents numeric problems
n	double	2	Dimensionless		exponential for deflection term for normal and tangent directions
normalForceModel	ForceModel				Normal force model (default is non-linear)
tangentForceModel	ForceModel				Tangent force model (default is linear)
tangent_x	double	3	Dimensionless		First tangent direction (inertial coordinates).
tangent_y	double	3	Dimensionless		Second tangent direction (inertial coordinates, orthogonal to tangent_x)

### Scratch

Name	Type	Size	Quantity	Units	Description
contactVelocity	double	3	Velocity		Velocity of contact point (node frame)
force	double	3	Force		Contact constraint force (node frame)
nodeAngularVelocity	double	3	AngularVelocity		Linear velocity of node center (node coordinates)
nodeLinearVelocity	double	3	Velocity		Linear velocity of node center (node coordinates)
nodeframeNormal	double	3	Dimensionless		normal vector written in node frame
normalVelocity	double	3	Velocity		Velocity of point in normal direction
penetrationDistance	double		Length		Penetration distance
tangentX	double	3	Dimensionless		Rotated params.tangent_x to match orientation of current contact normal
tangentY	double	3	Dimensionless		Rotated params.tangent_y to match orientation of current contact normal
tangent_z	double	3	Dimensionless		Nominal normal direction (inertial coordinates)
torque	double	3	Torque		Contact constraint torque (node frame)

### Discrete States

Name	Type	Size	Quantity	Units	Description
currentContactPlot	int				Current contact type (for plotting)
currentContactType	ContactType				Current contact type
normalForce	double	3	Force		Force of normal direction spring/damper (node frame)
tangentForce	double	3	Force		Force of tangent direction spring/damper (node frame)

## Continuous States

Name	Type	Size	Quantity	Units	Description
tangentDeflection	double	2	Length		Current deflection of contact point in tangential directions.

## Flow Ins

Name	Type	Size	Quantity	Units	Description
contactLocation	double	3	Length		Location of contact relative to body node (inertial coordinats for TESTING ONLY, see code)
mu	double		Dimensionless		coefficient of friction at contact point - used to determine maximum allowable tangent force
normalDirection	double	3	Dimensionless		Normal vector for contact (inertial coordinates)
penetrationDistance	double		Length		pen dist

## Flow Outs

Name	Type	Size	Quantity	Units	Description
currCState	double	2	Length		current cont state (used for testing)

## 32.4. SurfaceContactModels::CompliantContact2 Actuator Model



This model documentation is auto-generated. Documentation updates should be made to the model's `mdl` file to avoid the changes being lost.

[Doxygen link to all SurfaceContactModels models of Actuator class](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group_Actuator_group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group\_Actuator\_group.html)

### Keywords Doxygen groups

- [Constraint](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group_Constraint_keyword_group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group\_Constraint\_keyword\_group.html)
- [Constraint!Soft](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group_Constraint_Soft_keyword_group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group\_Constraint\_Soft\_keyword\_group.html)
- [Contact](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group_Contact_keyword_group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group\_Contact\_keyword\_group.html)
- [Contact!Terrain](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group_Contact_Terrain_keyword_group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group\_Contact\_Terrain\_keyword\_group.html)
- [Contact!Wheel](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group_Contact_Wheel_keyword_group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group\_Contact\_Wheel\_keyword\_group.html)
- [Crossley](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group_Crossley_keyword_group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group\_Crossley\_keyword\_group.html)
- [Friction](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group_Friction_keyword_group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group\_Friction\_keyword\_group.html)
- [Rover](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group_Rover_keyword_group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group\_Rover\_keyword\_group.html)
- [Rover!Dynamics](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group_Rover_Dynamics_keyword_group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group\_Rover\_Dynamics\_keyword\_group.html)
- [Rover!Wheel/Soil](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group_Rover_Wheel-Soil_keyword_group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group\_Rover\_Wheel-Soil\_keyword\_group.html)
- [Soil](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group_Soil_keyword_group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group\_Soil\_keyword\_group.html)
- [Spring/Damper](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group_Spring-Damper_keyword_group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group\_Spring-Damper\_keyword\_group.html)
- [Spring/Damper!Hunt](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group_Spring-Damper_Hunt_keyword_group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group\_Spring-Damper\_Hunt\_keyword\_group.html)
- [Spring/Damper!Nonlinear](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group_Spring-Damper_Nonlinear_keyword_group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group\_Spring-Damper\_Nonlinear\_keyword\_group.html)
- [Terrain](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group_Terrain_keyword_group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group\_Terrain\_keyword\_group.html)
- [Terrain!Contact](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group_Terrain_Contact_keyword_group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group\_Terrain\_Contact\_keyword\_group.html)
- [Wheel](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group_Wheel_keyword_group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group\_Wheel\_keyword\_group.html)
- [Wheel!Contact](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group_Wheel_Contact_keyword_group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group\_Wheel\_Contact\_keyword\_group.html)
- [Wheel!Soil](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group_Wheel_Soil_keyword_group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group\_Wheel\_Soil\_keyword\_group.html)
- [contact](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group_contact_keyword_group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group\_contact\_keyword\_group.html)



- [mechanics](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group__mechanics__keyword__group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group\_\_mechanics\_\_keyword\_\_group.html)

### Description

Applies compliant contact force to a body based on the non-linear Hunt Crossley spring/damper equations

This model computes and applies a compliant, frictional contact force for a single contact point. Current Limitations: Normal plane unit directions are written in the inertial frame. This does not allow for different slip/roll behaviour in the lateral and transverse directions. To overcome this, a technique for correctly displacing the contact point within a local reference frame (which may be rotating due to steering) must be developed.

from the SurfaceContactModels models library for the xref:Dshellpp\_module simulation framework.

### Model CompliantContact2 class details

For more information on the members and functions of this model class, please see [SurfaceContactModels::CompliantContact2 model class doxygen documentation](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group__CompliantContact2__group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group\_\_CompliantContact2\_\_group.html)

### Enums

#### ContactLocationFrame Enum

In the CompliantContact2 model definition, the original enum ContactLocationFrame is defined as:

```
enum ContactLocationFrame
{
    inertial = 0,
    body = 1
};
```

In the C model code, these enum values can be accessed by using the follow C enum definitions:

```
/// CompliantContact2ContactLocationFrame enum
enum CompliantContact2ContactLocationFrame
{
    COMPLIANT_CONTACT2_CONTACT_LOCATION_FRAME_INERTIAL = 0,
    COMPLIANT_CONTACT2_CONTACT_LOCATION_FRAME_BODY = 1
};
```

#### ForceModel Enum

In the CompliantContact2 model definition, the original enum ForceModel is defined as:

```
enum ForceModel
{
    nonlinear = 0,
    linear = 1,
    terra = 2
};
```

In the C model code, these enum values can be accessed by using the follow C enum definitions:

```
/// CompliantContact2ForceModel enum
enum CompliantContact2ForceModel
{
    COMPLIANT_CONTACT2_FORCE_MODEL_NONLINEAR = 0,
    COMPLIANT_CONTACT2_FORCE_MODEL_LINEAR = 1,
    COMPLIANT_CONTACT2_FORCE_MODEL_TERRA = 2
};
```

#### ContactType Enum

In the CompliantContact2 model definition, the original enum ContactType is defined as:

```
enum ContactType
{
    NoContact = 0,
    Sliding = 1,
    Rolling = 2
};
```

In the C model code, these enum values can be accessed by using the follow C enum definitions:

```
/// CompliantContact2ContactType enum
enum CompliantContact2ContactType
{
    COMPLIANT_CONTACT2_CONTACT_TYPE_NO_CONTACT = 0,
    COMPLIANT_CONTACT2_CONTACT_TYPE_SLIDING = 1,
    COMPLIANT_CONTACT2_CONTACT_TYPE_ROLLING = 2
};
```

### Parameters

Name	Type	Size	Quantity	Units	Description
alpha	double	3	LinearSpringDamping		alpha for normal, tangX and tangY directions
contactFrame	ContactLocationFrame				Contact location frame (default is body frame)
kp	double	3	LinearSpringStiffness		Spring constant for compliant contact in normal, tangX and tangY directions
maxForce	double		Force		Maximum allowed applied force - prevents numeric problems
n	double	3	Dimensionless		exponential for deflection term for normal, tangX and tangY directions
normalForceModel	ForceModel				Normal force model (default is non-linear)
tangXForceModel	ForceModel				Tangent force model for forward direction (default is linear)
tangY	double	3	Dimensionless		Sideways tangent unit direction (body coordinates, matches rotation axis of wheel)
tangYForceModel	ForceModel				Tangent force model for sideways direction (default is linear)

#### Scratch

Name	Type	Size	Quantity	Units	Description
contactVelocity	double	3	Velocity		Velocity of contact point (node frame)
contactVelocityNormal	double	3	Velocity		Velocity of contact point computed using nodeAngularVelocityNormal (node frame)
nodeAngularVelocity	double	3	AngularVelocity		Angular velocity of node center (node coordinates)
nodeAngularVelocityNormal	double	3	AngularVelocity		Angular velocity of node center with any portion about the wheel rotation axis zeroed out (node coordinates)
nodeLinearVelocity	double	3	Velocity		Linear velocity of node center (node coordinates)
nodeframeNormal	double	3	Dimensionless		normal vector written in node frame
normalVelocity	double	3	Velocity		Velocity of point in normal direction
penetrationDistance	double		Length		Penetration distance
tangX	double	3	Dimensionless		tangent unit direction (body coordinates) - orthogonal to params.tangY and the current terrain normal direction

#### Discrete States

Name	Type	Size	Quantity	Units	Description
currXContactType	ContactType				Current contact type direction
currYContactType	ContactType				Current contact type direction
normalForce	double	3	Force		Force of normal direction spring/damper (node frame)
tangXForce	double	3	Force		Force in tangX direction (node frame)
tangYForce	double	3	Force		Force in tangY direction (node frame)

#### Continuous States

Name	Type	Size	Quantity	Units	Description
tangentDeflection	double	2	Length		Current deflection of contact point in tangX and tangY directions.

#### Flow Ins

Name	Type	Size	Quantity	Units	Description
contactLocation	double	3	Length		Location of contact relative to node (body or inertial frame based on contactFrame parameter)
muX	double		Dimensionless		coefficient of friction at contact point - used to determine maximum allowable tangent force in x direction
muY	double		Dimensionless		coefficient of friction at contact point - used to determine maximum allowable tangent force in y direction
normalDirection	double	3	Dimensionless		Normal vector for contact (inertial coordinates)
penetrationDistance	double		Length		pen dist

#### Flow Outs

Name	Type	Size	Quantity	Units	Description
currCState	double	2	Length		current cont state (used for testing/debugging)

### 32.5. SurfaceContactModels::CompliantTerzaghi Actuator Model



This model documentation is auto-generated. Documentation updates should be made to the model's `mdl` file to avoid the changes being lost.

[Doxygen link to all SurfaceContactModels models of Actuator class](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group_Actuator_group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group\_Actuator\_group.html)

#### Keywords Doxygen groups

- [Constraint](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group_Constraint_keyword_group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group\_Constraint\_keyword\_group.html)
- [Constraint!Soft](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group_Constraint_Soft_keyword_group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group\_Constraint\_Soft\_keyword\_group.html)
- [Contact](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group_Contact_keyword_group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group\_Contact\_keyword\_group.html)

- [Contact!Terrain](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group__Contact__Terrain__keyword__group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group\_\_Contact\_\_Terrain\_\_keyword\_\_group.html)
- [Contact!Wheel](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group__Contact__Wheel__keyword__group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group\_\_Contact\_\_Wheel\_\_keyword\_\_group.html)
- [Crossley](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group__Crossley__keyword__group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group\_\_Crossley\_\_keyword\_\_group.html)
- [Friction](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group__Friction__keyword__group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group\_\_Friction\_\_keyword\_\_group.html)
- [Rover](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group__Rover__keyword__group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group\_\_Rover\_\_keyword\_\_group.html)
- [Rover!Dynamics](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group__Rover__Dynamics__keyword__group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group\_\_Rover\_\_Dynamics\_\_keyword\_\_group.html)
- [Rover!Wheel/Soil](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group__Rover__Wheel-Soil__keyword__group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group\_\_Rover\_\_Wheel-Soil\_\_keyword\_\_group.html)
- [Soil](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group__Soil__keyword__group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group\_\_Soil\_\_keyword\_\_group.html)
- [Spring/Damper](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group__Spring-Damper__keyword__group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group\_\_Spring-Damper\_\_keyword\_\_group.html)
- [Spring/Damper!Hunt](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group__Spring-Damper__Hunt__keyword__group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group\_\_Spring-Damper\_\_Hunt\_\_keyword\_\_group.html)
- [Spring/Damper!Nonlinear](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group__Spring-Damper__Nonlinear__keyword__group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group\_\_Spring-Damper\_\_Nonlinear\_\_keyword\_\_group.html)
- [Terrain](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group__Terrain__keyword__group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group\_\_Terrain\_\_keyword\_\_group.html)
- [Terrain!Contact](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group__Terrain__Contact__keyword__group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group\_\_Terrain\_\_Contact\_\_keyword\_\_group.html)
- [Wheel](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group__Wheel__keyword__group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group\_\_Wheel\_\_keyword\_\_group.html)
- [Wheel!Contact](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group__Wheel__Contact__keyword__group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group\_\_Wheel\_\_Contact\_\_keyword\_\_group.html)
- [Wheel!Soil](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group__Wheel__Soil__keyword__group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group\_\_Wheel\_\_Soil\_\_keyword\_\_group.html)
- [contact](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group__contact__keyword__group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group\_\_contact\_\_keyword\_\_group.html)
- [mechanics](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group__mechanics__keyword__group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group\_\_mechanics\_\_keyword\_\_group.html)

### Description

Applies compliant contact force to a body based on a simplified Terzaghi model.

This model computes and applies wheel/soil contact forces. Normal force is computed using a non-linear spring model. A linear spring/damper applies forces in the plane tangent to the contact normal. Terzaghi base soil model is used to compute the transition between rolling and sliding behavior in the tangent direction.

from the SurfaceContactModels models library for the xref:Dshellpp\_module simulation framework.

### Model CompliantTerzaghi class details

For more information on the members and functions of this model class, please see [SurfaceContactModels::CompliantTerzaghi model class doxygen documentation](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group__CompliantTerzaghi__group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group\_\_CompliantTerzaghi\_\_group.html)

### Enums

#### ContactType Enum

In the CompliantTerzaghi model definition, the original enum ContactType is defined as:

```
enum ContactType
{
    NoContact = 0,
    Sliding = 1,
    Rolling = 2
};
```

In the C model code, these enum values can be accessed by using the follow C enum definitions:

```
/// CompliantTerzaghiContactType enum
enum CompliantTerzaghiContactType
{
    COMPLIANT_TERZAGHI_CONTACT_TYPE_NO_CONTACT = 0,
    COMPLIANT_TERZAGHI_CONTACT_TYPE_SLIDING = 1,
    COMPLIANT_TERZAGHI_CONTACT_TYPE_ROLLING = 2
};
```

### Parameters

Name	Type	Size	Quantity	Units	Description
alpha	double	2	LinearSpringDamping		alpha for normal and tangent directions
demSpecName	string				The name of the top level DVar spec with the vehicle's TopoDem pointer

Name	Type	Size	Quantity	Units	Description
kp	double	2	LinearSpringStiffness		Spring constant for compliant contact in normal and tangent directions
maxForce	double		Force		Maximum allowed applied force - prevents numeric problems
max_stddev	double		Dimensionless		Maximum standard deviation for randomness
n	double	2	Dimensionless		exponential for deflection term for normal and tangent directions
normalForceModel	bool				Normal force model (default true, i.e. nonlinear)
rolling_resistance_scale	double		Dimensionless		scale factor for experimental rolling resistance code (default = 0)
seeds	double	2	Dimensionless	2	Initial random seeds
slope_stddev	double		Dimensionless		Slope of standard deviation (should be less than zero)
tangentForceModel	bool				Tangent force model (default false, i.e. linear)
tangent_x	double	3	Dimensionless	3	First tangent direction (inertial coordinates).
tangent_y	double	3	Dimensionless		Second tangent direction (inertial coordinates, orthogonal to tangent_x)
use_random	bool				Randomness off (0) or on (1)
wheel_radius	double		Length		wheel radius (used to compute contact patch area)
wheel_width	double		Length		wheel width (used to compute contact patch area)

### Scratch

Name	Type	Size	Quantity	Units	Description
Nc	double		Dimensionless		Terzaghi Cohesion constant
Ng	double		Dimensionless		Terzaghi Gamma constant
Nq	double		Dimensionless		Terzaghi Soil surcharge constant
area	double		Area		Computed area of contact patch
contactLocationInertial	double	3	Length		Location of contact (inertial frame)
contactVelocity	double	3	Velocity		Velocity of contact point (node frame)
force	double	3	Force		Contact constraint force (node frame)
gamma	double		SpecificWeight		Soil specific weight = gravity * density
mag_grav	double		Acceleration		Magnitude of gravity
nodeAngularVelocity	double	3	AngularVelocity		Linear velocity of node center (node coordinates)

Name	Type	Size	Quantity	Units	Description
nodeLinearVelocity	double	3	Velocity		Linear velocity of node center (node coordinates)
nodeframeNormal	double	3	Dimensionless		normal vector written in node frame
normalForceInertial	double	3	Force		Force of normal direction spring/damper (inertial frame)
normalVelocity	double	3	Velocity		Velocity of point in normal direction
patch_length	double		Length		Length of contact patch
penetrationDistance	double		Length		Penetration distance
tanPhi	double		Angle		Tangent of friction angle
tangentForceInertial	double	3	Force		Force of tangent direction spring/damper (inertial frame)
tangentX	double	3	Dimensionless		Rotated params.tangent_x to match orientation of current contact normal
tangentX_inertial	double	3	Dimensionless		Rotated params.tangent_x to match orientation of current contact normal (in inertial coordinates)
tangentY	double	3	Dimensionless		Rotated params.tangent_y to match orientation of current contact normal
tangentY_inertial	double	3	Dimensionless		Rotated params.tangent_y to match orientation of current contact normal (in inertial coordinates)
tangent_z	double	3	Dimensionless		Nominal normal direction (inertial coordinates)
torque	double	3	Torque		Contact constraint torque (node frame)

#### Discrete States

Name	Type	Size	Quantity	Units	Description
contact_area	double		Area		Estimate of contact patch area
currentContactPlot	int				Current contact type (for plotting)
currentContactType	ContactType				Current contact type
current_seeds	double	2	Dimensionless		Current random seeds
demSpecName	string				The name of the top level DVar spec with the vehicle's TopoDem pointer.
normalForce	double	3	Force		Force of normal direction spring/damper (node frame)
sinkage	double		Length		Estimate of sinkage
tangentForce	double	3	Force		Force of tangent direction spring/damper (node frame)
traction_scale	double		Dimensionless		Current traction scale

## Continuous States

Name	Type	Size	Quantity	Units	Description
tangentDeflection	double	2	Length		Current deflection of contact point in tangential directions.

## Flow Ins

Name	Type	Size	Quantity	Units	Description
cohesion	double		Pressure		Soil cohesion (kPa)
contactLocation	double	3	Length		Location of contact relative to body node (body frame)
density	double		Density		soil density (kg/m <sup>3</sup> )
gravity	double	3	Acceleration		Gravity vector
normalDirection	double	3	Dimensionless		Normal vector for contact (inertial frame)
penetrationDistance	double		Length		penetration distance
phi	double		Angle		internal friction angle (radians)

## Flow Outs

Name	Type	Size	Quantity	Units	Description
currCState	double	2	Length		current cont state (used for testing)

## 32.6. SurfaceContactModels::CompliantTerzaghiBaseAlt Actuator Model



This model documentation is auto-generated. Documentation updates should be made to the model's `mdl` file to avoid the changes being lost.

[Doxygen link to all SurfaceContactModels models of Actuator class](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group_Actuator_group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group\_Actuator\_group.html)

### Keywords Doxygen groups

- [Constraint](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group_Constraint_keyword_group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group\_Constraint\_keyword\_group.html)
- [Constraint!Soft](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group_Constraint_Soft_keyword_group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group\_Constraint\_Soft\_keyword\_group.html)
- [Contact](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group_Contact_keyword_group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group\_Contact\_keyword\_group.html)
- [Contact!Terrain](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group_Contact_Terrain_keyword_group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group\_Contact\_Terrain\_keyword\_group.html)
- [Crossley](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group_Crossley_keyword_group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group\_Crossley\_keyword\_group.html)
- [Friction](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group_Friction_keyword_group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group\_Friction\_keyword\_group.html)
- [Rover](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group_Rover_keyword_group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group\_Rover\_keyword\_group.html)
- [Rover!Dynamics](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group_Rover_Dynamics_keyword_group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group\_Rover\_Dynamics\_keyword\_group.html)
- [Rover!Pad/Soil](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group_Rover_Pad-Soil_keyword_group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group\_Rover\_Pad-Soil\_keyword\_group.html)
- [Soil](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group_Soil_keyword_group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group\_Soil\_keyword\_group.html)
- [Spring/Damper](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group_Spring-Damper_keyword_group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group\_Spring-Damper\_keyword\_group.html)
- [Spring/Damper!Hunt](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group_Spring-Damper_Hunt_keyword_group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group\_Spring-Damper\_Hunt\_keyword\_group.html)
- [Spring/Damper!Nonlinear](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group_Spring-Damper_Nonlinear_keyword_group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group\_Spring-Damper\_Nonlinear\_keyword\_group.html)
- [Terrain](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group_Terrain_keyword_group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group\_Terrain\_keyword\_group.html)
- [Terrain!Contact](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group_Terrain_Contact_keyword_group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group\_Terrain\_Contact\_keyword\_group.html)
- [contact](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group_contact_keyword_group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group\_contact\_keyword\_group.html)
- [mechanics](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group_mechanics_keyword_group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group\_mechanics\_keyword\_group.html)

### Description

Model that applies compliant contact force to a body based on a simplified Terzaghi model.

This model computes and applies soil contact forces on object in contact with the surface. This model assumes that when the pad is in contact with the soil, all parts of the pad are in contact. This model is a reimplement of the CompliantTerzaghiPad model except that it uses the TerrainContactForce helper class. Normal force is computed using a non-linear spring model. A linear spring/damper applies forces in the plane tangent to the contact normal. Terzaghi base soil model is used to compute the transition between non-sliding and sliding behavior in the tangent direction.

from the SurfaceContactModels models library for the xref:Dshellpp\_module simulation framework.

### Model CompliantTerzaghiBaseAlt class details

For more information on the members and functions of this model class, please see [SurfaceContactModels::CompliantTerzaghiBaseAlt model class doxygen documentation](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group__CompliantTerzaghiBaseAlt_group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group\_\_CompliantTerzaghiBaseAlt\_group.html)

### Enums

#### BodyType Enum

In the CompliantTerzaghiBaseAlt model definition, the original enum BodyType is defined as:

```
enum BodyType
{
    wheel = 0,
    pad = 1
};
```

In the C model code, these enum values can be accessed by using the follow C enum definitions:

```
/// CompliantTerzaghiBaseAltBodyType enum
enum CompliantTerzaghiBaseAltBodyType
{
    COMPLIANT_TERZAGHI_BASE_ALT_BODY_TYPE_WHEEL = 0,
    COMPLIANT_TERZAGHI_BASE_ALT_BODY_TYPE_PAD = 1
};
```

### Parameters

Name	Type	Size	Quantity	Units	Description
alpha	double	2	LinearSpringDamping		Damping coefficient for normal and tangent directions
bodyType	BodyType				Wheel or pad body type
default_cohesion	double		Pressure		Soil cohesion (kPa)
default_density	double		Density		soil density
default_phi	double		Angle		internal friction angle (radians)
demSpecName	string				The name of the top level DVar spec with the vehicle's TopoDem pointer.
kp	double	2	LinearSpringStiffness		Spring constant for compliant contact in normal and tangent directions
maxForce	double		Force		Maximum allowed applied force - prevents numeric problems
max_stddev	double		Dimensionless		Maximum standard deviation for randomness
n	double	2	Dimensionless		exponential for deflection term for normal and tangent directions
normalForceModel	bool				Normal force model (default true, i.e. nonlinear)



Name	Type	Size	Quantity	Units	Description
rolling_resistance_scale	double		Dimensionless		scale factor for experimental rolling resistance code (default = 0)
seeds	double	2	Dimensionless		Initial random seeds
slope_stddev	double		Dimensionless		Slope of standard deviation (should be less that zero)
tangentForceModel	bool				Tangent force model (default false, i.e. linear)
use_random	int				Randomness off (0) or on (1)

#### Scratch

Name	Type	Size	Quantity	Units	Description
cohesion	double		Pressure		Current soil cohesion
current_seeds	double	2	Dimensionless		Current random seeds
density	double		Density		Current soil density
force	double	3	Force		Contact constraint force (node frame)
phi	double		Angle		Current internal friction angle
torque	double	3	Torque		Contact constraint torque (node frame)
traction_scale	double		Dimensionless		Current traction scale

#### Continuous States

Name	Type	Size	Quantity	Units	Description
tangentDeflection	double	2	Length		Current deflection of contact point in tangential directions.

#### Flow Ins

Name	Type	Size	Quantity	Units	Description
gravity	double	3	Acceleration		Gravity vector
penetrationDistance	double		Length		penetration distance

#### Flow Outs

Name	Type	Size	Quantity	Units	Description
currCState	double	2	Length		current cont state (used for testing)

## 32.7. SurfaceContactModels::CompliantTerzaghiBaseAltVariableSoil Actuator Model



This model documentation is auto-generated. Documentation updates should be made to the model's `mdl` file to avoid the changes being lost.

[Doxygen link to all SurfaceContactModels models of Actuator class](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group_Actuator_group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group\_Actuator\_group.html)

#### Keywords Doxygen groups

- [Constraint](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group_Constraint_keyword_group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group\_Constraint\_keyword\_group.html)
- [Constraint!Soft](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group_Constraint_Soft_keyword_group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group\_Constraint\_Soft\_keyword\_group.html)

- [Contact](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group__Contact_keyword_group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group\_\_Contact\_keyword\_group.html)
- [Contact!Terrain](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group__Contact_Terrain_keyword_group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group\_\_Contact\_Terrain\_keyword\_group.html)
- [Crossley](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group__Crossley_keyword_group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group\_\_Crossley\_keyword\_group.html)
- [Friction](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group__Friction_keyword_group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group\_\_Friction\_keyword\_group.html)
- [Rover](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group__Rover_keyword_group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group\_\_Rover\_keyword\_group.html)
- [Rover!Dynamics](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group__Rover_Dynamics_keyword_group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group\_\_Rover\_Dynamics\_keyword\_group.html)
- [Rover!Pad/Soil](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group__Rover_Pad-Soil_keyword_group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group\_\_Rover\_Pad-Soil\_keyword\_group.html)
- [Rover!Wheel/Variable](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group__Rover_Wheel-Variable_keyword_group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group\_\_Rover\_Wheel-Variable\_keyword\_group.html)
- [Soil](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group__Soil_keyword_group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group\_\_Soil\_keyword\_group.html)
- [Spring/Damper](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group__Spring-Damper_keyword_group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group\_\_Spring-Damper\_keyword\_group.html)
- [Spring/Damper!Hunt](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group__Spring-Damper_Hunt_keyword_group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group\_\_Spring-Damper\_Hunt\_keyword\_group.html)
- [Spring/Damper!Nonlinear](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group__Spring-Damper_Nonlinear_keyword_group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group\_\_Spring-Damper\_Nonlinear\_keyword\_group.html)
- [Terrain](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group__Terrain_keyword_group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group\_\_Terrain\_keyword\_group.html)
- [Terrain!Contact](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group__Terrain_Contact_keyword_group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group\_\_Terrain\_Contact\_keyword\_group.html)
- [contact](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group__contact_keyword_group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group\_\_contact\_keyword\_group.html)
- [mechanics](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group__mechanics_keyword_group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group\_\_mechanics\_keyword\_group.html)
- [soil](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group__soil_keyword_group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group\_\_soil\_keyword\_group.html)

### Description

Model that applies compliant contact force to a body based on a simplified Terzaghi model on a terrain with variable soil properties.

This model computes and applies soil contact forces on object in contact with the surface. This model assumes that when the pad is in contact with the soil, all parts of the pad are in contact. This model is a reimplementaion of the CompliantTerzaghiPad model except that it uses the TerrainContactForce helper class. Normal force is computed using a non-linear spring model. A linear spring/damper applies forces in the plane tangent to the contact normal. Terzahi base soil model is used to compute the transition between non-sliding and sliding behavior in the tangent direction. Soil properties are specified through a SurfaceIndicantOverlay SimScape object.

from the SurfaceContactModels models library for the xref:Dshellpp\_module simulation framework.

### Model CompliantTerzaghiBaseAltVariableSoil class details

For more information on the members and functions of this model class, please see [SurfaceContactModels::CompliantTerzaghiBaseAltVariableSoil model class doxygen documentation](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group__CompliantTerzaghiBaseAltVariableSoil_group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group\_\_CompliantTerzaghiBaseAltVariableSoil\_group.html)

### Enums

#### BodyType Enum

In the CompliantTerzaghiBaseAltVariableSoil model definition, the original enum BodyType is defined as:

```
enum BodyType
{
    wheel = 0,
    pad = 1
};
```

In the C model code, these enum values can be accessed by using the follow C enum definitions:

```
/// CompliantTerzaghiBaseAltVariableSoilBodyType enum
enum CompliantTerzaghiBaseAltVariableSoilBodyType
{
    COMPLIANT_TERZAGHI_BASE_ALT_VARIABLE_SOIL_BODY_TYPE_WHEEL = 0,
    COMPLIANT_TERZAGHI_BASE_ALT_VARIABLE_SOIL_BODY_TYPE_PAD = 1
};
```

### Parameters

Name	Type	Size	Quantity	Units	Description
bodyType	BodyType				Wheel or pad body type
demSpecName	string				The name of the top level DVar spec with the vehicle's TopoDem pointer.
max_stddev	double		Dimensionless		Maximum standard deviation for randomness
seeds	double	2	Dimensionless		Initial random seeds

Name	Type	Size	Quantity	Units	Description
slope_stddev	double		Dimensionless		Slope of standard deviation (should be less than zero)
terzaghi_soil_params	TerzaghiSoilParams	*			Variable size list of Struct containing Terzaghi parameter values for each soil type
use_random	int				Randomness off (0) or on (1)

#### Scratch

Name	Type	Size	Quantity	Units	Description
body_pos	double	3	Length		The location of the body in terrain frame
current_seeds	double	2	Dimensionless		Current random seeds
current_terzaghi_soil_params	TerzaghiSoilParams				Current soil parameters
force	double	3	Force		Contact constraint force (node frame)
soil_type	int				Current soil type under the wheel.
torque	double	3	Torque		Contact constraint torque (node frame)
traction_scale	double		Dimensionless		Current traction scale

#### Continuous States

Name	Type	Size	Quantity	Units	Description
tangentDeflection	double	2	Length		Current deflection of contact point in tangential directions.

#### Flow Ins

Name	Type	Size	Quantity	Units	Description
gravity	double	3	Acceleration		Gravity vector
penetrationDistance	double		Length		penetration distance

#### Flow Outs

Name	Type	Size	Quantity	Units	Description
currCState	double	2	Length		current cont state (used for testing)

## 32.8. SurfaceContactModels::CompliantTerzaghiCylindricalWheelAlt Actuator Model



This model documentation is auto-generated. Documentation updates should be made to the model's `mdl` file to avoid the changes being lost.

[Doxygen link to all SurfaceContactModels models of Actuator class](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group_Actuator_group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group\_Actuator\_group.html)

#### Keywords Doxygen groups

- [Constraint](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group_Constraint_keyword_group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group\_Constraint\_keyword\_group.html)
- [Constraint!Soft](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group_Constraint_Soft_keyword_group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group\_Constraint\_Soft\_keyword\_group.html)
- [Contact](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group_Contact_keyword_group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group\_Contact\_keyword\_group.html)
- [Contact!Terrain](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group_Contact_Terrain_keyword_group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group\_Contact\_Terrain\_keyword\_group.html)

- [Contact!Wheel](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group__Contact__Wheel__keyword__group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group\_\_Contact\_\_Wheel\_\_keyword\_\_group.html)
- [Crossley](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group__Crossley__keyword__group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group\_\_Crossley\_\_keyword\_\_group.html)
- [Friction](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group__Friction__keyword__group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group\_\_Friction\_\_keyword\_\_group.html)
- [Rover](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group__Rover__keyword__group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group\_\_Rover\_\_keyword\_\_group.html)
- [Rover!Dynamics](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group__Rover__Dynamics__keyword__group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group\_\_Rover\_\_Dynamics\_\_keyword\_\_group.html)
- [Rover!Wheel/Soil](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group__Rover__Wheel-Soil__keyword__group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group\_\_Rover\_\_Wheel-Soil\_\_keyword\_\_group.html)
- [Soil](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group__Soil__keyword__group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group\_\_Soil\_\_keyword\_\_group.html)
- [Spring/Damper](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group__Spring-Damper__keyword__group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group\_\_Spring-Damper\_\_keyword\_\_group.html)
- [Spring/Damper!Hunt](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group__Spring-Damper__Hunt__keyword__group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group\_\_Spring-Damper\_\_Hunt\_\_keyword\_\_group.html)
- [Spring/Damper!Nonlinear](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group__Spring-Damper__Nonlinear__keyword__group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group\_\_Spring-Damper\_\_Nonlinear\_\_keyword\_\_group.html)
- [Terrain](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group__Terrain__keyword__group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group\_\_Terrain\_\_keyword\_\_group.html)
- [Terrain!Contact](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group__Terrain__Contact__keyword__group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group\_\_Terrain\_\_Contact\_\_keyword\_\_group.html)
- [Wheel](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group__Wheel__keyword__group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group\_\_Wheel\_\_keyword\_\_group.html)
- [Wheel!Contact](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group__Wheel__Contact__keyword__group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group\_\_Wheel\_\_Contact\_\_keyword\_\_group.html)
- [Wheel!Soil](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group__Wheel__Soil__keyword__group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group\_\_Wheel\_\_Soil\_\_keyword\_\_group.html)
- [contact](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group__contact__keyword__group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group\_\_contact\_\_keyword\_\_group.html)
- [mechanics](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group__mechanics__keyword__group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group\_\_mechanics\_\_keyword\_\_group.html)

### Description

Applies compliant contact force to a body based on a simplified Terzaghi model.

This model computes and applies wheel/soil contact forces. This model is a reimplementation of the CompliantTerzaghi model except that it uses the TerrainContactForce helper class. Normal force is computed using a non-linear spring model. A linear spring/damper applies forces in the plane tangent to the contact normal. Terzaghi base soil model is used to compute the transition between rolling and sliding behavior in the tangent direction.

from the SurfaceContactModels models library for the xref:Dshellpp\_module simulation framework.

### Model CompliantTerzaghiCylindricalWheelAlt class details

For more information on the members and functions of this model class, please see [SurfaceContactModels::CompliantTerzaghiCylindricalWheelAlt model class doxygen documentation](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group__CompliantTerzaghiCylindricalWheelAlt__group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group\_\_CompliantTerzaghiCylindricalWheelAlt\_\_group.html)

### Enums

### Parameters

Name	Type	Size	Quantity	Units	Description
wheel_radius	double		Length		wheel radius (used to compute contact patch area)
wheel_width	double		Length		wheel width (used to compute contact patch area)

## 32.9. SurfaceContactModels::CompliantTerzaghiCylindricalWheelAltVariableSoil Actuator Model



This model documentation is auto-generated. Documentation updates should be made to the model's `mdl` file to avoid the changes being lost.

[Doxygen link to all SurfaceContactModels models of Actuator class](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group__Actuator__group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group\_\_Actuator\_\_group.html)

### Keywords Doxygen groups

- [Constraint](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group__Constraint__keyword__group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group\_\_Constraint\_\_keyword\_\_group.html)
- [Constraint!Soft](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group__Constraint__Soft__keyword__group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group\_\_Constraint\_\_Soft\_\_keyword\_\_group.html)
- [Contact](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group__Contact__keyword__group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group\_\_Contact\_\_keyword\_\_group.html)
- [Contact!Terrain](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group__Contact__Terrain__keyword__group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group\_\_Contact\_\_Terrain\_\_keyword\_\_group.html)
- [Contact!Wheel](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group__Contact__Wheel__keyword__group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group\_\_Contact\_\_Wheel\_\_keyword\_\_group.html)
- [Crossley](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group__Crossley__keyword__group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group\_\_Crossley\_\_keyword\_\_group.html)
- [Friction](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group__Friction__keyword__group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group\_\_Friction\_\_keyword\_\_group.html)
- [Rover](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group__Rover__keyword__group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group\_\_Rover\_\_keyword\_\_group.html)

- [Rover!Dynamics](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group_Rover_Dynamics_keyword_group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group\_Rover\_Dynamics\_keyword\_group.html)
- [Rover!Wheel/Soil](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group_Rover_Wheel-Soil_keyword_group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group\_Rover\_Wheel-Soil\_keyword\_group.html)
- [Rover!Wheel/Variable](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group_Rover_Wheel-Variable_keyword_group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group\_Rover\_Wheel-Variable\_keyword\_group.html)
- [Soil](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group_Soil_keyword_group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group\_Soil\_keyword\_group.html)
- [Spring/Damper](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group_Spring-Damper_keyword_group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group\_Spring-Damper\_keyword\_group.html)
- [Spring/Damper!Hunt](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group_Spring-Damper_Hunt_keyword_group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group\_Spring-Damper\_Hunt\_keyword\_group.html)
- [Spring/Damper!Nonlinear](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group_Spring-Damper_Nonlinear_keyword_group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group\_Spring-Damper\_Nonlinear\_keyword\_group.html)
- [Terrain](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group_Terrain_keyword_group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group\_Terrain\_keyword\_group.html)
- [Terrain!Contact](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group_Terrain_Contact_keyword_group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group\_Terrain\_Contact\_keyword\_group.html)
- [Wheel](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group_Wheel_keyword_group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group\_Wheel\_keyword\_group.html)
- [Wheel!Contact](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group_Wheel_Contact_keyword_group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group\_Wheel\_Contact\_keyword\_group.html)
- [Wheel!Soil](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group_Wheel_Soil_keyword_group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group\_Wheel\_Soil\_keyword\_group.html)
- [contact](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group_contact_keyword_group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group\_contact\_keyword\_group.html)
- [mechanics](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group_mechanics_keyword_group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group\_mechanics\_keyword\_group.html)
- [soil](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group_soil_keyword_group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group\_soil\_keyword\_group.html)

### Description

Applies compliant contact force to a body based on a simplified Terzaghi model on a terrain with variable soil properties.

This model computes and applies wheel/soil contact forces. This model is a reimplemention of the CompliantTerzaghi model except that it uses the TerrainContactForce helper class. Normal force is computed using a non-linear spring model. A linear spring/damper applies forces in the plane tangent to the contact normal. Terzahi base soil model is used to compute the transition between rolling and sliding behavior in the tangent direction. Soil properties are specified through a SurfaceIndicantOverlay SimScape object.

from the SurfaceContactModels models library for the xref:Dshellpp\_module simulation framework.

### Model CompliantTerzaghiCylindricalWheelAltVariableSoil class details

For more information on the members and functions of this model class, please see [SurfaceContactModels::CompliantTerzaghiCylindricalWheelAltVariableSoil model class doxygen documentation](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group_CompliantTerzaghiCylindricalWheelAltVariableSoil_model_class_doxygen_documentation) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group\_CompliantTerzaghiCylindricalWheelAltVariableSoil\_group.html)

### Enums

### Parameters

Name	Type	Size	Quantity	Units	Description
wheel_radius	double		Length		wheel radius (used to compute contact patch area)
wheel_width	double		Length		wheel width (used to compute contact patch area)

## 32.10. SurfaceContactModels::CompliantTerzaghiPad Actuator Model



This model documentation is auto-generated. Documentation updates should be made to the model's `mdl` file to avoid the changes being lost.

[Doxygen link to all SurfaceContactModels models of Actuator class](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group_Actuator_group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group\_Actuator\_group.html)

### Keywords Doxygen groups

- [Constraint](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group_Constraint_keyword_group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group\_Constraint\_keyword\_group.html)
- [Constraint!Soft](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group_Constraint_Soft_keyword_group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group\_Constraint\_Soft\_keyword\_group.html)
- [Contact](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group_Contact_keyword_group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group\_Contact\_keyword\_group.html)
- [Contact!Terrain](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group_Contact_Terrain_keyword_group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group\_Contact\_Terrain\_keyword\_group.html)
- [Crossley](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group_Crossley_keyword_group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group\_Crossley\_keyword\_group.html)
- [Friction](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group_Friction_keyword_group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group\_Friction\_keyword\_group.html)
- [Rover](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group_Rover_keyword_group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group\_Rover\_keyword\_group.html)
- [Rover!Dynamics](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group_Rover_Dynamics_keyword_group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group\_Rover\_Dynamics\_keyword\_group.html)
- [Rover!Pad/Soil](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group_Rover_Pad-Soil_keyword_group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group\_Rover\_Pad-Soil\_keyword\_group.html)
- [Soil](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group_Soil_keyword_group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group\_Soil\_keyword\_group.html)

- [Spring/Damper](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group__Spring-Damper__keyword__group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group\_\_Spring-Damper\_\_keyword\_\_group.html)
- [Spring/Damper!Hunt](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group__Spring-Damper_Hunt__keyword__group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group\_\_Spring-Damper\_Hunt\_\_keyword\_\_group.html)
- [Spring/Damper!Nonlinear](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group__Spring-Damper_Nonlinear__keyword__group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group\_\_Spring-Damper\_Nonlinear\_\_keyword\_\_group.html)
- [Terrain](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group__Terrain__keyword__group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group\_\_Terrain\_\_keyword\_\_group.html)
- [Terrain!Contact](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group__Terrain_Contact__keyword__group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group\_\_Terrain\_Contact\_\_keyword\_\_group.html)
- [contact](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group__contact__keyword__group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group\_\_contact\_\_keyword\_\_group.html)
- [mechanics](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group__mechanics__keyword__group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group\_\_mechanics\_\_keyword\_\_group.html)

### Description

Applies compliant contact force to a body based on a simplified Terzaghi model.

This model computes and applies soil contact forces on flat pad in contact with the surface. This model assumes that when the pad is in contact with the soil, all parts of the pad are in contact. Normal force is computed using a non-linear spring model. A linear spring/damper applies forces in the plane tangent to the contact normal. Terzaghi base soil model is used to compute the transition between non-sliding and sliding behavior in the tangent direction.

from the SurfaceContactModels models library for the xref:Dshellpp\_module simulation framework.

### Model CompliantTerzaghiPad class details

For more information on the members and functions of this model class, please see [SurfaceContactModels::CompliantTerzaghiPad model class doxygen documentation](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group__CompliantTerzaghiPad__group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group\_\_CompliantTerzaghiPad\_\_group.html)

### Enums

#### ContactType Enum

In the CompliantTerzaghiPad model definition, the original enum ContactType is defined as:

```
enum ContactType
{
    NoContact = 0,
    Sliding = 1,
    NotSliding = 2
};
```

In the C model code, these enum values can be accessed by using the follow C enum definitions:

```
/// CompliantTerzaghiPadContactType enum
enum CompliantTerzaghiPadContactType
{
    COMPLIANT_TERZAGHI_PAD_CONTACT_TYPE_NO_CONTACT = 0,
    COMPLIANT_TERZAGHI_PAD_CONTACT_TYPE_SLIDING = 1,
    COMPLIANT_TERZAGHI_PAD_CONTACT_TYPE_NOT_SLIDING = 2
};
```

### Parameters

Name	Type	Size	Quantity	Units	Description
alpha	double	2	LinearSpringDamping		alpha for normal and tangent directions
demSpecName	string				The name of the top level DVar spec with the vehicle's TopoDem pointer
kp	double	2	LinearSpringStiffness		Spring constant for compliant contact in normal and tangent directions
maxForce	double		Force		Maximum allowed applied force - prevents numeric problems
max_stddev	double		Dimensionless		Maximum standard deviation for randomness
n	double	2	Dimensionless		exponential for deflection term for normal and tangent directions
normalForceModel	bool				Normal force model (default true, i.e. nonlinear)

Name	Type	Size	Quantity	Units	Description
pad_radius	double		Length		pad radius (used to compute contact patch area)
rolling_resistance_scale	double		Dimensionless		scale factor for experimental rolling resistance code (default = 0)
seeds	double	2	Dimensionless		Initial random seeds
slope_stddev	double		Dimensionless		Slope of standard deviation (should be less than zero)
tangentForceModel	bool				Tangent force model (default false, i.e. linear)
tangent_x	double	3	Dimensionless		First tangent direction (inertial coordinates).
tangent_y	double	3	Dimensionless		Second tangent direction (inertial coordinates, orthogonal to tangent_x)
use_random	bool				Randomness off (0) or on (1)

### Scratch

Name	Type	Size	Quantity	Units	Description
Nc	double		Dimensionless		Terzaghi Cohesion constant
Ng	double		Dimensionless		Terzaghi Gamma constant
Nq	double		Dimensionless		Terzaghi Soil surcharge constant
area	double		Area		Computed area of contact patch
contactLocationInertial	double	3	Length		Location of contact (inertial frame)
contactVelocity	double	3	Velocity		Velocity of contact point (node frame)
force	double	3	Force		Contact constraint force (node frame)
gamma	double		SpecificWeight		Soil specific weight = gravity * density
mag_grav	double		Acceleration		Magnitude of gravity
nodeAngularVelocity	double	3	AngularVelocity		Linear velocity of node center (node coordinates)
nodeLinearVelocity	double	3	Velocity		Linear velocity of node center (node coordinates)
nodeframeNormal	double	3	Dimensionless		normal vector written in node frame
normalForceInertial	double	3	Force		Force of normal direction spring/damper (inertial frame)
normalVelocity	double	3	Velocity		Velocity of point in normal direction
penetrationDistance	double		Length		Penetration distance
tanPhi	double		Angle		Tangent of friction angle
tangentForceInertial	double	3	Force		Force of tangent direction spring/damper (inertial frame)

Name	Type	Size	Quantity	Units	Description
tangentX	double	3	Dimensionless		Rotated params.tangent_x to match orientation of current contact normal
tangentY	double	3	Dimensionless		Rotated params.tangent_y to match orientation of current contact normal
tangent_z	double	3	Dimensionless		Nominal normal direction (inertial coordinates)
torque	double	3	Torque		Contact constraint torque (node frame)

#### Discrete States

Name	Type	Size	Quantity	Units	Description
contact_area	double		Area		Estimate of contact patch area
currentContactPlot	int				Current contact type (for plotting)
currentContactType	ContactType				Current contact type
current_seeds	double	2	Dimensionless		Current random seeds
normalForce	double	3	Force		Force of normal direction spring/damper (node frame)
sinkage	double		Length		Estimate of sinkage
tangentForce	double	3	Force		Force of tangent direction spring/damper (node frame)
traction_scale	double		Dimensionless		Current traction scale

#### Continuous States

Name	Type	Size	Quantity	Units	Description
tangentDeflection	double	2	Length		Current deflection of contact point in tangential directions.

#### Flow Ins

Name	Type	Size	Quantity	Units	Description
cohesion	double		Pressure		Soil cohesion (kPa)
contactLocation	double	3	Length		Location of contact relative to body node (body frame)
density	double		Density		soil density (kg/m <sup>3</sup> )
gravity	double	3	Acceleration		Gravity vector
normalDirection	double	3	Dimensionless		Normal vector for contact (inertial frame)
penetrationDistance	double		Length		penetration distance
phi	double		Angle		internal friction angle (radians)

#### Flow Outs

Name	Type	Size	Quantity	Units	Description
currCState	double	2	Length		current cont state (used for testing)



## 32.11. SurfaceContactModels::CompliantTerzaghiPadAlt Actuator Model



This model documentation is auto-generated. Documentation updates should be made to the model's `mdl` file to avoid the changes being lost.

[Doxygen link to all SurfaceContactModels models of Actuator class](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group__Actuator_group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group\_\_Actuator\_group.html)

### Keywords Doxygen groups

- [Constraint](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group__Constraint_keyword_group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group\_\_Constraint\_keyword\_group.html)
- [Constraint!Soft](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group__Constraint_Soft_keyword_group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group\_\_Constraint\_Soft\_keyword\_group.html)
- [Contact](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group__Contact_keyword_group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group\_\_Contact\_keyword\_group.html)
- [Contact!Terrain](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group__Contact_Terrain_keyword_group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group\_\_Contact\_Terrain\_keyword\_group.html)
- [Crossley](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group__Crossley_keyword_group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group\_\_Crossley\_keyword\_group.html)
- [Friction](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group__Friction_keyword_group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group\_\_Friction\_keyword\_group.html)
- [Rover](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group__Rover_keyword_group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group\_\_Rover\_keyword\_group.html)
- [Rover!Dynamics](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group__Rover_Dynamics_keyword_group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group\_\_Rover\_Dynamics\_keyword\_group.html)
- [Rover!Pad/Soil](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group__Rover_Pad-Soil_keyword_group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group\_\_Rover\_Pad-Soil\_keyword\_group.html)
- [Soil](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group__Soil_keyword_group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group\_\_Soil\_keyword\_group.html)
- [Spring/Damper](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group__Spring-Damper_keyword_group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group\_\_Spring-Damper\_keyword\_group.html)
- [Spring/Damper!Hunt](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group__Spring-Damper_Hunt_keyword_group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group\_\_Spring-Damper\_Hunt\_keyword\_group.html)
- [Spring/Damper!Nonlinear](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group__Spring-Damper_Nonlinear_keyword_group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group\_\_Spring-Damper\_Nonlinear\_keyword\_group.html)
- [Terrain](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group__Terrain_keyword_group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group\_\_Terrain\_keyword\_group.html)
- [Terrain!Contact](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group__Terrain_Contact_keyword_group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group\_\_Terrain\_Contact\_keyword\_group.html)
- [contact](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group__contact_keyword_group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group\_\_contact\_keyword\_group.html)
- [mechanics](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group__mechanics_keyword_group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group\_\_mechanics\_keyword\_group.html)

### Description

Applies compliant contact force to a body based on a simplified Terzaghi model.

This model computes and applies soil contact forces on flat pad in contact with the surface. This model assumes that when the pad is in contact with the soil, all parts of the pad are in contact. This model is a reimplementation of the CompliantTerzaghiPad model except that it uses the TerrainContactForce helper class. Normal force is computed using a non-linear spring model. A linear spring/damper applies forces in the plane tangent to the contact normal. Terzaghi base soil model is used to compute the transition between non-sliding and sliding behavior in the tangent direction.

from the SurfaceContactModels models library for the xref:Dshellpp\_module simulation framework.

### Model CompliantTerzaghiPadAlt class details

For more information on the members and functions of this model class, please see [SurfaceContactModels::CompliantTerzaghiPadAlt model class doxygen documentation](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group__CompliantTerzaghiPadAlt_group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group\_\_CompliantTerzaghiPadAlt\_group.html)

### Enums

### Parameters

Name	Type	Size	Quantity	Units	Description
pad_radius	double		Length		pad radius (used to compute contact patch area)

## 32.12. SurfaceContactModels::CompliantTerzaghiRotatingPad Actuator Model



This model documentation is auto-generated. Documentation updates should be made to the model's `mdl` file to avoid the changes being lost.

[Doxygen link to all SurfaceContactModels models of Actuator class](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group__Actuator_group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group\_\_Actuator\_group.html)

### Keywords Doxygen groups

- [Bevamerter](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group__Bevamerter_keyword_group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group\_\_Bevamerter\_keyword\_group.html)
- [Constraint](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group__Constraint_keyword_group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group\_\_Constraint\_keyword\_group.html)

- [Constraint!Soft](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group__Constraint_Soft_keyword_group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group\_\_Constraint\_Soft\_keyword\_group.html)
- [Contact](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group__Contact_keyword_group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group\_\_Contact\_keyword\_group.html)
- [Contact!Terrain](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group__Contact_Terrain_keyword_group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group\_\_Contact\_Terrain\_keyword\_group.html)
- [Crossley](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group__Crossley_keyword_group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group\_\_Crossley\_keyword\_group.html)
- [Friction](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group__Friction_keyword_group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group\_\_Friction\_keyword\_group.html)
- [Rover](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group__Rover_keyword_group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group\_\_Rover\_keyword\_group.html)
- [Rover!Dynamics](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group__Rover_Dynamics_keyword_group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group\_\_Rover\_Dynamics\_keyword\_group.html)
- [Rover!Pad/Soil](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group__Rover_Pad-Soil_keyword_group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group\_\_Rover\_Pad-Soil\_keyword\_group.html)
- [Soil](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group__Soil_keyword_group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group\_\_Soil\_keyword\_group.html)
- [Spring/Damper](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group__Spring-Damper_keyword_group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group\_\_Spring-Damper\_keyword\_group.html)
- [Spring/Damper!Hunt](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group__Spring-Damper_Hunt_keyword_group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group\_\_Spring-Damper\_Hunt\_keyword\_group.html)
- [Spring/Damper!Nonlinear](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group__Spring-Damper_Nonlinear_keyword_group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group\_\_Spring-Damper\_Nonlinear\_keyword\_group.html)
- [Terrain](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group__Terrain_keyword_group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group\_\_Terrain\_keyword\_group.html)
- [Terrain!Contact](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group__Terrain_Contact_keyword_group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group\_\_Terrain\_Contact\_keyword\_group.html)
- [contact](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group__contact_keyword_group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group\_\_contact\_keyword\_group.html)
- [mechanics](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group__mechanics_keyword_group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group\_\_mechanics\_keyword\_group.html)

### Description

Applies compliant contact force to an annular body based on a simplified Terzaghi model and computes torque from shear stress.

This model is essential the CompliantTerzaghiPadAlt model with additional code to compute torques if the pad rotates. This model computes and applies soil contact forces on an annular pad in contact with the surface. This model assumes that when the pad is in contact with the soil, all parts of the pad are in contact. This model is a reimplementaion of the CompliantTerzaghiPad model except that it uses the TerrainContactForce helper class. Normal force is computed using a non-linear spring model. Normal torque is computed from Bekker's formula for the shear stress: shear stress = (soil\_cohesion + normal\_stress \* coefficient\_of\_internal\_friction) \* (1 - exp(-j/k)) where j = shear displacement k = soil modulus of deformation A linear spring/damper applies forces in the plane tangent to the contact normal. Terzaghi base soil model is used to compute the transition between non-sliding and sliding behavior in the tangent direction.

from the SurfaceContactModels models library for the xref:Dshellpp\_module simulation framework.

### Model CompliantTerzaghiRotatingPad class details

For more information on the members and functions of this model class, please see [SurfaceContactModels::CompliantTerzaghiRotatingPad model class doxygen documentation](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group__CompliantTerzaghiRotatingPad_group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group\_\_CompliantTerzaghiRotatingPad\_group.html)

### Enums

### Parameters

Name	Type	Size	Quantity	Units	Description
deformation_modulus	double		Length		horizontal soil modulus of deformation (meters)
inner_radius	double		Length		inner pad radius (non-zero if pad is annular ring)
pad_radius	double		Length		outer pad radius

### Scratch

Name	Type	Size	Quantity	Units	Description
J	double		Length		displacement (valid only if penetration > 0)
current_theta	double		Angle		Current angle
delta_theta	double		Angle		Angular Displacement (valid only if penetration > 0)
effective_pad_radius	double		Length		radius for computing normal stress
normal_stress	double		Pressure		normal stress (valid only if penetration > 0)

Name	Type	Size	Quantity	Units	Description
normal_torque	double		Torque		normal torque from shear stress (valid only if penetration > 0)
r_mean	double		Length		mean radius for computing displacement (valid only if penetration > 0)
shear_stress	double		Pressure		shear stress (valid only if penetration > 0)

#### Discrete States

Name	Type	Size	Quantity	Units	Description
theta_start	double		Angle		If penetration > 0: angle at start of penetration otherwise equal to current_theta

### 32.13. SurfaceContactModels::ComputePlanePenetration Sensor Model



This model documentation is auto-generated. Documentation updates should be made to the model's `mdl` file to avoid the changes being lost.

[Doxygen link to all SurfaceContactModels models of Sensor class](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group__Sensor__group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group\_\_Sensor\_\_group.html)

#### Keywords Doxygen groups

- [Computational](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group__Computational__keyword__group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group\_\_Computational\_\_keyword\_\_group.html)
- [Distance](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group__Distance__keyword__group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group\_\_Distance\_\_keyword\_\_group.html)
- [Distance!Penetration](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group__Distance__Penetration__keyword__group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group\_\_Distance\_\_Penetration\_\_keyword\_\_group.html)
- [Penetration](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group__Penetration__keyword__group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group\_\_Penetration\_\_keyword\_\_group.html)
- [Penetration!Distance](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group__Penetration__Distance__keyword__group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group\_\_Penetration\_\_Distance\_\_keyword\_\_group.html)
- [Rover](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group__Rover__keyword__group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group\_\_Rover\_\_keyword\_\_group.html)
- [Rover!Wheel/Soil](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group__Rover__Wheel-Soil__keyword__group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group\_\_Rover\_\_Wheel-Soil\_\_keyword\_\_group.html)
- [Surface](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group__Surface__keyword__group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group\_\_Surface\_\_keyword\_\_group.html)
- [Surface!Plane](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group__Surface__Plane__keyword__group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group\_\_Surface\_\_Plane\_\_keyword\_\_group.html)
- [Wheel](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group__Wheel__keyword__group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group\_\_Wheel\_\_keyword\_\_group.html)
- [Wheel!Contact](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group__Wheel__Contact__keyword__group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group\_\_Wheel\_\_Contact\_\_keyword\_\_group.html)
- [Wheel!Soil](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group__Wheel__Soil__keyword__group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group\_\_Wheel\_\_Soil\_\_keyword\_\_group.html)
- [contact](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group__contact__keyword__group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group\_\_contact\_\_keyword\_\_group.html)
- [geometry](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group__geometry__keyword__group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group\_\_geometry\_\_keyword\_\_group.html)
- [geometry!Distance](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group__geometry__Distance__keyword__group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group\_\_geometry\_\_Distance\_\_keyword\_\_group.html)

#### Description

Computes the penetration of a body node into a flat plane.

dummy

from the SurfaceContactModels models library for the xref:Dshellpp\_module simulation framework.

#### Model ComputePlanePenetration class details

For more information on the members and functions of this model class, please see [SurfaceContactModels::ComputePlanePenetration model class doxygen documentation](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group__ComputePlanePenetration__group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group\_\_ComputePlanePenetration\_\_group.html)

#### Parameters

Name	Type	Size	Quantity	Units	Description
normal	double	3	Dimensionless		normal to plane

Name	Type	Size	Quantity	Units	Description
point	double	3	Length		point on plane

#### Flow Outs

Name	Type	Size	Quantity	Units	Description
penetrationDistance	double		Length		penetration distance

## 32.14. SurfaceContactModels::FialaTire Actuator Model



This model documentation is auto-generated. Documentation updates should be made to the model's `mdl` file to avoid the changes being lost.

[Doxygen link to all SurfaceContactModels models of Actuator class](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group__Actuator_group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group\_\_Actuator\_group.html)

#### Keywords Doxygen groups

- [Constraint](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group__Constraint_keyword_group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group\_\_Constraint\_keyword\_group.html)
- [Constraint!Soft](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group__Constraint_Soft_keyword_group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group\_\_Constraint\_Soft\_keyword\_group.html)
- [Contact](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group__Contact_keyword_group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group\_\_Contact\_keyword\_group.html)
- [Contact!Terrain](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group__Contact_Terrain_keyword_group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group\_\_Contact\_Terrain\_keyword\_group.html)
- [Contact!Wheel](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group__Contact_Wheel_keyword_group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group\_\_Contact\_Wheel\_keyword\_group.html)
- [Friction](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group__Friction_keyword_group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group\_\_Friction\_keyword\_group.html)
- [Rover](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group__Rover_keyword_group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group\_\_Rover\_keyword\_group.html)
- [Rover!Dynamics](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group__Rover_Dynamics_keyword_group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group\_\_Rover\_Dynamics\_keyword\_group.html)
- [Rover!Wheel/Soil](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group__Rover_Wheel-Soil_keyword_group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group\_\_Rover\_Wheel-Soil\_keyword\_group.html)
- [Terrain](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group__Terrain_keyword_group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group\_\_Terrain\_keyword\_group.html)
- [Terrain!Contact](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group__Terrain_Contact_keyword_group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group\_\_Terrain\_Contact\_keyword\_group.html)
- [Tire](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group__Tire_keyword_group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group\_\_Tire\_keyword\_group.html)
- [Wheel](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group__Wheel_keyword_group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group\_\_Wheel\_keyword\_group.html)
- [Wheel!Contact](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group__Wheel_Contact_keyword_group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group\_\_Wheel\_Contact\_keyword\_group.html)
- [Wheel!Soil](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group__Wheel_Soil_keyword_group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group\_\_Wheel\_Soil\_keyword\_group.html)
- [contact](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group__contact_keyword_group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group\_\_contact\_keyword\_group.html)

#### Description

Applies contact spatial force for a soft tire using Fiala model.

This model computes and applies tire/hard ground contact forces.

from the SurfaceContactModels models library for the xref:Dshellpp\_module simulation framework.

#### Model FialaTire class details

For more information on the members and functions of this model class, please see [SurfaceContactModels::FialaTire model class doxygen documentation](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group__FialaTire_group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group\_\_FialaTire\_group.html)

#### Parameters

Name	Type	Size	Quantity	Units	Description
demSpecName	string				The name of the top level DVar spec with the vehicle's TopoDem pointer.
friction_limits	double	2	Dimensionless		The static and dynamic coefficients of friction
lateral_tire_stiffness	double		Unspecified		The lateral tire stiffness.
longitudinal_tire_stiffness	double		Force		The longitudinal tire stiffness.

Name	Type	Size	Quantity	Units	Description
normal_damping	double		LinearSpringDamping		The normal damping coefficient for the tire.
normal_stiffness	double		LinearSpringStiffness		The normal stiffness coefficient for the tire.
rolling_resistance_coeff	double		Length		The rolling resistance moment coefficient
wheel_radius	double		Length		wheel radius (used to compute contact patch area)
wheel_width	double		Length		wheel width (used to compute contact patch area)

#### Scratch

Name	Type	Size	Quantity	Units	Description
force	double	3	Force		Contact force (contact frame)
torque	double	3	Torque		Contact moment (contact frame)
wheel_moment	double	3	Torque		Wheel moment (contact frame)

#### Flow Ins

Name	Type	Size	Quantity	Units	Description
deformation	double		Length		tire deformation
gravity	double	3	Acceleration		Gravity vector

### 32.15. SurfaceContactModels::MagicFormulaTireHMMWV Actuator Model



This model documentation is auto-generated. Documentation updates should be made to the model's `mdl` file to avoid the changes being lost.

[Doxygen link to all SurfaceContactModels models of Actuator class](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group__Actuator_group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group\_\_Actuator\_group.html)

#### Keywords Doxygen groups

- [Constraint](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group__Constraint_keyword_group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group\_\_Constraint\_keyword\_group.html)
- [Constraint!Soft](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group__Constraint_Soft_keyword_group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group\_\_Constraint\_Soft\_keyword\_group.html)
- [Contact](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group__Contact_keyword_group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group\_\_Contact\_keyword\_group.html)
- [Contact!Terrain](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group__Contact_Terrain_keyword_group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group\_\_Contact\_Terrain\_keyword\_group.html)
- [Contact!Wheel](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group__Contact_Wheel_keyword_group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group\_\_Contact\_Wheel\_keyword\_group.html)
- [Friction](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group__Friction_keyword_group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group\_\_Friction\_keyword\_group.html)
- [Rover](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group__Rover_keyword_group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group\_\_Rover\_keyword\_group.html)
- [Rover!Dynamics](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group__Rover_Dynamics_keyword_group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group\_\_Rover\_Dynamics\_keyword\_group.html)
- [Rover!Wheel/Soil](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group__Rover_Wheel-Soil_keyword_group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group\_\_Rover\_Wheel-Soil\_keyword\_group.html)
- [Terrain](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group__Terrain_keyword_group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group\_\_Terrain\_keyword\_group.html)
- [Terrain!Contact](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group__Terrain_Contact_keyword_group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group\_\_Terrain\_Contact\_keyword\_group.html)
- [Tire](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group__Tire_keyword_group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group\_\_Tire\_keyword\_group.html)
- [Wheel](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group__Wheel_keyword_group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group\_\_Wheel\_keyword\_group.html)
- [Wheel!Contact](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group__Wheel_Contact_keyword_group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group\_\_Wheel\_Contact\_keyword\_group.html)
- [Wheel!Soil](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group__Wheel_Soil_keyword_group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group\_\_Wheel\_Soil\_keyword\_group.html)
- [contact](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group__contact_keyword_group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group\_\_contact\_keyword\_group.html)

#### Description

Applies contact spatial force for a soft tire using the Magic Formula HMMWV tire model from UMich for the HMMWV vehicle 315\_80R22\_5 tire .

This model computes and applies tire/hard ground contact forces.

from the SurfaceContactModels models library for the xref:Dshellpp\_module simulation framework.

#### Model MagicFormulaTireHMMWV class details

For more information on the members and functions of this model class, please see [SurfaceContactModels::MagicFormulaTireHMMWV model class doxygen documentation](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group_MagicFormulaTireHMMWV_group.html) ([https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group\\_MagicFormulaTireHMMWV\\_group.html](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group_MagicFormulaTireHMMWV_group.html))

#### Parameters

Name	Type	Size	Quantity	Units	Description
demSpecName	string				The name of the top level DVar spec with the vehicle's TopoDem pointer.
normal_damping	double		LinearSpringDamping		The normal damping coefficient for the tire.
normal_stiffness	double		LinearSpringStiffness		The normal stiffness coefficient for the tire.

#### Scratch

Name	Type	Size	Quantity	Units	Description
force	double	3	Force		Contact force (contact frame)
torque	double	3	Torque		Contact moment (contact frame)
wheel_moment	double	3	Torque		Wheel moment (contact frame)

#### Discrete States

Name	Type	Size	Quantity	Units	Description
prev_deformation	double		Length		Tire deformation at the previous time tick

#### Flow Ins

Name	Type	Size	Quantity	Units	Description
camber_angle	double		Angle		the tire tilt angle
deformation	double		Length		tire deformation
gravity	double	3	Acceleration		Gravity vector

## 32.16. SurfaceContactModels::MagicFormulaTireMRZR Actuator Model



This model documentation is auto-generated. Documentation updates should be made to the model's `mdl` file to avoid the changes being lost.

[Doxygen link to all SurfaceContactModels models of Actuator class](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group_Actuator_group.html) ([https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group\\_Actuator\\_group.html](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group_Actuator_group.html))

#### Keywords Doxygen groups

- [Constraint](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group_Constraint_keyword_group.html) ([https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group\\_Constraint\\_keyword\\_group.html](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group_Constraint_keyword_group.html))
- [Constraint!Soft](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group_Constraint_Soft_keyword_group.html) ([https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group\\_Constraint\\_Soft\\_keyword\\_group.html](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group_Constraint_Soft_keyword_group.html))
- [Contact](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group_Contact_keyword_group.html) ([https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group\\_Contact\\_keyword\\_group.html](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group_Contact_keyword_group.html))
- [Contact!Terrain](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group_Contact_Terrain_keyword_group.html) ([https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group\\_Contact\\_Terrain\\_keyword\\_group.html](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group_Contact_Terrain_keyword_group.html))
- [Contact!Wheel](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group_Contact_Wheel_keyword_group.html) ([https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group\\_Contact\\_Wheel\\_keyword\\_group.html](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group_Contact_Wheel_keyword_group.html))

- [Friction](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group__Friction_keyword_group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group\_\_Friction\_keyword\_group.html)
- [Rover](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group__Rover_keyword_group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group\_\_Rover\_keyword\_group.html)
- [Rover!Dynamics](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group__Rover_Dynamics_keyword_group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group\_\_Rover\_Dynamics\_keyword\_group.html)
- [Rover!Wheel/Soil](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group__Rover_Wheel-Soil_keyword_group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group\_\_Rover\_Wheel-Soil\_keyword\_group.html)
- [Terrain](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group__Terrain_keyword_group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group\_\_Terrain\_keyword\_group.html)
- [Terrain!Contact](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group__Terrain_Contact_keyword_group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group\_\_Terrain\_Contact\_keyword\_group.html)
- [Tire](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group__Tire_keyword_group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group\_\_Tire\_keyword\_group.html)
- [Wheel](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group__Wheel_keyword_group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group\_\_Wheel\_keyword\_group.html)
- [Wheel!Contact](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group__Wheel_Contact_keyword_group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group\_\_Wheel\_Contact\_keyword\_group.html)
- [Wheel!Soil](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group__Wheel_Soil_keyword_group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group\_\_Wheel\_Soil\_keyword\_group.html)
- [contact](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group__contact_keyword_group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group\_\_contact\_keyword\_group.html)

#### Description

Applies contact spatial force for a soft tire using the Magic Formula MRZR tire model from UMich for the MRZR vehicle tire .

This model computes and applies tire/hard ground contact forces.

from the SurfaceContactModels models library for the xref:Dshellpp\_module simulation framework.

#### Model MagicFormulaTireMRZR class details

For more information on the members and functions of this model class, please see [SurfaceContactModels::MagicFormulaTireMRZR model class doxygen documentation](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group__MagicFormulaTireMRZR_model_class_doxygen_documentation) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group\_\_MagicFormulaTireMRZR\_group.html)

#### Parameters

Name	Type	Size	Quantity	Units	Description
demSpecName	string				The name of the top level DVar spec with the vehicle's TopoDem pointer.
normal_damping	double		LinearSpringDamping		The normal damping coefficient for the tire.
normal_stiffness	double		LinearSpringStiffness		The normal stiffness coefficient for the tire.

#### Scratch

Name	Type	Size	Quantity	Units	Description
force	double	3	Force		Contact force (contact frame)
torque	double	3	Torque		Contact moment (contact frame)
wheel_moment	double	3	Torque		Wheel moment (contact frame)

#### Discrete States

Name	Type	Size	Quantity	Units	Description
prev_deformation	double		Length		Tire deformation at the previous time tick

#### Flow Ins

Name	Type	Size	Quantity	Units	Description
camber_angle	double		Angle		the tire tilt angle
deformation	double		Length		tire deformation
gravity	double	3	Acceleration		Gravity vector

## 32.17. SurfaceContactModels::ScmDartsModel Actuator Model



This model documentation is auto-generated. Documentation updates should be made to the model's `mdl` file to avoid the changes being lost.

[Doxygen link to all SurfaceContactModels models of Actuator class](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group__Actuator__group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group\_\_Actuator\_\_group.html)

### Description

Soil Contact model for wheel-soil interaction

from the SurfaceContactModels models library for the xref:Dshellpp\_module simulation framework.

### Model ScmDartsModel class details

For more information on the members and functions of this model class, please see [SurfaceContactModels::ScmDartsModel model class doxygen documentation](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group__ScmDartsModel__group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group\_\_ScmDartsModel\_\_group.html)

### Parameters

Name	Type	Size	Quantity	Units	Description
deform_param	deformationParameter				soil deformation parameter
dem	object				The terrain TopoDem instance.
elastic_layer	double		Length		elasticLayer algorithm parameter describing the max elastic layer thickness
erosion_limit	int				erosionLimit max number of allowed erosion loops
numberOfChords	int				number of chords used in the deposition algorithm
numberOfThreads	int				numberOfThreads multi threading settings
scm_flags	int				flags SCM options
soil_param	soilParameter				soil parameter
vd	double		Velocity		velocity for regularization
wheel_param	wheelParameter				Characteristics of wheel
wheel_radius	double		Length		wheel radius (used to compute contact patch area)
wheel_width	double		Length		wheel width (used to compute contact patch area)

### Scratch

Name	Type	Size	Quantity	Units	Description
time	double		Time		current simulation time
wheel_pos	double	3	Torque		Contact constraint torque (node frame)
wheel_vel	double	3	Velocity		Current traction scale

### Flow Ins

Name	Type	Size	Quantity	Units	Description
gravity	double	3	Acceleration		gravity environment parameter describing the gravitational acceleration



## Flow Outs

Name	Type	Size	Quantity	Units	Description
wheel_force	double	3	Force		Force on wheels in x, y, and z direction
wheel_torque	double	3	Torque		Torque on wheels in x, y, and z direction

## 32.18. SurfaceContactModels::TerrainPenAnalytic Sensor Model



This model documentation is auto-generated. Documentation updates should be made to the model's `mdl` file to avoid the changes being lost.

[Doxygen link to all SurfaceContactModels models of Sensor class](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group__Sensor__group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group\_\_Sensor\_\_group.html)

### Keywords Doxygen groups

- [Computational](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group__Computational__keyword__group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group\_\_Computational\_\_keyword\_\_group.html)
- [Contact](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group__Contact__keyword__group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group\_\_Contact\_\_keyword\_\_group.html)
- [Contact!Terrain](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group__Contact__Terrain__keyword__group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group\_\_Contact\_\_Terrain\_\_keyword\_\_group.html)
- [Contact!Wheel](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group__Contact__Wheel__keyword__group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group\_\_Contact\_\_Wheel\_\_keyword\_\_group.html)
- [Distance](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group__Distance__keyword__group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group\_\_Distance\_\_keyword\_\_group.html)
- [Distance!Penetration](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group__Distance__Penetration__keyword__group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group\_\_Distance\_\_Penetration\_\_keyword\_\_group.html)
- [Penetration](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group__Penetration__keyword__group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group\_\_Penetration\_\_keyword\_\_group.html)
- [Penetration!Distance](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group__Penetration__Distance__keyword__group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group\_\_Penetration\_\_Distance\_\_keyword\_\_group.html)
- [Rover](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group__Rover__keyword__group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group\_\_Rover\_\_keyword\_\_group.html)
- [Rover!Wheel](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group__Rover__Wheel__keyword__group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group\_\_Rover\_\_Wheel\_\_keyword\_\_group.html)
- [Rover!Wheel/Soil](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group__Rover__Wheel__Soil__keyword__group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group\_\_Rover\_\_Wheel\_\_Soil\_\_keyword\_\_group.html)
- [Surface](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group__Surface__keyword__group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group\_\_Surface\_\_keyword\_\_group.html)
- [Swift](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group__Swift__keyword__group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group\_\_Swift\_\_keyword\_\_group.html)
- [Terrain](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group__Terrain__keyword__group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group\_\_Terrain\_\_keyword\_\_group.html)
- [contact](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group__contact__keyword__group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group\_\_contact\_\_keyword\_\_group.html)
- [geometry](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group__geometry__keyword__group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group\_\_geometry\_\_keyword\_\_group.html)
- [geometry!Distance](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group__geometry__Distance__keyword__group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group\_\_geometry\_\_Distance\_\_keyword\_\_group.html)
- [geometry!Swift](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group__geometry__Swift__keyword__group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group\_\_geometry\_\_Swift\_\_keyword\_\_group.html)

### Description

Calculates penetration distance of wheel shape into terrain

from the SurfaceContactModels models library for the xref:Dshellpp\_module simulation framework.

### Model TerrainPenAnalytic class details

For more information on the members and functions of this model class, please see [SurfaceContactModels::TerrainPenAnalytic model class doxygen documentation](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group__TerrainPenAnalytic__group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group\_\_TerrainPenAnalytic\_\_group.html)

### Parameters

Name	Type	Size	Quantity	Units	Description
axis	double	3	Dimensionless		Rotation axis of wheel
demSpecName	string				The name of the top level DVar spec with the vehicle's TopoDem pointer
depthfactor	double		Dimensionless		multiplicative factor for wheel penetration of soil on topodem

Name	Type	Size	Quantity	Units	Description
padding	double		Dimensionless		not used
patch_size	double		Length		terrain patch size
radius	double		Length		Actual wheel radius
showtracks	double		Flag		enable terrain change due to tracks
velocityThreshold	double		Velocity		Set Minimum Velocity to show wheel tracks
wheel_penetration_patch_resolution	double		Length		Maximum subsampled resolution to use for computing terrain patch statistics
width	double		Length		Wheel width

#### Scratch

Name	Type	Size	Quantity	Units	Description
normal	double	3	Dimensionless		scratch normal test

#### Flow Ins

Name	Type	Size	Quantity	Units	Description
cohesion	double		Pressure		input soil mechanics cohesion parameter
density	double		Density		input soil mechanics density parameter
frictionAngle	double		Angle		input soil mechanics friction parameter

#### Flow Outs

Name	Type	Size	Quantity	Units	Description
cohesion	double		Pressure		soil mechanics cohesion parameter
contactLocation	double	3	Length		contact point location (body frame)
density	double		Density		soil mechanics density parameter
frictionAngle	double		Angle		soil mechanics friction parameter
penetrationDistance	double		Length		penetration distance
terrainNormal	double	3	Dimensionless		normal to terrain at contact point

### 32.19. SurfaceContactModels::TerrainPenAnalyticBaseAlt Sensor Model



This model documentation is auto-generated. Documentation updates should be made to the model's `mdl` file to avoid the changes being lost.

[Doxygen link to all SurfaceContactModels models of Sensor class](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group__Sensor__group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group\_\_Sensor\_\_group.html)

#### Keywords Doxygen groups

- [Computational](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group__Computational__keyword__group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group\_\_Computational\_\_keyword\_\_group.html)
- [Contact](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group__Contact__keyword__group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group\_\_Contact\_\_keyword\_\_group.html)

- [Contact!Pad](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group_Contact_Pad_keyword_group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group\_Contact\_Pad\_keyword\_group.html)
- [Contact!Terrain](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group_Contact_Terrain_keyword_group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group\_Contact\_Terrain\_keyword\_group.html)
- [Distance](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group_Distance_keyword_group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group\_Distance\_keyword\_group.html)
- [Distance!Penetration](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group_Distance_Penetration_keyword_group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group\_Distance\_Penetration\_keyword\_group.html)
- [Penetration](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group_Penetration_keyword_group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group\_Penetration\_keyword\_group.html)
- [Penetration!Distance](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group_Penetration_Distance_keyword_group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group\_Penetration\_Distance\_keyword\_group.html)
- [Rover](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group_Rover_keyword_group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group\_Rover\_keyword\_group.html)
- [Rover!Pad/Soil](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group_Rover_Pad-Soil_keyword_group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group\_Rover\_Pad-Soil\_keyword\_group.html)
- [Surface](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group_Surface_keyword_group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group\_Surface\_keyword\_group.html)
- [Swift](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group_Swift_keyword_group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group\_Swift\_keyword\_group.html)
- [Terrain](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group_Terrain_keyword_group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group\_Terrain\_keyword\_group.html)
- [contact](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group_contact_keyword_group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group\_contact\_keyword\_group.html)
- [geometry](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group_geometry_keyword_group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group\_geometry\_keyword\_group.html)
- [geometry!Distance](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group_geometry_Distance_keyword_group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group\_geometry\_Distance\_keyword\_group.html)
- [geometry!Swift](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group_geometry_Swift_keyword_group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group\_geometry\_Swift\_keyword\_group.html)

### Description

Base class for calculating penetration distance of an object into the terrain

from the SurfaceContactModels models library for the xref:Dshellpp\_module simulation framework.

### Model TerrainPenAnalyticBaseAlt class details

For more information on the members and functions of this model class, please see [SurfaceContactModels::TerrainPenAnalyticBaseAlt model class doxygen documentation](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group_TerrainPenAnalyticBaseAlt_group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group\_TerrainPenAnalyticBaseAlt\_group.html)

### Parameters

Name	Type	Size	Quantity	Units	Description
demSpecName	string				The name of the top level DVar spec with the vehicle's TopoDem pointer.
patch_resolution	double		Length		Maximum subsampled resolution to use for computing terrain patch statistics
patch_size	double		Length		terrain patch size
use_node_flag	bool				Flag to select whether the node (if false) or the body (if true) location is used for penetration check.

### Scratch

Name	Type	Size	Quantity	Units	Description
contact_pos	double	3	Length		The location of the contact location (inertial frame)
terrain_height	double		Length		The terrain height at the body location

### Flow Outs

Name	Type	Size	Quantity	Units	Description
penetrationDistance	double		Length		penetration distance

## 32.20. SurfaceContactModels::TerrainPenAnalyticCylindricalWheelAlt Sensor Model



This model documentation is auto-generated. Documentation updates should be made to the model's `mdl` file to avoid the changes being lost.

[Doxygen link to all SurfaceContactModels models of Sensor class](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group__Sensor__group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group\_\_Sensor\_\_group.html)

### Keywords Doxygen groups

- [Computational](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group__Computational__keyword__group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group\_\_Computational\_\_keyword\_\_group.html)
- [Contact](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group__Contact__keyword__group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group\_\_Contact\_\_keyword\_\_group.html)
- [Contact!Terrain](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group__Contact__Terrain__keyword__group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group\_\_Contact\_\_Terrain\_\_keyword\_\_group.html)
- [Contact!Wheel](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group__Contact__Wheel__keyword__group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group\_\_Contact\_\_Wheel\_\_keyword\_\_group.html)
- [Distance](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group__Distance__keyword__group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group\_\_Distance\_\_keyword\_\_group.html)
- [Distance!Penetration](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group__Distance__Penetration__keyword__group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group\_\_Distance\_\_Penetration\_\_keyword\_\_group.html)
- [Penetration](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group__Penetration__keyword__group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group\_\_Penetration\_\_keyword\_\_group.html)
- [Penetration!Distance](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group__Penetration__Distance__keyword__group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group\_\_Penetration\_\_Distance\_\_keyword\_\_group.html)
- [Rover](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group__Rover__keyword__group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group\_\_Rover\_\_keyword\_\_group.html)
- [Rover!Wheel](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group__Rover__Wheel__keyword__group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group\_\_Rover\_\_Wheel\_\_keyword\_\_group.html)
- [Rover!Wheel/Soil](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group__Rover__Wheel__Soil__keyword__group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group\_\_Rover\_\_Wheel\_\_Soil\_\_keyword\_\_group.html)
- [Surface](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group__Surface__keyword__group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group\_\_Surface\_\_keyword\_\_group.html)
- [Swift](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group__Swift__keyword__group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group\_\_Swift\_\_keyword\_\_group.html)
- [Terrain](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group__Terrain__keyword__group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group\_\_Terrain\_\_keyword\_\_group.html)
- [contact](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group__contact__keyword__group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group\_\_contact\_\_keyword\_\_group.html)
- [geometry](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group__geometry__keyword__group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group\_\_geometry\_\_keyword\_\_group.html)
- [geometry!Distance](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group__geometry__Distance__keyword__group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group\_\_geometry\_\_Distance\_\_keyword\_\_group.html)
- [geometry!Swift](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group__geometry__Swift__keyword__group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group\_\_geometry\_\_Swift\_\_keyword\_\_group.html)

### Description

Calculates penetration distance of cylindrical wheel into terrain

This model is a reimplementaion of the TerrainPenAnalytic model except that it uses the TerrainPeneration helper class.

from the SurfaceContactModels models library for the xref:Dshellpp\_module simulation framework.

### Model TerrainPenAnalyticCylindricalWheelAlt class details

For more information on the members and functions of this model class, please see [SurfaceContactModels::TerrainPenAnalyticCylindricalWheelAlt model class doxygen documentation](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group__TerrainPenAnalyticCylindricalWheelAlt__group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group\_\_TerrainPenAnalyticCylindricalWheelAlt\_\_group.html)

### Parameters

Name	Type	Size	Quantity	Units	Description
contact_wheel_aligned	int				If 0, the contact frame is aligned with the terrain, else with the wheel
radius	double		Length		Actual wheel radius
width	double		Length		Wheel width

### Flow Outs

Name	Type	Size	Quantity	Units	Description
camber_angle	double		Angle		The wheel tilt angle

## 32.21. SurfaceContactModels::TerrainPenAnalyticPad Sensor Model



This model documentation is auto-generated. Documentation updates should be made to the model's `mdl` file to avoid the changes being lost.

[Doxygen link to all SurfaceContactModels models of Sensor class](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group__Sensor_group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group\_\_Sensor\_group.html)

#### Keywords Doxygen groups

- [Computational](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group__Computational_keyword_group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group\_\_Computational\_keyword\_group.html)
- [Contact](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group__Contact_keyword_group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group\_\_Contact\_keyword\_group.html)
- [Contact!Pad](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group__Contact_Pad_keyword_group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group\_\_Contact\_Pad\_keyword\_group.html)
- [Contact!Terrain](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group__Contact_Terrain_keyword_group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group\_\_Contact\_Terrain\_keyword\_group.html)
- [Distance](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group__Distance_keyword_group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group\_\_Distance\_keyword\_group.html)
- [Distance!Penetration](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group__Distance_Penetration_keyword_group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group\_\_Distance\_Penetration\_keyword\_group.html)
- [Penetration](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group__Penetration_keyword_group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group\_\_Penetration\_keyword\_group.html)
- [Penetration!Distance](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group__Penetration_Distance_keyword_group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group\_\_Penetration\_Distance\_keyword\_group.html)
- [Rover](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group__Rover_keyword_group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group\_\_Rover\_keyword\_group.html)
- [Rover!Pad/Soil](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group__Rover_Pad-Soil_keyword_group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group\_\_Rover\_Pad-Soil\_keyword\_group.html)
- [Surface](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group__Surface_keyword_group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group\_\_Surface\_keyword\_group.html)
- [Swift](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group__Swift_keyword_group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group\_\_Swift\_keyword\_group.html)
- [Terrain](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group__Terrain_keyword_group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group\_\_Terrain\_keyword\_group.html)
- [contact](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group__contact_keyword_group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group\_\_contact\_keyword\_group.html)
- [geometry](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group__geometry_keyword_group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group\_\_geometry\_keyword\_group.html)
- [geometry!Distance](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group__geometry_Distance_keyword_group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group\_\_geometry\_Distance\_keyword\_group.html)
- [geometry!Swift](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group__geometry_Swift_keyword_group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group\_\_geometry\_Swift\_keyword\_group.html)

#### Description

Calculates penetration distance of a circular pad shape into terrain

from the SurfaceContactModels models library for the xref:Dshellpp\_module simulation framework.

#### Model TerrainPenAnalyticPad class details

For more information on the members and functions of this model class, please see [SurfaceContactModels::TerrainPenAnalyticPad model class doxygen documentation](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group__TerrainPenAnalyticPad_group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group\_\_TerrainPenAnalyticPad\_group.html)

#### Parameters

Name	Type	Size	Quantity	Units	Description
demSpecName	string				The name of the top level DVar spec with the vehicle's TopoDem pointer.
num_points	int				Number of points to evaluate around pad circumference (>=0)
pad_radius	double		Length		Radius of the pad (m)
patch_resolution	double		Length		Maximum subsampled resolution to use for computing terrain patch statistics

#### Scratch

Name	Type	Size	Quantity	Units	Description
normal	double	3	Dimensionless		scratch normal test
patch_size	double		Length		terrain patch size
point_sep_angle	double		Angle		Angle between successive points on the circumference

#### Flow Ins

Name	Type	Size	Quantity	Units	Description
------	------	------	----------	-------	-------------

Name	Type	Size	Quantity	Units	Description
cohesion	double		Stress		input soil mechanics cohesion parameter
density	double		Density		input soil mechanics density parameter
frictionAngle	double		Angle		input soil mechanics friction parameter

#### Flow Outs

Name	Type	Size	Quantity	Units	Description
cohesion	double		Stress		soil mechanics cohesion parameter
contactLocation	double	3	Length		contact point location (body frame)
density	double		Density		soil mechanics density parameter
frictionAngle	double		Angle		soil mechanics friction parameter
penetrationDistance	double		Length		penetration distance
terrainNormal	double	3	Dimensionless		normal to terrain at contact point (inertial frame)

## 32.22. SurfaceContactModels::TerrainPenAnalyticPadAlt Sensor Model



This model documentation is auto-generated. Documentation updates should be made to the model's `mdl` file to avoid the changes being lost.

[Doxygen link to all SurfaceContactModels models of Sensor class](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group__Sensor__group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group\_\_Sensor\_\_group.html)

#### Keywords Doxygen groups

- [Computational](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group__Computational__keyword__group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group\_\_Computational\_\_keyword\_\_group.html)
- [Contact](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group__Contact__keyword__group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group\_\_Contact\_\_keyword\_\_group.html)
- [Contact!Pad](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group__Contact__Pad__keyword__group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group\_\_Contact\_\_Pad\_\_keyword\_\_group.html)
- [Contact!Terrain](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group__Contact__Terrain__keyword__group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group\_\_Contact\_\_Terrain\_\_keyword\_\_group.html)
- [Distance](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group__Distance__keyword__group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group\_\_Distance\_\_keyword\_\_group.html)
- [Distance!Penetration](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group__Distance__Penetration__keyword__group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group\_\_Distance\_\_Penetration\_\_keyword\_\_group.html)
- [Penetration](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group__Penetration__keyword__group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group\_\_Penetration\_\_keyword\_\_group.html)
- [Penetration!Distance](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group__Penetration__Distance__keyword__group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group\_\_Penetration\_\_Distance\_\_keyword\_\_group.html)
- [Rover](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group__Rover__keyword__group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group\_\_Rover\_\_keyword\_\_group.html)
- [Rover!Pad/Soil](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group__Rover__Pad-Soil__keyword__group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group\_\_Rover\_\_Pad-Soil\_\_keyword\_\_group.html)
- [Surface](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group__Surface__keyword__group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group\_\_Surface\_\_keyword\_\_group.html)
- [Swift](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group__Swift__keyword__group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group\_\_Swift\_\_keyword\_\_group.html)
- [Terrain](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group__Terrain__keyword__group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group\_\_Terrain\_\_keyword\_\_group.html)
- [contact](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group__contact__keyword__group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group\_\_contact\_\_keyword\_\_group.html)
- [geometry](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group__geometry__keyword__group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group\_\_geometry\_\_keyword\_\_group.html)
- [geometry!Distance](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group__geometry__Distance__keyword__group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group\_\_geometry\_\_Distance\_\_keyword\_\_group.html)
- [geometry!Swift](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group__geometry__Swift__keyword__group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group\_\_geometry\_\_Swift\_\_keyword\_\_group.html)

#### Description

Calculates penetration distance of a circular pad shape into terrain

This model is a reimplemention of the TerrainPenAnalyticPad model except that it uses the TerrainPeneration helper class.

from the SurfaceContactModels models library for the xref:Dshellpp\_module simulation framework.

### Model TerrainPenAnalyticPadAlt class details

For more information on the members and functions of this model class, please see [SurfaceContactModels::TerrainPenAnalyticPadAlt model class doxygen documentation](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group__TerrainPenAnalyticPadAlt_group.html) ([https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group\\_\\_TerrainPenAnalyticPadAlt\\_group.html](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group__TerrainPenAnalyticPadAlt_group.html))

### Parameters

Name	Type	Size	Quantity	Units	Description
num_points	int				Number of points to evaluate around pad circumference (>=0)
pad_radius	double		Length		Radius of the pad (m)
pen_offset	double	3	Length		The translational offset from the body frame to the sensing plane (body frame)

## 32.23. SurfaceContactModels::TyreContact Actuator Model



This model documentation is auto-generated. Documentation updates should be made to the model's `mdl` file to avoid the changes being lost.

[Doxygen link to all SurfaceContactModels models of Actuator class](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group__Actuator_group.html) ([https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group\\_\\_Actuator\\_group.html](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group__Actuator_group.html))

### Keywords Doxygen groups

- [Constraint](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group__Constraint_keyword_group.html) ([https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group\\_\\_Constraint\\_keyword\\_group.html](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group__Constraint_keyword_group.html))
- [Constraint!Soft](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group__Constraint_Soft_keyword_group.html) ([https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group\\_\\_Constraint\\_Soft\\_keyword\\_group.html](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group__Constraint_Soft_keyword_group.html))
- [Contact](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group__Contact_keyword_group.html) ([https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group\\_\\_Contact\\_keyword\\_group.html](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group__Contact_keyword_group.html))
- [Contact!Terrain](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group__Contact_Terrain_keyword_group.html) ([https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group\\_\\_Contact\\_Terrain\\_keyword\\_group.html](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group__Contact_Terrain_keyword_group.html))
- [Contact!Wheel](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group__Contact_Wheel_keyword_group.html) ([https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group\\_\\_Contact\\_Wheel\\_keyword\\_group.html](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group__Contact_Wheel_keyword_group.html))
- [Crossley](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group__Crossley_keyword_group.html) ([https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group\\_\\_Crossley\\_keyword\\_group.html](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group__Crossley_keyword_group.html))
- [Friction](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group__Friction_keyword_group.html) ([https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group\\_\\_Friction\\_keyword\\_group.html](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group__Friction_keyword_group.html))
- [Rover](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group__Rover_keyword_group.html) ([https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group\\_\\_Rover\\_keyword\\_group.html](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group__Rover_keyword_group.html))
- [Rover!Dynamics](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group__Rover_Dynamics_keyword_group.html) ([https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group\\_\\_Rover\\_Dynamics\\_keyword\\_group.html](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group__Rover_Dynamics_keyword_group.html))
- [Rover!Wheel/Soil](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group__Rover_Wheel-Soil_keyword_group.html) ([https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group\\_\\_Rover\\_Wheel-Soil\\_keyword\\_group.html](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group__Rover_Wheel-Soil_keyword_group.html))
- [Soil](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group__Soil_keyword_group.html) ([https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group\\_\\_Soil\\_keyword\\_group.html](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group__Soil_keyword_group.html))
- [Spring/Damper](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group__Spring-Damper_keyword_group.html) ([https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group\\_\\_Spring-Damper\\_keyword\\_group.html](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group__Spring-Damper_keyword_group.html))
- [Spring/Damper!Hunt](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group__Spring-Damper_Hunt_keyword_group.html) ([https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group\\_\\_Spring-Damper\\_Hunt\\_keyword\\_group.html](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group__Spring-Damper_Hunt_keyword_group.html))
- [Spring/Damper!Nonlinear](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group__Spring-Damper_Nonlinear_keyword_group.html) ([https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group\\_\\_Spring-Damper\\_Nonlinear\\_keyword\\_group.html](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group__Spring-Damper_Nonlinear_keyword_group.html))
- [Terrain](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group__Terrain_keyword_group.html) ([https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group\\_\\_Terrain\\_keyword\\_group.html](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group__Terrain_keyword_group.html))
- [Terrain!Contact](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group__Terrain_Contact_keyword_group.html) ([https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group\\_\\_Terrain\\_Contact\\_keyword\\_group.html](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group__Terrain_Contact_keyword_group.html))
- [Wheel](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group__Wheel_keyword_group.html) ([https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group\\_\\_Wheel\\_keyword\\_group.html](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group__Wheel_keyword_group.html))
- [Wheel!Contact](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group__Wheel_Contact_keyword_group.html) ([https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group\\_\\_Wheel\\_Contact\\_keyword\\_group.html](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group__Wheel_Contact_keyword_group.html))
- [Wheel!Soil](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group__Wheel_Soil_keyword_group.html) ([https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group\\_\\_Wheel\\_Soil\\_keyword\\_group.html](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group__Wheel_Soil_keyword_group.html))
- [contact](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group__contact_keyword_group.html) ([https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group\\_\\_contact\\_keyword\\_group.html](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group__contact_keyword_group.html))
- [mechanics](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group__mechanics_keyword_group.html) ([https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group\\_\\_mechanics\\_keyword\\_group.html](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group__mechanics_keyword_group.html))

### Description

Applies compliant contact force to a body based on Magic Formula Tyre Model.

This model computes and applies wheel/soil contact forces treating the Tire as a flexible body and uses the interpolated data based on various tire parameters Created by Adam Ryason and Calvin Kuo

from the SurfaceContactModels models library for the xref:Dshellpp\_module simulation framework.

### Model TyreContact class details

For more information on the members and functions of this model class, please see [SurfaceContactModels::TyreContact model class doxygen documentation](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group__TyreContact_group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/SurfaceContactModels/html/group\_\_TyreContact\_group.html)

## Enums

### ForceModel Enum

In the TyreContact model definition, the original enum ForceModel is defined as:

```
enum ForceModel
{
    nonlinear = 0,
    linear = 1
};
```

In the C model code, these enum values can be accessed by using the follow C enum definitions:

```
/// TyreContactForceModel enum
enum TyreContactForceModel
{
    TYRE_CONTACT_FORCE_MODEL_NONLINEAR = 0,
    TYRE_CONTACT_FORCE_MODEL_LINEAR = 1
};
```

### ContactType Enum

In the TyreContact model definition, the original enum ContactType is defined as:

```
enum ContactType
{
    NoContact = 0,
    Sliding = 1,
    Rolling = 2
};
```

In the C model code, these enum values can be accessed by using the follow C enum definitions:

```
/// TyreContactContactType enum
enum TyreContactContactType
{
    TYRE_CONTACT_CONTACT_TYPE_NO_CONTACT = 0,
    TYRE_CONTACT_CONTACT_TYPE_SLIDING = 1,
    TYRE_CONTACT_CONTACT_TYPE_ROLLING = 2
};
```

## Parameters

Name	Type	Size	Quantity	Units	Description
alpha	double	2	LinearSpringDamping		alpha for normal and tangent directions
demSpecName	string				The name of the top level DVar spec with the vehicle's TopoDem pointer.
kp	double	2	LinearSpringStiffness		Spring constant for compliant contact in normal and tangent directions
maxForce	double		Force		Maximum allowed applied force - prevents numeric problems
n	double	2	Dimensionless		exponential for deflection term for normal and tangent directions
normalForceModel	ForceModel				Normal force model (default is nonlinear)
tangentForceModel	ForceModel				Tangent force model (default is linear)
tangent_x	double	3	Dimensionless		First tangent direction (inertial coordinates).
tangent_y	double	3	Dimensionless		Second tangent direction (inertial coordinates, orthogonal to tanget_x)



Name	Type	Size	Quantity	Units	Description
wheel_radius	double		Length		wheel radius (used to compute contact patch area)

#### Scratch

Name	Type	Size	Quantity	Units	Description
ContProjectionVelocity	double		Velocity		Projection of contact point velocity along the Directional Velocity
ContVelocity	double	3	Velocity		Velocity of contact point (node frame)
DirectionMag	double		Velocity		Magnitude of the Direction Vector (node coords)
HubLinearVelocity	double	3	Velocity		Linear velocity of Hub / Wheel Center (node coordinates)
HubMag	double		Velocity		Magnitude of the Hub velocity vector
HubProjeNormVelocity	double	3	Velocity		Projection of tire velocity along the Rotational Vector
HubProjectionVelocity	double	3	Velocity		Projection of tire velocity along the Directional Velocity
HubProjectionVelocityX	double		Velocity		Projection of tire velocity along the Directional Velocity
HubProjectionVelocityY	double		Velocity		Projection of tire velocity along the Directional Velocity
Latforce	double		Force		Lateral Force scalar
Longforce	double		Force		Longitudinal Force scalar
NodeVelocity	double	3	Velocity		Velocity of contact point in Inertial
NormalMag	double		Force		Magnitude of the Normal Force (node coords)
NormalO	double		Force		sum of normal forces used for calculating the average force
contactLocationInertial	double	3	Length		Location of contact (inertial frame)
count	int				counts the step since last steady state (normal force)
force	double	3	Force		Contact constraint force (node frame)
latslip	double		Angle		Lateral Slip Ratio
nodeAngularVelocity	double	3	AngularVelocity		Angular velocity of contact node (node coordinates)
nodeLinearVelocity	double	3	Velocity		Linear velocity of contact node (node coordinates)
nodeframeNormal	double	3	Dimensionless		normal vector written in node frame

Name	Type	Size	Quantity	Units	Description
normalForce	double	3	Force		Force of normal direction spring/damper (node frame)
normalVelocity	double	3	Velocity		Velocity of point in normal direction
okay	bool				sum of normal forces used for calculating the average force
penetrationDistance	double		Length		Penetration distance
slipVelocity	double		Velocity		Slip Velocity of the Contact Point
slope	double		Force		sum of normal forces used for calculating the average force
tangentForce	double	3	Force		Force of tangent direction spring/damper (node frame)
tangentX	double	3	Dimensionless		Rotated params.tangent_x to match orientation of current contact normal
tangentX_inertial	double	3	Dimensionless		Rotated params.tangent_x to match orientation of current contact normal (in inertial coordinates)
tangentY	double	3	Dimensionless		Rotated params.tangent_y to match orientation of current contact normal
tangentY_inertial	double	3	Dimensionless		Rotated params.tangent_y to match orientation of current contact normal (in inertial coordinates)
tangent_z	double	3	Dimensionless		Nominal normal direction (inertial coordinates)
torque	double	3	Torque		Contact constraint torque (node frame)
unitDir	double	3	Dimensionless		Direction of the Tire (node frame)
unitHub	double	3	Dimensionless		Hub Vel unit vector
unitWheel	double	3	Dimensionless		Rotation unit vector
wheelRate	double	3	AngularVelocity		Wheel rate about its spin axis (node coordinates)
wheelVector	double	3	Dimensionless	3	Wheel Vector (node coordinates)
wheelslip	double		Dimensionless		Longitudinal Slip Ratio

#### Flow Ins

Name	Type	Size	Quantity	Units	Description
cohesion	double		Pressure		Soil cohesion (kPa)
contactLocation	double	3	Length		Location of contact relative to body node (body frame)
density	double		Density		soil density (kg/m <sup>3</sup> )
gravity	double	3	Acceleration		Gravity vector

Name	Type	Size	Quantity	Units	Description
normalDirection	double	3	Dimensionless		Normal vector for contact (inertial frame)
penetrationDistance	double		Length		penetration distance
phi	double		Angle		internal friction angle (radians)

#### Flow Outs

Name	Type	Size	Quantity	Units	Description
currCState	double	2	Length		current cont state (used for testing)

## 33. VehicleModels Dshell model library

### Doxygen reference to Models in VehicleModels by Type

- [Actuators](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/VehicleModels/html/group__Actuator_group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/VehicleModels/html/group\_\_Actuator\_group.html)
- [Sensors](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/VehicleModels/html/group__Sensor_group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/VehicleModels/html/group\_\_Sensor\_group.html)
- [Motors](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/VehicleModels/html/group__Motor_group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/VehicleModels/html/group\_\_Motor\_group.html)
- [Encoders](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/VehicleModels/html/group__Encoder_group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/VehicleModels/html/group\_\_Encoder\_group.html)
- [Flows](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/VehicleModels/html/group__Flow_group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/VehicleModels/html/group\_\_Flow\_group.html)
- [Continuous](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/VehicleModels/html/group__ContStates_group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/VehicleModels/html/group\_\_ContStates\_group.html)

### 33.1. VehicleModels::FixedThruster Actuator Model



This model documentation is auto-generated. Documentation updates should be made to the model's `mdl` file to avoid the changes being lost.

[Doxygen link to all VehicleModels models of Actuator class](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/VehicleModels/html/group__Actuator_group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/VehicleModels/html/group\_\_Actuator\_group.html)

#### Keywords Doxygen groups

- [Force](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/VehicleModels/html/group__Force_keyword_group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/VehicleModels/html/group\_\_Force\_keyword\_group.html)
- [Fuel](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/VehicleModels/html/group__Fuel_keyword_group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/VehicleModels/html/group\_\_Fuel\_keyword\_group.html)
- [Resource](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/VehicleModels/html/group__Resource_keyword_group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/VehicleModels/html/group\_\_Resource\_keyword\_group.html)
- [Thruster](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/VehicleModels/html/group__Thruster_keyword_group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/VehicleModels/html/group\_\_Thruster\_keyword\_group.html)
- [Thruster!Constant](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/VehicleModels/html/group__Thruster_Constant_keyword_group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/VehicleModels/html/group\_\_Thruster\_Constant\_keyword\_group.html)
- [Thruster!On/Off](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/VehicleModels/html/group__Thruster_On-Off_keyword_group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/VehicleModels/html/group\_\_Thruster\_On-Off\_keyword\_group.html)
- [consumption](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/VehicleModels/html/group__consumption_keyword_group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/VehicleModels/html/group\_\_consumption\_keyword\_group.html)
- [model](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/VehicleModels/html/group__model_keyword_group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/VehicleModels/html/group\_\_model\_keyword\_group.html)
- [model!Fuel](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/VehicleModels/html/group__model_Fuel_keyword_group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/VehicleModels/html/group\_\_model\_Fuel\_keyword\_group.html)

#### Description

Models a thruster with constant force output, ON/OFF commands and no delay. This thruster tracks fuel consumption and outputs total fuel burned.

The FixedThruster model accepts an on(1) or off(0) flowIn, and has parameters for thruster force level, direction (orientation) and specific impulse. The thrust commands are executed without delay, and there is no ramp up or ramp down time on the thrust output. Fuel consumption is calculated based on the specific impulse. The fuel\_used flowOut should be tied to a FuelTank model in order to change the mass and inertial properties for the s/c. The fuel\_remaining flowOut of the FuelTank model should be tied to the fuel\_remaining flowIn of this model. This allows the thruster to know when all available fuel has been used.

from the VehicleModels models library for the xref:Dshellpp\_module simulation framework.

#### Model FixedThruster class details

For more information on the members and functions of this model class, please see [VehicleModels::FixedThruster model class doxygen documentation](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/VehicleModels/html/group__FixedThruster_group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/VehicleModels/html/group\_\_FixedThruster\_group.html)

#### Parameters

Name	Type	Size	Quantity	Units	Description
isp	double		SpecificImpulse		specific impulse
level	double		Force		amount of force generated by this thruster

#### Flow Ins

Name	Type	Size	Quantity	Units	Description
on_off	double		Dimensionless		command to turn thruster on (!=0) or off (0)

### 33.2. VehicleModels::FuelManifold Actuator Model



This model documentation is auto-generated. Documentation updates should be made to the model's `mdl` file to avoid the changes being lost.

[Doxygen link to all VehicleModels models of Actuator class](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/VehicleModels/html/group__Actuator_group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/VehicleModels/html/group\_\_Actuator\_group.html)

### Description

Models a simple fuel manifold.

The FuelManifold model accepts two inputs which denotes the amount of fuel burned/drawn from a set of sinks and the amount of fuel available from a set of sources. The inputs are variable sized so that several thrusters/sinks and several tanks can share a single FuelManifold model. It outputs two array of doubles: `fuel_burned_output`, the sum of the input fuel draws divided by the size of `flowOuts()`→`fuel_burned`, and `fuel_remaining_output`, the sum of input fuel availability replicated on all slices. Note: when the `fuel_remaining` is zero on any line the draw rates for the other lines will be adjusted by their ratios to give a total flow rate of 1.0.

from the VehicleModels models library for the xref:Dshellpp\_module simulation framework.

### Model FuelManifold class details

For more information on the members and functions of this model class, please see [VehicleModels::FuelManifold model class doxygen documentation](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/VehicleModels/html/group__FuelManifold_group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/VehicleModels/html/group\_\_FuelManifold\_group.html)

### Parameters

Name	Type	Size	Quantity	Units	Description
<code>fuel_draw_weights</code>	double	*	Dimensionless		Fuel draw weighting for the <code>fuel_burned_output</code> lines

### Discrete States

Name	Type	Size	Quantity	Units	Description
<code>current_fuel_draw_weights</code>	double	*	Dimensionless		The current fuel draw weighting for the <code>fuel_burned_output</code> lines
<code>fuel_burned_output</code>	double	*	Mass		The total amount of fuel burned for each tank as far as this manifold knows
<code>num_burned_inputs</code>	unsigned int (uint)				size of <code>flowIns()</code> → <code>fuel_burned_input</code>
<code>num_burned_outputs</code>	unsigned int (uint)				size of <code>flowOuts()</code> → <code>fuel_burned_output</code>
<code>total_fuel_burned</code>	double		Mass		total fuel burned
<code>total_fuel_remaining</code>	double		Mass		total remaining fuel

### Flow Ins

Name	Type	Size	Quantity	Units	Description
<code>fuel_burned_input</code>	double	*	Unspecified		Total amount of fuel burned from each input
<code>fuel_remaining_input</code>	double	*	Mass		Total amount of remaining fuel from each input

### Flow Outs

Name	Type	Size	Quantity	Units	Description
<code>fuel_burned_output</code>	double	*	Unspecified		Sum of thruster fuel burned divided by <code>num_outputs</code> multiplied by <code>fuel_draw_weighting</code>
<code>fuel_remaining_output</code>	double		Mass		Sum of fuel remaining in tanks connected to manifold

### 33.3. VehicleModels::FuelTank Actuator Model



This model documentation is auto-generated. Documentation updates should be made to the model's `mdl` file to avoid the changes being lost.

[Doxygen link to all VehicleModels models of Actuator class](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/VehicleModels/html/group__Actuator__group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/VehicleModels/html/group\_\_Actuator\_\_group.html)

#### Description

Models a simple fuel tank.

The FuelTank model accepts a single input which denotes the amount of fuel burned. This input is variable sized so that several thrusters can share a single FuelTank model. It makes Darts calls to update the mass and inertial properties of the spacecraft based on the amount of fuel remaining in the tank. It outputs the remaining fuel mass (this output can be tied into the thruster models). Parameters denote the starting amount of fuel in the tank and the inertia matrix and CM location for the tank when empty. As fuel is removed from the tank the inertia matrix and CM location are interpolated (linearly) between the starting value and the given value for the empty tank.

from the VehicleModels models library for the xref:Dshellpp\_module simulation framework.

#### Model FuelTank class details

For more information on the members and functions of this model class, please see [VehicleModels::FuelTank model class doxygen documentation](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/VehicleModels/html/group__FuelTank__group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/VehicleModels/html/group\_\_FuelTank\_\_group.html)

#### Enums

##### InertiaOriginType Enum

In the FuelTank model definition, the original enum InertiaOriginType is defined as:

```
enum InertiaOriginType
{
    BODY_ORIGIN = 0,
    CENTER_OF_MASS = 1
};
```

In the C model code, these enum values can be accessed by using the follow C enum definitions:

```
/// FuelTankInertiaOriginType enum
enum FuelTankInertiaOriginType
{
    FUEL_TANK_INERTIA_ORIGIN_TYPE_BODY_ORIGIN = 0,
    FUEL_TANK_INERTIA_ORIGIN_TYPE_CENTER_OF_MASS = 1
};
```

#### Parameters

Name	Type	Size	Quantity	Units	Description
empty_CM	double	3	Length		CM location for empty tank
empty_inertia	double	9	MomentsOfInertia		Inertia matrix for empty tank
empty_mass	double		Mass		Mass of fuel tank when empty
inertia_data_origin	InertiaOriginType				Origin for for inertia matrix data
initial_CM	double	3	Length		CM location for full tank
initial_inertia	double	9	MomentsOfInertia		Inertia matrix for full tank
num_thrusters	unsigned int (uint)				Number of thrusters using this fuel tank
starting_fuel	double		Mass		Initial mass of fuel in tank

#### Discrete States

Name	Type	Size	Quantity	Units	Description
fuel_remaining	double		Mass		remaining fuel
sum_fuel_consumed	double		Mass		total fuel consumed

## Flow Ins

Name	Type	Size	Quantity	Units	Description
fuel_burned	double	*	Mass		Total amount of consumed fuel

## Flow Outs

Name	Type	Size	Quantity	Units	Description
fuel_remaining	double		Mass		Fuel remaining in tank

## 33.4. VehicleModels::FuelTankDryMassWithTableLookup Actuator Model



This model documentation is auto-generated. Documentation updates should be made to the model's `mdl` file to avoid the changes being lost.

[Doxygen link to all VehicleModels models of Actuator class \(https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/VehicleModels/html/group\\_\\_Actuator\\_group.html\)](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/VehicleModels/html/group__Actuator_group.html)

### Description

Models a simple fuel tank. Mass and inertia are interpolated from a user-supplied table

Based on EDLR4Models/FuelTankWithTableLookup with the addition of the `update_mass_properties` parameter to enable/disable updating the body mass and inertia. \* The tableFile containing the mass/inertia should have this format: 1. Text file with 13 columns. Each column is a floating point number. Lines beginning with a non-numeric character are treated as comments. 2. First column contains mass values (the independent value). 3. 2nd to 10th columns contain the inertia values. 11th to 13th columns contain the CM location. 2nd column contains inertia[0][0] value 3rd column contains inertia[0][1] value 4th column contains inertia[0][2] value 5th column contains inertia[1][0] value 6th column contains inertia[1][1] value 7th column contains inertia[1][2] value 8th column contains inertia[2][0] value 9th column contains inertia[2][1] value 10th column contains inertia[2][2] value 11th column contains CM\_loc[0] value 12th column contains CM\_loc[1] value 13th column contains CM\_loc[2] value Interpolation is done through the `mathc90 dilup()` function with `ndeg=3` and `lup=1` (see `mathc90/doc/ch12-01.pdf` for dilup documentation). There is no extrapolation; end points are used if dilup reports extrapolation is necessary.

from the VehicleModels models library for the xref:Dshellpp\_module simulation framework.

### Model FuelTankDryMassWithTableLookup class details

For more information on the members and functions of this model class, please see [VehicleModels::FuelTankDryMassWithTableLookup model class doxygen documentation \(https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/VehicleModels/html/group\\_\\_FuelTankDryMassWithTableLookup\\_group.html\)](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/VehicleModels/html/group__FuelTankDryMassWithTableLookup_group.html)

### Enums

#### InertiaOriginType Enum

In the FuelTankDryMassWithTableLookup model definition, the original enum InertiaOriginType is defined as:

```
enum InertiaOriginType
{
    BODY_ORIGIN = 0,
    CENTER_OF_MASS = 1
};
```

In the C model code, these enum values can be accessed by using the follow C enum definitions:

```
/// FuelTankDryMassWithTableLookupInertiaOriginType enum
enum FuelTankDryMassWithTableLookupInertiaOriginType
{
    FUEL_TANK_DRY_MASS_WITH_TABLE_LOOKUP_INERTIA_ORIGIN_TYPE_BODY_ORIGIN = 0,
    FUEL_TANK_DRY_MASS_WITH_TABLE_LOOKUP_INERTIA_ORIGIN_TYPE_CENTER_OF_MASS = 1
};
```

### Parameters

Name	Type	Size	Quantity	Units	Description
empty_CM	double	3	Length		CM location for empty tank
empty_inertia	double	9	MomentsOfInertia		Inertia matrix for empty tank (about CM)
empty_mass	double		Mass		Mass of fuel tank when empty

Name	Type	Size	Quantity	Units	Description
inertia_data_origin	InertiaOriginType				Origin for for inertia matrix data
initial_CM	double	3	Length		CM location for full tank
initial_inertia	double	9	MomentsOfInertia		Inertia matrix for full tank (about CM)
num_thrusters	unsigned int (uint)				Number of thrusters using this fuel tank
starting_fuel	double		Mass		Initial mass of fuel in tank
tableFile	string				File containing mass and inertia table

#### Scratch

Name	Type	Size	Quantity	Units	Description
current_CM	double	3	Length		Current CM from 'tableFile' lookup
current_inertia	double	9	MomentsOfInertia		Current inertia from 'tableFile' lookup
current_mass	double		Mass		Current mass from 'tableFile' lookup

#### Discrete States

Name	Type	Size	Quantity	Units	Description
dry_mass_consumed	double		Mass		Total amount of consumed dry mass
dry_mass_remaining	double		Mass		Remaining dry mass
fuel_remaining	double		Mass		remaining fuel
sum_fuel_consumed	double		Mass		total fuel consumed

#### Flow Ins

Name	Type	Size	Quantity	Units	Description
dry_mass_consumed	double		Mass		Total amount of dry mass consumed
fuel_burned	double	*	Mass		Total amount of consumed fuel

#### Flow Outs

Name	Type	Size	Quantity	Units	Description
fuel_remaining	double		Mass		Fuel remaining in tank

### 33.5. VehicleModels::FuelTankWithTableLookup Actuator Model



This model documentation is auto-generated. Documentation updates should be made to the model's `mdl` file to avoid the changes being lost.

[Doxygen link to all VehicleModels models of Actuator class](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/VehicleModels/html/group__Actuator_group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/VehicleModels/html/group\_\_Actuator\_group.html)

#### Description

Models a simple fuel tank. Mass and inertia are interpolated from a user-supplied table



The FuelTank model accepts a single input which denotes the amount of fuel burned. This input is variable sized so that several thrusters can share a single FuelTank model. It makes Darts calls to update the mass and inertial properties of the spacecraft based on a user-supplied table. It outputs the remaining fuel mass (this output can be tied into the thruster models). Parameters denote the starting amount of fuel in the tank and the inertia matrix and CM location for the tank when empty. As fuel is removed from the tank the inertia matrix and CM location are interpolated (linearly) between the starting value and the given value for the empty tank. The file containing the mass/inertia should have this format: 1. Text file with 13 columns. Each column is a floating point number: Lines beginning with a non-numeric character are treated as comments. 2. First column contains mass values (the independent value). (This is the total fuel tank mass including dry tank and fuel.) 3. 2nd to 10th columns contain the inertia values. 11th to 13th columns contain the CM location. 2nd column contains inertia[0][0] value 3rd column contains inertia[0][1] value 4th column contains inertia[0][2] value 5th column contains inertia[1][0] value 6th column contains inertia[1][1] value 7th column contains inertia[1][2] value 8th column contains inertia[2][0] value 9th column contains inertia[2][1] value 10th column contains inertia[2][2] value 11th column contains CM\_loc[0] value 12th column contains CM\_loc[1] value 13th column contains CM\_loc[2] value Interpolation is done through the mathc90 dilup() function with ndeg=3 and lup=1 (see mathc90/doc/ch12-01.pdf for dilup documentation). There is no extrapolation; end points are used if dilup reports extrapolation is necessary.

from the VehicleModels models library for the xref:Dshellpp\_module simulation framework.

#### Model FuelTankWithTableLookup class details

For more information on the members and functions of this model class, please see [VehicleModels::FuelTankWithTableLookup model class doxygen documentation](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/VehicleModels/html/group__FuelTankWithTableLookup_group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/VehicleModels/html/group\_\_FuelTankWithTableLookup\_group.html)

#### Enums

#### Parameters

Name	Type	Size	Quantity	Units	Description
tableFile	string				File containing mass and inertia table

#### Scratch

Name	Type	Size	Quantity	Units	Description
current_CM	double	3	Length		Current CM from 'tableFile' lookup
current_inertia	double	9	MomentsOfInertia		Current inertia from 'tableFile' lookup
current_mass	double		Mass		Current mass from 'tableFile' lookup

### 33.6. VehicleModels::PulsedThruster Actuator Model



This model documentation is auto-generated. Documentation updates should be made to the model's `mdl` file to avoid the changes being lost.

[Doxygen link to all VehicleModels models of Actuator class](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/VehicleModels/html/group__Actuator_group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/VehicleModels/html/group\_\_Actuator\_group.html)

#### Description

from the VehicleModels models library for the xref:Dshellpp\_module simulation framework.

#### Model PulsedThruster class details

For more information on the members and functions of this model class, please see [VehicleModels::PulsedThruster model class doxygen documentation](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/VehicleModels/html/group__PulsedThruster_group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/VehicleModels/html/group\_\_PulsedThruster\_group.html)

#### Parameters

Name	Type	Size	Quantity	Units	Description
isp_gfactor	double		Acceleration		acceleration of gravity used for computing fuel consumption
min_commanded_time	double		Time		min commanded time for thruster

#### Discrete States

Name	Type	Size	Quantity	Units	Description
off_time	double		Time		duration from beginning of I/O to off
true_thrusting	double		Flag		Decides if the craft is thrusting or not

#### Flow Ins

Name	Type	Size	Quantity	Units	Description
on_time_cmd	double		Time		command to turn thruster on 0.0 is off to time duration

### 33.7. VehicleModels::PulsedThrusterBlowdown Actuator Model



This model documentation is auto-generated. Documentation updates should be made to the model's `mdl` file to avoid the changes being lost.

[Doxygen link to all VehicleModels models of Actuator class](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/VehicleModels/html/group__Actuator_group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/VehicleModels/html/group\_\_Actuator\_group.html)

#### Description

Models a pulsed thruster with a thrust blowdown model for reduced thrust and ISP with decreasing fuel tank pressure.

Similar to PulsedThruster, the model checks whether a thrust command is currently active and with the ON/OFF commands from the FixedThruster class. It uses an isothermal ( $\gamma = 1.0$ ) or adiabatic model ( $\gamma > 1.0$ ) to determine the internal tank pressure, and deduces the changing pulse thrust level and ISP at every IO step. The corresponding thrust levels and ISP are used for the computation of thrust forces in the ThrusterBase class.

from the VehicleModels models library for the xref:Dshellpp\_module simulation framework.

#### Model PulsedThrusterBlowdown class details

For more information on the members and functions of this model class, please see [VehicleModels::PulsedThrusterBlowdown model class doxygen documentation](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/VehicleModels/html/group__PulsedThrusterBlowdown_group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/VehicleModels/html/group\_\_PulsedThrusterBlowdown\_group.html)

#### Parameters

Name	Type	Size	Quantity	Units	Description
V_tank	double		Volume		structural tank volume (not fuel volume)
fuel_density	double		Density		density of the propellant
fuel_mass_init	double	*	Mass		initial propellant mass per tank
gamma	double		Dimensionless		Isotropic coefficient for fuel vapor ( $c_p/c_v$ )
isp_C	double		Dimensionless		Multiplier coefficient for isp blowdown equation
isp_exp	double		Dimensionless		Exponential parameter for isp blowdown equation
level_C	double		Dimensionless		Multiplier coefficient for thrust blowdown equation
level_exp	double		Dimensionless		Exponential parameter for thrust blowdown equation
num_tanks	int				Number of tanks a thruster is using (define fuel remaining in one tank)
p_tank_init	double		Pressure		initial pressure in corresponding thruster fuel tank

## Discrete States

Name	Type	Size	Quantity	Units	Description
isp	double		SpecificImpulse		Maximum specific impulse calculated according to blowdown model
level	double		Force		Maximum thrust force generated by the thruster according to blowdown model
p_tank	double	*	Pressure		pressure in corresponding thruster fuel tanks

## Flow Ins

Name	Type	Size	Quantity	Units	Description
fuel_mass_linked_tanks	double	*	Mass		Mass of fuel remaining in tanks that are connected to thrust, is vector.

## Flow Outs

Name	Type	Size	Quantity	Units	Description
isp_out	double		SpecificImpulse		Maximum specific impulse calculated according to blowdown model
level_out	double		Force		Maximum constant thrust force generated by the thruster according to blowdown model

## 33.8. VehicleModels::SimpleTurn Actuator Model



This model documentation is auto-generated. Documentation updates should be made to the model's `mdl` file to avoid the changes being lost.

[Doxygen link to all VehicleModels models of Actuator class](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/VehicleModels/html/group__Actuator_group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/VehicleModels/html/group\_\_Actuator\_group.html)

### Description

This model applies forces in a specified direction to simulate a spacecraft force controller. It also controls the prescribed attitude degrees of freedom using a PD controller.

This model attempts to apply forces in the specified direction at the desired throttle setting. The force could be either be applied in the inertial direction or along the body-fixed thrust axis. The former corresponds to an ideal attitude control that instantaneously turns the spacecraft. The model also computes angular accelerations to align the local thrust axis with the specified thrust direction. Thus both forces and attitude are controlled although in an idealized and somewhat independent fashion.

from the VehicleModels models library for the xref:Dshellpp\_module simulation framework.

### Model SimpleTurn class details

For more information on the members and functions of this model class, please see [VehicleModels::SimpleTurn model class doxygen documentation](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/VehicleModels/html/group__SimpleTurn_group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/VehicleModels/html/group\_\_SimpleTurn\_group.html)

### Enums

#### ThrustDirectionMode Enum

In the SimpleTurn model definition, the original enum ThrustDirectionMode is defined as:

```
enum ThrustDirectionMode
{
    BODYFIXED = 0,
    INERTIAL = 1
};
```

In the C model code, these enum values can be accessed by using the follow C enum definitions:

```

/// SimpleTurnThrustDirectionMode enum
enum SimpleTurnThrustDirectionMode
{
    SIMPLE_TURN_THRUST_DIRECTION_MODE_BODYFIXED = 0,
    SIMPLE_TURN_THRUST_DIRECTION_MODE_INERTIAL = 1
};

```

### AccelerationFrame Enum

In the SimpleTurn model definition, the original enum AccelerationFrame is defined as:

```

enum AccelerationFrame
{
    BODY = 0,
    INERTIAL = 1
};

```

In the C model code, these enum values can be accessed by using the follow C enum definitions:

```

/// SimpleTurnAccelerationFrame enum
enum SimpleTurnAccelerationFrame
{
    SIMPLE_TURN_ACCELERATION_FRAME_BODY = 0,
    SIMPLE_TURN_ACCELERATION_FRAME_INERTIAL = 1
};

```

### AttitudeControlMode Enum

In the SimpleTurn model definition, the original enum AttitudeControlMode is defined as:

```

enum AttitudeControlMode
{
    PDCONTROL = 0,
    OPENLOOP = 1
};

```

In the C model code, these enum values can be accessed by using the follow C enum definitions:

```

/// SimpleTurnAttitudeControlMode enum
enum SimpleTurnAttitudeControlMode
{
    SIMPLE_TURN_ATTITUDE_CONTROL_MODE_PDCONTROL = 0,
    SIMPLE_TURN_ATTITUDE_CONTROL_MODE_OPENLOOP = 1
};

```

### Parameters

Name	Type	Size	Quantity	Units	Description
accFrame	AccelerationFrame				Frame in which acceleration commands are specified.
burns_fuel	int				flag to turn on (1) or off (0) mass/fuel consumption
cant_angle	double		Angle		Engine cant angle (radians)
controlMode	AttitudeControlMode				Control mode for angular motion.
forceMode	ThrustDirectionMode				Direction mode for force application.
isp_max	double		SpecificImpulse		maximum specific impulse
isp_min	double		SpecificImpulse		minimum specific impulse
local_thrust_axis	double	3	Dimensionless		Local thrust axis (unit vector, body frame).
max_angular_accel	double	1	AngularAcceleration		Maximum prescribed angular acceleration
max_level	double		Force		Maximum thrust force magnitude/engine (newtons).

Name	Type	Size	Quantity	Units	Description
min_level	double		Force		Minimum thrust force magnitude/engine (newtons).
num_thr	int				Number of descent engines
omega_gain	double		RotationalSpringDamping		Damping term (greater than zero) (gain on angular velocity)
planetOmega	double	3	AngularVelocity		Planet rotation rate in the PCR frame.
planetPosition	double	3	Length		Position of planetary center in inertial frame.
theta_gain	double		RotationalSpringStiffness		Proportional gain based on angle between desired and actual thrust direction (greater than zero)
velocity_cutoff	double		Velocity		Relative velocity level below which actuator becomes inactive (m/s)

#### Scratch

Name	Type	Size	Quantity	Units	Description
accel	double	6	Mixed		Prescribed accelerations for attitude control
angle_error	double		Angle		Error between desired angle direction and actual direction.
body_pos	double	3	Length		Position of body wrt planetary center (inertial frame).
body_quat	double	4	Quaternion		Attitude of body wrt inertial frame.
force	double	3	Force		Applied force vector (body frame)
gr_omega	double	3	Velocity	3	ground relative angular velocity (body frame)
gr_vel	double	3	Velocity		ground relative linear velocity (body frame)
neg_gr_vel	double	3	Velocity		negative of ground relative linear velocity (body frame)
omegaB	double	3	AngularVelocity		angular velocity of body (body frame).
vell	double	3	Velocity		absolute linear velocity of body (inertial frame).

#### Discrete States

Name	Type	Size	Quantity	Units	Description
cur_coeff	double		Dimensionless		Current fraction of thrust (from 0 to 1)
cur_isp	double		SpecificImpulse		Current isp
cur_thrust	double		Force		Current thrust

Name	Type	Size	Quantity	Units	Description
velocity_cutoff_activated	int				Whether velocity cutoff was activated

#### Continuous States

Name	Type	Size	Quantity	Units	Description
fuel_used	double		Mass		State to track fuel consumption of thruster

#### Flow Ins

Name	Type	Size	Quantity	Units	Description
accdir	double	3	Dimensionless		Vector in which force is to be applied.
angacc	double	3	AngularAcceleration		Vector of angular acc used in open-loop control.
throttle	double		Dimensionless		Throttle: 0 gives min_level, 1 (or more) gives max_level. Force is linear between throttle values of 0 and 1. Negative throttle settings are clipped to min_level. Throttle inputs greater than 1 are clipped to max_level.

#### Flow Outs

Name	Type	Size	Quantity	Units	Description
fuel_used	double		Mass		total fuel used by thruster

### 33.9. VehicleModels::ThrottledProfileThruster Actuator Model



This model documentation is auto-generated. Documentation updates should be made to the model's `mdl` file to avoid the changes being lost.

[Doxygen link to all VehicleModels models of Actuator class](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/VehicleModels/html/group__Actuator_group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/VehicleModels/html/group\_\_Actuator\_group.html)

#### Keywords Doxygen groups

- [Fuel](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/VehicleModels/html/group__Fuel_keyword_group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/VehicleModels/html/group\_\_Fuel\_keyword\_group.html)
- [Resource](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/VehicleModels/html/group__Resource_keyword_group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/VehicleModels/html/group\_\_Resource\_keyword\_group.html)
- [Thruster](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/VehicleModels/html/group__Thruster_keyword_group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/VehicleModels/html/group\_\_Thruster\_keyword\_group.html)
- [Thruster!On/Off](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/VehicleModels/html/group__Thruster_On-Off_keyword_group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/VehicleModels/html/group\_\_Thruster\_On-Off\_keyword\_group.html)
- [Thruster!Throttled](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/VehicleModels/html/group__Thruster_Throttled_keyword_group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/VehicleModels/html/group\_\_Thruster\_Throttled\_keyword\_group.html)
- [consumption](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/VehicleModels/html/group__consumption_keyword_group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/VehicleModels/html/group\_\_consumption\_keyword\_group.html)
- [model](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/VehicleModels/html/group__model_keyword_group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/VehicleModels/html/group\_\_model\_keyword\_group.html)
- [model!Fuel](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/VehicleModels/html/group__model_Fuel_keyword_group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/VehicleModels/html/group\_\_model\_Fuel\_keyword\_group.html)

#### Description

Extended version of `NewThrottledThruster`.

Extended version of `NewThrottledThruster` with additional parameters for delay, cutoff\_criteria and coefficient table. The 'coeff' is now a parameter instead of a signal. If the coeffFile parameter is specified, the thruster coefficient is interpolated from the file and the coeff parameter is not used. If the thrustFile parameter is specified: the thrust magnitude is taken from the thrustFile and coeffFile and coeff are not used. If the flowrateFile parameter is specified: the fuel flowrate is taken from the flowrateFile. The coeffFile should have this format: 1. Text file with 2 columns. Each column is a floating point number. Columns are separated by 1 or more spaces. Lines beginning with a non-numeric character are treated as comments. 2. First column contains 'burn time' values in seconds (the independent value). 3. Second column contains the thruster level coefficient (0.0=0%, 1.0=100%) The thrustFile should have this format: 1. Text file with 2 columns. Each column is a floating point number. Columns are separated by 1 or more spaces. Lines beginning with a non-numeric character are treated as comments. 2. First column contains 'burn time' values in seconds (the independent value). 3. Second column contains the thrust magnitude. The flowrateFile should have this format: 1. Text file with 2 columns. Each column is a floating point number. Columns are separated by 1 or more spaces. Lines beginning with a non-

numeric character are treated as comments. 2. First column contains 'burn time' values in seconds (the independent value). 3. Second column contains the fuel flowrate The dryMassFlowrateFile should have this format: 1. Text file with 2 columns. Each column is a floating point number. Columns are separated by 1 or more spaces. Lines beginning with a non-numeric character are treated as comments. 2. First column contains 'burn time' values in seconds (the independent value). 3. Second column contains the dry mass flowrate (kg/s) Interpolation is done through the mathc90 dilup() function with ndeg=3 and lup=1 (see mathc90/doc/ch12-01.pdf for dilup documentation). There is no extrapolation; end points are used if dilup reports extrapolation is necessary.

from the VehicleModels models library for the xref:Dshellpp\_module simulation framework.

### Model ThrottledProfileThruster class details

For more information on the members and functions of this model class, please see [VehicleModels::ThrottledProfileThruster model class doxygen documentation](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/VehicleModels/html/group__ThrottledProfileThruster_group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/VehicleModels/html/group\_\_ThrottledProfileThruster\_group.html)

### Enums

#### CutOffCriteria Enum

In the ThrottledProfileThruster model definition, the original enum CutOffCriteria is defined as:

```
enum CutOffCriteria
{
    FUEL_MASS = 0,
    BURN_TIME = 1
};
```

In the C model code, these enum values can be accessed by using the follow C enum definitions:

```
/// ThrottledProfileThrusterCutOffCriteria enum
enum ThrottledProfileThrusterCutOffCriteria
{
    THROTTLED_PROFILE_THRUSTER_CUT_OFF_CRITERIA_FUEL_MASS = 0,
    THROTTLED_PROFILE_THRUSTER_CUT_OFF_CRITERIA_BURN_TIME = 1
};
```

### Parameters

Name	Type	Size	Quantity	Units	Description
coeff	double		Dimensionless		percentage of thrust (from 0 (minimum) to 1 (maximum)); used only if coeffFile is not specified
coeffFile	string				file containing burn time vs. thruster level coefficient; use to compute thrust magnitude. Not used if thrustFile is specified.
cutoff_criteria	CutOffCriteria				specifies criteria to cut off engine, either 'FUEL_MASS' or 'BURN_TIME'
cutoff_value	double		Unspecified		numeric value with units of kg or sec depending on the cutoff_criteria
delay	double		Time		delay time before engine starts thrusting
dryMassFlowrateFile	string				file containing burn time vs. dry mass flowrate; if not specified, dry mass does not reduce
flowrateFile	string				file containing burn time vs. fuel flowrate; if not specified, the flowrate will be estimated from the ISP
isp_max	double		SpecificImpulse		maximum specific impulse
isp_min	double		SpecificImpulse		minimum specific impulse
max_level	double		Force		maximum force generated by this thruster when on

Name	Type	Size	Quantity	Units	Description
min_level	double		Force		minimum force generated by this thruster when on
thrustFile	string				file containing burn time vs. thrust magnitude. If not specified, thrust magnitude is computed from coeff parameter:

#### Discrete States

Name	Type	Size	Quantity	Units	Description
cumulative_impulse	double		Impulse		cumulative impulse i.e. summation of $F*dt$ where $dt$ is incremental burn time; $F = \text{average}(F[i], F[i-1])$
cur_coeff	double		Dimensionless		current percentage of thrust (from 0 to 1)
cur_dryMassFlowrate	double		MassChangeRate		current dry mass flowrate
cur_flowrate	double		MassChangeRate		current fuel flowrate
cur_isp	double		SpecificImpulse		current specific impulse (only used if flowrateFile is not given)
cur_thrust	double		Force		current thrust magnitude
dryMassFlowrateFileBurnTimeOffset	double		Time		This value will be subtracted from burn_time for the dry mass flowrate table lookup
flowrateFileBurnTimeOffset	double		Time		This value will be subtracted from burn_time for the fuel flowrate table lookup
on_off_time	double		Time		time when thruster was last turned on/off (seconds)
thrustFileBurnTimeOffset	double		Time		This value will be subtracted from burn_time for the table lookup

#### Continuous States

Name	Type	Size	Quantity	Units	Description
dry_mass_consumed	double		Mass		State to track dry mass consumption of thruster

#### Flow Ins

Name	Type	Size	Quantity	Units	Description
on_off	double		Dimensionless		command to turn thruster on (!=0) or off (0)

#### Flow Outs

Name	Type	Size	Quantity	Units	Description
------	------	------	----------	-------	-------------



Name	Type	Size	Quantity	Units	Description
dry_mass_consumed	double		Mass		total dry mass used by thruster

### 33.10. VehicleModels::ThrottledProfileThrusterWithBackPressure Actuator Model



This model documentation is auto-generated. Documentation updates should be made to the model's `mdl` file to avoid the changes being lost.

[Doxygen link to all VehicleModels models of Actuator class](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/VehicleModels/html/group__Actuator_group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/VehicleModels/html/group\_\_Actuator\_group.html)

#### Keywords Doxygen groups

- [Back](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/VehicleModels/html/group__Back_keyword_group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/VehicleModels/html/group\_\_Back\_keyword\_group.html)
- [Fuel](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/VehicleModels/html/group__Fuel_keyword_group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/VehicleModels/html/group\_\_Fuel\_keyword\_group.html)
- [Nozzle](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/VehicleModels/html/group__Nozzle_keyword_group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/VehicleModels/html/group\_\_Nozzle\_keyword\_group.html)
- [Resource](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/VehicleModels/html/group__Resource_keyword_group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/VehicleModels/html/group\_\_Resource\_keyword\_group.html)
- [Thruster](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/VehicleModels/html/group__Thruster_keyword_group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/VehicleModels/html/group\_\_Thruster\_keyword\_group.html)
- [Thruster!On/Off](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/VehicleModels/html/group__Thruster_On-Off_keyword_group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/VehicleModels/html/group\_\_Thruster\_On-Off\_keyword\_group.html)
- [Thruster!Throttled](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/VehicleModels/html/group__Thruster_Throttled_keyword_group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/VehicleModels/html/group\_\_Thruster\_Throttled\_keyword\_group.html)
- [area](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/VehicleModels/html/group__area_keyword_group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/VehicleModels/html/group\_\_area\_keyword\_group.html)
- [consumption](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/VehicleModels/html/group__consumption_keyword_group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/VehicleModels/html/group\_\_consumption\_keyword\_group.html)
- [model](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/VehicleModels/html/group__model_keyword_group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/VehicleModels/html/group\_\_model\_keyword\_group.html)
- [model!Fuel](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/VehicleModels/html/group__model_Fuel_keyword_group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/VehicleModels/html/group\_\_model\_Fuel\_keyword\_group.html)
- [pressure](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/VehicleModels/html/group__pressure_keyword_group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/VehicleModels/html/group\_\_pressure\_keyword\_group.html)

#### Description

This model takes into account reduced thrust due to back pressure and nozzle area.

Subclassed from `ThrottledProfileThruster` with thrust reduced by the back pressure multiplied by nozzle area.

from the `VehicleModels` models library for the `xref:Dshellpp_module` simulation framework.

#### Model `ThrottledProfileThrusterWithBackPressure` class details

For more information on the members and functions of this model class, please see [VehicleModels::ThrottledProfileThrusterWithBackPressure model class doxygen documentation](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/VehicleModels/html/group__ThrottledProfileThrusterWithBackPressure_group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/VehicleModels/html/group\_\_ThrottledProfileThrusterWithBackPressure\_group.html)

#### Enums

#### Parameters

Name	Type	Size	Quantity	Units	Description
nozzle_area	double		Area		nozzle area (m**2)

#### Discrete States

Name	Type	Size	Quantity	Units	Description
back_pressure_force	double		Force		total back pressure force (newtons)

#### Flow Ins

Name	Type	Size	Quantity	Units	Description
back_pressure	double		Pressure		net back pressure (newtons/m**2)

### 33.11. VehicleModels::ThrottledThruster Actuator Model



This model documentation is auto-generated. Documentation updates should be made to the model's `mdl` file to avoid the changes being lost.

[Doxygen link to all VehicleModels models of Actuator class](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/VehicleModels/html/group__Actuator_group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/VehicleModels/html/group\_\_Actuator\_group.html)

#### Keywords Doxygen groups

- [Fuel](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/VehicleModels/html/group__Fuel_keyword_group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/VehicleModels/html/group\_\_Fuel\_keyword\_group.html)
- [Resource](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/VehicleModels/html/group__Resource_keyword_group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/VehicleModels/html/group\_\_Resource\_keyword\_group.html)
- [Thruster](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/VehicleModels/html/group__Thruster_keyword_group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/VehicleModels/html/group\_\_Thruster\_keyword\_group.html)
- [Thruster!On/Off](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/VehicleModels/html/group__Thruster_On-Off_keyword_group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/VehicleModels/html/group\_\_Thruster\_On-Off\_keyword\_group.html)
- [Thruster!Throttled](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/VehicleModels/html/group__Thruster_Throttled_keyword_group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/VehicleModels/html/group\_\_Thruster\_Throttled\_keyword\_group.html)
- [consumption](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/VehicleModels/html/group__consumption_keyword_group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/VehicleModels/html/group\_\_consumption\_keyword\_group.html)
- [model](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/VehicleModels/html/group__model_keyword_group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/VehicleModels/html/group\_\_model\_keyword\_group.html)
- [model!Fuel](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/VehicleModels/html/group__model_Fuel_keyword_group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/VehicleModels/html/group\_\_model\_Fuel\_keyword\_group.html)

#### Description

Models a throttled thruster with ON/OFF command, percentage of thrust and no delay. This thruster tracks fuel consumption and outputs total fuel burned.

The ThrottledThruster model accepts an on(1) or off(0) flowIn plus a percentage of thrust (minimum thrust 0, maximum is 1). It has parameters for direction (orientation) of thrusters and both minimum and maximum thruster force level. The thrust commands are executed without delay, and there is no ramp up or ramp down time on the thrust output. In addition, the model tracks fuel consumption with a continuous state and outputs the total fuel consumed by the thruster. This can be tied to a FuelTank model in order to change the mass and inertial properties of the s/c to model fuel depletion. The fuel\_remaining flowOut of the FuelTank model should be tied to the fuel\_remaining flowIn of this model. This allows the thruster to know when all available fuel has been used.

from the VehicleModels models library for the xref:Dshellpp\_module simulation framework.

#### Model ThrottledThruster class details

For more information on the members and functions of this model class, please see [VehicleModels::ThrottledThruster model class doxygen documentation](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/VehicleModels/html/group__ThrottledThruster_group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/VehicleModels/html/group\_\_ThrottledThruster\_group.html)

#### Parameters

Name	Type	Size	Quantity	Units	Description
isp_gfactor	double		Acceleration		acceleration of gravity used for computing fuel consumption
isp_max	double		SpecificImpulse		maximum specific impulse
isp_min	double		SpecificImpulse		minimum specific impulse
max_level	double		Force		maximum force generated by this thruster when on
min_level	double		Force		minimum force generated by this thruster when on

#### Discrete States

Name	Type	Size	Quantity	Units	Description
cur_coeff	double		Dimensionless		current percentage of thrust (from 0 to 1)
isp	double		SpecificImpulse		specific impulse
level	double		Force		current force level

#### Flow Ins

Name	Type	Size	Quantity	Units	Description
coeff	double		Dimensionless		percentage of thrust (from 0 (minimum) to 1 (maximum))
on_off	double		Dimensionless		command to turn thruster on (!=0) or off (0)

### 33.12. VehicleModels::ThrottledThrusterMinimal Actuator Model



This model documentation is auto-generated. Documentation updates should be made to the model's `mdl` file to avoid the changes being lost.

[Doxygen link to all VehicleModels models of Actuator class](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/VehicleModels/html/group__Actuator_group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/VehicleModels/html/group\_\_Actuator\_group.html)

#### Keywords Doxygen groups

- [Fuel](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/VehicleModels/html/group__Fuel_keyword_group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/VehicleModels/html/group\_\_Fuel\_keyword\_group.html)
- [Resource](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/VehicleModels/html/group__Resource_keyword_group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/VehicleModels/html/group\_\_Resource\_keyword\_group.html)
- [Thruster](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/VehicleModels/html/group__Thruster_keyword_group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/VehicleModels/html/group\_\_Thruster\_keyword\_group.html)
- [Thruster!On/Off](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/VehicleModels/html/group__Thruster_On-Off_keyword_group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/VehicleModels/html/group\_\_Thruster\_On-Off\_keyword\_group.html)
- [Thruster!Throttled](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/VehicleModels/html/group__Thruster_Throttled_keyword_group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/VehicleModels/html/group\_\_Thruster\_Throttled\_keyword\_group.html)
- [consumption](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/VehicleModels/html/group__consumption_keyword_group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/VehicleModels/html/group\_\_consumption\_keyword\_group.html)
- [model](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/VehicleModels/html/group__model_keyword_group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/VehicleModels/html/group\_\_model\_keyword\_group.html)
- [model!Fuel](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/VehicleModels/html/group__model_Fuel_keyword_group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/VehicleModels/html/group\_\_model\_Fuel\_keyword\_group.html)

#### Description

Models a thruster with its exit velocity and thrust force. This thruster tracks fuel consumption and outputs total fuel burned.

The ThrottledThrusterMinimal model accepts the thrust force and exit velocity. This is a very basic set of inputs, and more sophisticated models can be built on top of this one. It has parameters for direction (orientation) of thrusters and both minimum and maximum thruster force level. In addition, the model tracks fuel consumption with a continuous state and outputs the total fuel consumed by the thruster. This can be tied to a FuelTank model in order to change the mass and inertial properties of the s/c to model fuel depletion. The fuel\_remaining flowOut of the FuelTank model should be tied to the fuel\_remaining flowIn of this model. This allows the thruster to know when all available fuel has been used.

from the VehicleModels models library for the xref:Dshellpp\_module simulation framework.

#### Model ThrottledThrusterMinimal class details

For more information on the members and functions of this model class, please see [VehicleModels::ThrottledThrusterMinimal model class doxygen documentation](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/VehicleModels/html/group__ThrottledThrusterMinimal_group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/VehicleModels/html/group\_\_ThrottledThrusterMinimal\_group.html)

#### Parameters

Name	Type	Size	Quantity	Units	Description
max_level	double		Force		maximum force generated by this thruster when on
min_level	double		Force		minimum force generated by this thruster when on

#### Discrete States

Name	Type	Size	Quantity	Units	Description
thrust	double		Force		current force level generated by this thruster

#### Flow Ins

Name	Type	Size	Quantity	Units	Description
current_thrust	double		Force		Force currently generated by the thruster
exit_velocity	double		Velocity		Thruster exit velocity, in m/s

### 33.13. VehicleModels::ThrottledThrusterWithBackPressure Actuator Model



This model documentation is auto-generated. Documentation updates should be made to the model's `mdl` file to avoid the changes being lost.

[Doxygen link to all VehicleModels models of Actuator class](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/VehicleModels/html/group__Actuator_group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/VehicleModels/html/group\_\_Actuator\_group.html)

#### Keywords Doxygen groups

- [Back](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/VehicleModels/html/group__Back_keyword_group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/VehicleModels/html/group\_\_Back\_keyword\_group.html)
- [Fuel](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/VehicleModels/html/group__Fuel_keyword_group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/VehicleModels/html/group\_\_Fuel\_keyword\_group.html)
- [Nozzle](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/VehicleModels/html/group__Nozzle_keyword_group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/VehicleModels/html/group\_\_Nozzle\_keyword\_group.html)
- [Resource](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/VehicleModels/html/group__Resource_keyword_group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/VehicleModels/html/group\_\_Resource\_keyword\_group.html)
- [Thruster](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/VehicleModels/html/group__Thruster_keyword_group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/VehicleModels/html/group\_\_Thruster\_keyword\_group.html)
- [Thruster!On/Off](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/VehicleModels/html/group__Thruster_On-Off_keyword_group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/VehicleModels/html/group\_\_Thruster\_On-Off\_keyword\_group.html)
- [Thruster!Throttled](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/VehicleModels/html/group__Thruster_Throttled_keyword_group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/VehicleModels/html/group\_\_Thruster\_Throttled\_keyword\_group.html)
- [area](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/VehicleModels/html/group__area_keyword_group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/VehicleModels/html/group\_\_area\_keyword\_group.html)
- [consumption](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/VehicleModels/html/group__consumption_keyword_group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/VehicleModels/html/group\_\_consumption\_keyword\_group.html)
- [model](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/VehicleModels/html/group__model_keyword_group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/VehicleModels/html/group\_\_model\_keyword\_group.html)
- [model!Fuel](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/VehicleModels/html/group__model_Fuel_keyword_group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/VehicleModels/html/group\_\_model\_Fuel\_keyword\_group.html)
- [pressure](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/VehicleModels/html/group__pressure_keyword_group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/VehicleModels/html/group\_\_pressure\_keyword\_group.html)

#### Description

Models a throttled thruster with ON/OFF command, percentage of thrust and no delay. Takes into account reduced thrust due to back pressure and nozzle area. This thruster tracks fuel consumption and outputs total fuel burned.

The ThrottledThruster model accepts an on(1) or off(0) flowIn plus a percentage of thrust (minimum thrust 0, maximum is 1). Thrust is reduced by the back pressure multiplied by nozzle area. It has parameters for direction (orientation) of thrusters and both minimum and maximum thruster force level. The thrust commands are executed without delay, and there is no ramp up or ramp down time on the thrust output. In addition, the model tracks fuel consumption with a continuous state and outputs the total fuel consumed by the thruster. This can be tied to a FuelTank model in order to change the mass and inertial properties of the s/c to model fuel depletion. The fuel\_remaining flowOut of the FuelTank model should be tied to the fuel\_remaining flowIn of this model. This allows the thruster to know when all available fuel has been used.

from the VehicleModels models library for the xref:Dshellpp\_module simulation framework.

#### Model ThrottledThrusterWithBackPressure class details

For more information on the members and functions of this model class, please see [VehicleModels::ThrottledThrusterWithBackPressure model class doxygen documentation](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/VehicleModels/html/group__ThrottledThrusterWithBackPressure_group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/VehicleModels/html/group\_\_ThrottledThrusterWithBackPressure\_group.html)

#### Parameters

Name	Type	Size	Quantity	Units	Description
nozzle_area	double		Area		nozzle area (m**2)

#### Discrete States

Name	Type	Size	Quantity	Units	Description
back_pressure_force	double		Force		total back pressure force (newtons)

#### Flow Ins

Name	Type	Size	Quantity	Units	Description
back_pressure	double		Pressure		net back pressure (newtons/m**2)

### 33.14. VehicleModels::ThrusterBase Actuator Model



This model documentation is auto-generated. Documentation updates should be made to the model's `mdl` file to avoid the changes being lost.

[Doxygen link to all VehicleModels models of Actuator class](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/VehicleModels/html/group__Actuator_group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/VehicleModels/html/group\_\_Actuator\_group.html)

#### Keywords Doxygen groups

- [Fuel](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/VehicleModels/html/group__Fuel_keyword_group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/VehicleModels/html/group\_\_Fuel\_keyword\_group.html)
- [Resource](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/VehicleModels/html/group__Resource_keyword_group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/VehicleModels/html/group\_\_Resource\_keyword\_group.html)

- [Thruster](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/VehicleModels/html/group_Thruster_keyword_group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/VehicleModels/html/group\_Thruster\_keyword\_group.html)
- [Thruster!Base](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/VehicleModels/html/group_Thruster_Base_keyword_group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/VehicleModels/html/group\_Thruster\_Base\_keyword\_group.html)
- [consumption](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/VehicleModels/html/group_consumption_keyword_group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/VehicleModels/html/group\_consumption\_keyword\_group.html)
- [model](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/VehicleModels/html/group_model_keyword_group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/VehicleModels/html/group\_model\_keyword\_group.html)
- [model!Fuel](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/VehicleModels/html/group_model_Fuel_keyword_group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/VehicleModels/html/group\_model\_Fuel\_keyword\_group.html)

### Description

Models the most basic properties of a thruster. Not for direct use.

This models the basic properties common to all thrusters. Simulations generally want more than just what this class provides, so you probably want to use one of the models derived from this one to provide the specific functionality you need.

from the VehicleModels models library for the xref:Dshellpp\_module simulation framework.

### Model ThrusterBase class details

For more information on the members and functions of this model class, please see [VehicleModels::ThrusterBase model class doxygen documentation](https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/VehicleModels/html/group_ThrusterBase_group.html) (https://dartslab.jpl.nasa.gov/internal/www/DLabDocs/modules/VehicleModels/html/group\_ThrusterBase\_group.html)

### Parameters

Name	Type	Size	Quantity	Units	Description
BASEBODY_UUID	int				<b>NOT FOR USER INPUT</b> - UUID for the base body of the spacecraft.
burns_fuel	int				flag to turn on (1) or off (0) mass/fuel consumption
orientation	double	3	Dimensionless		direction thruster is pointing

### Scratch

Name	Type	Size	Quantity	Units	Description
burn_time	double		Time		Engine on-time FOR THIS CYCLE
burn_time_total	double		Time		Combined engine on-time FOR ALL CYCLES
cg2node_cbFrame	double	3	Length		Vector from total system cg to force node in CapsuleBase frame. Requires BASEBODY_UUID param
moment_cbFrame	double	3	Torque		Moment = vector from the total system cg to the force node [cross] force vector (which includes all atmospheric effects) in CapsuleBase frame. Requires BASEBODY_UUID param
nodeloc_cbFrame	double	3	Length		Force node location in CapsuleBase frame. Requires BASEBODY_UUID param

### Discrete States

Name	Type	Size	Quantity	Units	Description
cur_posn	int				whether the thruster is currently on or off
step_posn	int				whether the thruster was ever on this step

Name	Type	Size	Quantity	Units	Description
warning	int				out of fuel warning flag

#### Continuous States

Name	Type	Size	Quantity	Units	Description
fuel_used	double		Mass		State to track fuel consumption of thruster

#### Flow Ins

Name	Type	Size	Quantity	Units	Description
fuel_remaining	double		Mass		Mass of fuel remaining in tank.

#### Flow Outs

Name	Type	Size	Quantity	Units	Description
fVec	double	3	Force		force direction and magnitude
fuel_used	double		Mass		total fuel used by thruster

## Index

---

1. Note that some of the code snippets used for illustrative purposes here are from the regression test: DshellCommon/test/test\_Ndarts/test\_VehicleAssembly\_spice.
2. Note that some of the code snippets used for illustrative purposes here are from the regression test: DshellCommon/test/test\_Ndarts/test\_VehicleAssembly\_spice.

Last updated 2023-10-06 06:11:52 -0700