User Manual 4.14 Assemblies in PATRIUS: Building and using an assembly

De Wiki
Aller à : navigation, rechercher
User Manual 4.14 Assemblies in PATRIUS: Building and using an assembly

Sommaire

Introduction

Scope

This section covers the process of building assemblies. Assemblies can be build using:

- The AssemblyBuilder class for full, complex Assemblies including mobile parts.
- The Vehicle class for usual satellites.

Javadoc

The main assembly objects are available in the package `fr.cnes.sirius.patrius.assembly`

Library	Javadoc
Patrius	Package fr.cnes.sirius.patrius.assembly

Links

None as of now.

Useful Documents

None as of now.

Package Overview

Please refer to the [SPC_VEH_Home#HOverview Assemblies presentation chapter].

Features Description

The AssemblyBuilder

This construction is to be made using the builder (AssemblyBuilder class). The first step is to create an instance of this builder.

It provides then all the methods needed to build an assembly :

To define the main part (method “addMainPart”)
: This method creates the main part by giving it a name (String type). This part can remain a virtual one (only a name and a frame) or describe the main body of the assembly. Its frame is the main frame that will be used to link the assembly to the rest of the main tree of frames by an attitude law. From this frame are defined the positions of all other pieces in the local tree of frames.

To add a part (method addPart)
: To add parts other than the main one is facultative : an assembly can be defined only by one part. Adding parts can only be done if at least the main part has been created.

To add a part to an assembly, two information are necessary : to give it a name, and to define its associated frame from the one of an existing part. To define this frames, two ways are proposed :

- Using fixed transformations: by a transformation (Transform class : a translation and rotation) or by a translation (Vector3D : center of the new frame in the parent frame) and then a rotation (Rotation from the parent frame).
- Using n implementation of TransformStateProvider: the part will hence be a mobile part.

To add a property to a part (method addProperty)
: For each property, there is a specific class. The entries are the ones needed to describe this particular property. When the property is instanciated, the user can assign it to a part. One part can only have one property of each type (see the "presentation" chapter).

To get the built assembly (method returnAssembly)

The output of this method is the assembly (Assembly class) as built until this operation. It contains the assembly with all the parts created by the user (including their properties and frames).

To link it to the tree of frames

The assembly can be linked to the main tree of frames by defining its main part's frame in it. The user can simply define this frame or use a SpacecraftState object to do so.

The Vehicle

An assembly can also be build using the `Vehicle` class. The Vehicle class is meant to be used for quickly building usual satellite containing:

- A main body (sphere, cylinder or parrallelepiped)
- *n* orientable solar panels
- *n* engines
- *n* tanks
- Aerodynamic properties (apply to main body and solar panels)
- Radiative properties (apply to main body and solar panels)

The vehicle class possesses methods to return an Assembly built with user-provided features.

Getting Started

Building an assembly from the assembly builder

The AssemblyBuilder class is a utility class that allows the user to create an assembly, like a satellite or a ground station. When creating an instance of this class, a new assembly is instantiated internally and the user can thus use all the methods of the AssemblyBuilder class described hereunder.

Instantiating an assembly builder

The user can instantiate an assembly builder by calling the constructor of the AssemblyBuilder class:

```
// An AssemblyBuilder instance
final AssemblyBuilder builder = new AssemblyBuilder();
```

See section [SPC_VBU_Home#HRecovertheassembly Recover the assembly] to build an instance of AssemblyBuilder with an already existing assembly.

When this is done, a new assembly is created in the builder instance.

Adding parts to the assembly

The user must first create the main part of the assembly. After having done so, other parts can be added to the assembly using the methods provided in AssemblyBuilder.

Adding the main part

The method `addMainPart` is the only method that adds a MainPart to the assembly. This method creates the main part by assigning a name (a String) to it. This part can be a virtual one (only a name and a frame) or can be used to describe the main body of satellite (like the platform). Its frame is the assemblies main frame : it will be used to link the assembly to the OREKIT frames tree by an attitude law, allowing the user to position the satellite on an orbit and perform otherwise complex computations (eg. orientation of TC/TM antennas in the groundstation frame).

The user can add the main part like so :

```
// Adding main part
String mainBody = "mainBody";
builder.addMainPart(mainBody);
```

Adding a part

The method `addPart` allows the user to add a part to the assembly. Adding parts other than the main one is not necessary : an assembly can be defined by its MainPart only.
To add a part to the assembly, three things are necessary :

- a name (String type- adding parts with the same name is forbidden),
- the name of the parent part (either the mainPart or another part),
- a frame that represents the orientation of the part with respect to its parent.

Creating this frame can be done by defining the static transformation (Transform class) from the parent part's frame, or more directly by giving the translation vector (coordinates of the zero point of the new part's frame expressed in the parent part's frame) and the rotation then applied to get the new part's frame (rotation from the parent part's frame).

The following code will add a part to the assembly defined by a frame transformation relatively to the mainPart :

```
// Adding a part
//  Create transformation from mainPart
final Vector3D translation = new Vector3D(1.0, 0.0, 0.0);
final Transform transformation = new Transform(date, translation);
//  Name of the new part
final String newPart = "newPart";
//  Add it to the assembly
builder.addPart(newPart, mainBody, transformation);
```

The translation here defines the position of the new part's frame "zero" point in the frame of the parent part.

The following code will add a part to the assembly defined by a translation (Vector3D) and then a rotation (Rotation) relatively to the newly created translatedPart :

```
// Adding a part
//  Create the translation
final Vector3D translation = new Vector3D(3.0,-2.0, 1.0);
//  Create the rotation
```

```
final Vector3D rotationAxis = Vector3D.PLUS_K;
final double angle = FastMath.PI / 4.0;
final Rotation rotation = new Rotation(rotationAxis , angle);
//  Name of the new part
final String anotherPart = "anotherPart";
//  Add it to the assembly
builder.addPart(anotherPart, mainBody, translation, rotation);
```

The following code add a mobile part with one degree of liberty to the assembly:

```
// One DOL Transform provider (uniform rotation around +K axis)
final OrientationAngleProvider oap = new OrientationAngleProvider() {
    @Override
    public double getOrientationAngle(final PVCoordinatesProvider pvProv,
final AbsoluteDate date) throws PatriusException {
        return date.durationFrom(initialDate) * FastMath.PI / 4.;
    }
};
final TransformStateProvider t = new
OrientationAngleTransform(Transform.IDENTITY, Vector3D.PLUS_K, oap);

// Add part to assembly
builder.addPart("Solar panel", "Main", t);
```

Note : Adding parts can be done only if the main part has been created.

Adding properties to a part

Creating a property

The user can create a property using the specific class designed to model said property (such classes implement the [SPC_VBU_Interfaces IPartProperty interface]) like so :

```
// Creating a mass property
final IPartProperty propertyA = new PropertyA(PropertyAInputParams);
```

Adding a property to a part

The method `addProperty` allows the user to add the selected property to a part of the assembly like so :

```
// Creating a mass property
builder.addProperty(propertyA, mainBody);
```

Available properties

Each property is of a particular type amongst a list (see [SPC_VEH_Feature2 Part Properties]) and one part can only have only one property of each type (so as to avoid having different masses for the same part, for example).

Recover the assembly

Using the method `returnAssembly`, the user can get the assembly as built. It contains the created parts with all their properties and frames. This method can be used like so :

```
// Get the created assembly
Assembly myAssembly = builder.returnAssembly();
```

Note : Calling this method will clear the assembly field in the AssemblyBuilder instance. Thus, if this method is called again, a `new Assembly()` is returned. If the user wants to modify the assembly after having called the `returnAssembly()` method, then a new AssemblyBuilder must be instanciated like so :

```
AssemblyBuilder newBuilder = new AssemblyBuilder(myAssembly);
```

Positioning the assembly

When the AssemblyBuilder instance is made, the assemblies main part is associated to a default frame that is **not** linked to the Orekit frames tree. The mainPart frame is an orphan leaf that is also the root of the assemblies frames tree and that is not defined by a transformation from a frame on the Orekit tree. Thus, transformations between this frame (and the frames linked to the mainPart frame) and the Orekit frames tree are not possible.

In order to allow such transformations, the user must define the position and orientation of the assembly relatively to an existing Orekit frame. This is done by overwriting the mainPart frame and by setting it to a frame that is linked to the Orekit frames tree. Specifically, the parts frames, including the main part, are instances of UpdatableFrame. This allows updating the transform between the reference frame and the main part frame (for example during propagation) whitout having to create frames all over again.

Assuming `gcrf` is the reference frame for the propagation, the user can create an initial UpdatableFrame with an identity transformation and set it as the main parts frame. It is important to note that the reference frame (parent) of the main part is coherent with the inertial frame for the propagation.

```
final UpdatableFrame mainFrame = new UpdatableFrame(gcrf, Transform.IDENTITY,
"mainPartFrame");
builder.initMainPartFrame(mainFrame);
```

Additionally, the user can pass the initial orbit bulletin (SpacecraftState) to a method bearing the same name that creates an initial frame with the correct position and orientation :

```
builder.initMainPartFrame(initialSpacecraftState);
```

Building an assembly from the Vehicle

The Assembly can also be built from a `Vehicle` object, which represents a usual satellite :
- The main body of the satellite is a `CrossSectionProvider` with available sphere, cylinder or

parallelepiped shape.
- The vehicle can be composed of several solar panels, tanks and engines (materialised by `TankProperty` and `PropulsiveProperty`). In order to build the Assembly, the user should start calling: `final Vehicle vehicle = new Vehicle()(CrossSectionProvider))`

Then add parts and properties to the vehicle using the following methods :

- `setMainShape(CrossSectionProvider))`
- `setDryMass(double mass)`
- `setAerodynamicsProperties(IParamDiffFunction cx, IParamDiffFunction cz)`
- `setRadiativeProperties(double kabs, double kSpec, double kDiff, double kabsIR, double kSpecIR, double kDiffIR)`
- `addEngine(PropulsiveProperty)`
- `addTank(TankProperty)`
- `addSolarPanel(Vector3D normal, double area)`

The only compulsory method to call is `setMainShape(CrossSectionProvider))` (exception thrown otherwise).
Aerodynamic properties and radiative properties application is the following:
- AERO_FACET properties are added to solar panels (for use with AeroModel)
- AERO_CROSS_SECTION is added to main body (for use with AeroModel)
- AERO_GLOBAL is added to main body (for use with DragLiftModel)
- RADIATIVE_FACET and RADIATIVE/RADIATIVEIR properties are added to solar panels (for use with DirectRadiativeModel and RediffusedRadiativeModel)
- RADIATIVE_CROSS_SECTION and RADIATIVE/RADIATIVEIR are added to main body (for use with DirectRadiativeModel and RediffusedRadiativeModel)

Finally the Assembly is built using the methods `createAssembly(Frame)`.
Note that another method `createAssembly(Frame, cMass, cDrag, cSRP)` is available: it allows to build an Assembly whose mass, drag and SRP properties have a multiplicative coefficient (useful for Monte-Carlo simulations for instance).

As the Vehicle offers a `CrossSectionProvider` as main shape, aero and radiative models of PATRIUS take into account cross section and attitude in order to compute accurate cross section used in drag and SRP computation.

Link to the tree of frames

Once the vehicle has been built with the builder, it can be linked to the frames tree. This operation is to be done using one of the following methods of the Assembly class (see javadoc for more info) :

- **initMainPartFrame(SpacecraftState)**: initialize the main part's frame using a SpacecraftState; the frame is computed from the spacecraft PV coordinates and attitude, and the parent frame is the spacecraft definition frame;
- **initMainPartFrame(UpdatableFrame)**: initialize the main part's frame using an UpdatableFrame.

The frames of all the other parts are automatically updated during this initialization process.

Once the vehicle has been built and its parent frame initialized, the user may need to update the main part frame with respect to its parent frame. Two methods exist in the Assembly class to this purpose:

- **updateMainPartFrame(SpacecraftState)**: update the main part frame using the PV coordinates and attitude of the SpacecraftState;
- **updateMainPartFrame(Transform)**: update the main part frame using a Transform instance.

Note : It is assumed the user has already initialized the parent frame (using one of the two methods above). If that is not the case, an IllegalArgumentException is thrown.

A code sample of the previous functions is hereafter.

```
SpacecraftState state = new SpacecraftState(orbit);
final UpdatableFrame frame = new UpdatableFrame(gcrf, transform, "main");

// to initialize the main part frame using a SpacecraftState:
assembly.initMainPartFrame(state);
// or using an UpdatableFrame: assembly.initMainPartFrame(frame);

// to update the main part frame using a SpacecraftState (only if the main
part has already a parent frame):
assembly.updateMainPartFrame(state);
// or using a Tranform: assembly.updateMainPartFrame(transform);
```

Access to the information

Once the vehicle built, the user can access the parts’ properties :

- Access to the concerned part : call to the vehicle’s method “getPart(String)” with the name of the part.
- Access to the property : call to the part’s method “getProperty(PropertyType)” with the type of property wanted.

The user can first check if this part contains a property of this type by the part’s “hasProperty(PropertyType)” that returns a Boolean.

Only the information about each part can be get this way : to have access to global information computed from several parts, a "model" shall be used.

Models

Once built, the assembly shall be used through models, that will use data of the properties in their computations.
Models are independent of the assembly description : a model can use it or not.
A model can use another model.
For details about models, please refer each themes associated chapters. ("Properties and models")

Contents

Interfaces

Interface	Summary	Javadoc
IPart	Interface for parts composing an Assembly object.	IPart

IPartProperty	Interface for the properties of the Assembly parts.	IPartProperty
TransformStateProvider	Interface for generic transform providers based on spacecraft state.	TransformStateProvider

Classes

Class	Summary	Javadoc
Assembly	This class implements an assembly object.	Assembly
AssemblyBuilder	This class is a builder to create instances of the Assembly object.	AssemblyBuilder
MainPart	This class represents the Assembly main part.	MainPart
Part	This class represents a generic (fixed) part of the Assembly.	Part
MobilePart	This class represents a mobile part of the Assembly.	MobilePart
PropertyType	This enumeration lists all the possible property types that can be associated to an assembly part.	PropertyType
Vehicle	This class implements an vehicle object : a satellite with solar panels, engines and tanks enabling an easy Assembly construction.	Vehicle
OrientationAngleTransform	This class represents a one-degree of liberty transformation for mobile parts (for example a rotating solar panel).	OrientationAngleTransform

Variantes

Affichages

- Lire
- Voir le texte source
- Historique
- Exporter en PDF

Plus

Rechercher

Rechercher Lire

PATRIUS

- Welcome

Evolutions

- Main differences between V4.14 and V4.13
- Main differences between V4.13 and V4.12
- Main differences between V4.12 and V4.11
- Main differences between V4.11 and V4.10
- Main differences between V4.10 and V4.9
- Main differences between V4.9 and V4.8
- Main differences between V4.8 and V4.7
- Main differences between V4.7 and V4.6.1
- Main differences between V4.6.1 and V4.5.1
- Main differences between V4.5.1 and V4.4
- Main differences between V4.4 and V4.3
- Main differences between V4.3 and V4.2
- Main differences between V4.2 and V4.1.1
- Main differences between V4.1.1 and V4.1
- Main differences between V4.1 and V4.0
- Main differences between V4.0 and V3.4.1

User Manual

- User Manual 4.14
- User Manual 4.13

- User Manual 4.12
- User Manual 4.11
- User Manual 4.10
- User Manual 4.9
- User Manual 4.8
- User Manual 4.7
- User Manual 4.6.1
- User Manual 4.5.1
- User Manual 4.4
- User Manual 4.3
- User Manual 4.2
- User Manual 4.1
- User Manual 4.0
- User Manual 3.4.1
- User Manual 3.3

Tutorials

- Tutorials 4.14
- Tutorials 4.13.5
- Tutorials 4.12.1
- Tutorials 4.8.1
- Tutorials 4.5.1
- Tutorials 4.4
- Tutorials 4.1
- Tutorials 4.0

Links

- CNES freeware server

Navigation

- Accueil
- Modifications récentes
- Page au hasard
- Aide

Outils

- Pages liées
- Suivi des pages liées
- Pages spéciales
- Adresse de cette version
- Information sur la page
- Citer cette page