2019-05-15
Björn Möller
Andreas Rydell

How to build a SISO Space FOM federate using Pitch Developer Studio (Draft 5)

Contents

About this document 3
Prerequisites 3
What you need 3
Overall process 3
Creating object classes for your federation in Visual OMT 4
Creating the spacecraft 4
Creating the astronaut 6
Creating the CameraSensor 7
Summary 7
Generating the HLA Module in Pitch Developer Studio 8
Develop Code for a Late Joiner 12
Develop Code for an Early Joiner 12
Code examples 14
Java 14
Start sequence 14
C++ 14

About this document

This document provides a cookbook for developing a federate for the SISO Space Reference FOM using code generated from Pitch Developer Studio. Only the HLA module of the federate is covered, i.e. the code that connects to the HLA federation. The code for the simulation models, for example space vehicle dynamics or astronaut behavior is left to the reader to develop.

This document describes how to implement a late joining federate, i.e. a federate that joins into an already existing federation, and which doesn't have any of the predefined roles Master, Pacer or Root Reference Frame Publisher.

Prerequisites

The following is assumed

1. The reader knows programming in C++ or Java and is familiar with the development environment of his choice.
2. The reader has read the following introductory papers, available for download from www.sisostds.org, www.researchgate.net and www.pitchtechnologies.com: "A First Look at the Upcoming Spare Reference FOM" (16F-SIW-017) and "Design and Principles Enabling the Space Reference FOM" (17F-SIW-038)
3. The reader knows the basics of HLA, for example by reading the free HLA Tutorial, available from www.pitchtechnologies.com/tutorial
4. The reader has read the Pitch Developer Studio User's Guide
5. The reader has access to a draft of the Space Reference FOM

This document applies to both C++ and Java development. In case there are differences between these are explicitly described.

What you need

You will need the following:

1. The Space Reference FOM modules, and any additional FOM modules that you want to use, for example if you designed space vehicles with additional attributes
2. Your development environment (compiler, editor, etc)
3. Pitch Developer Studio for generating code
4. Access to a running Space FOM federation. You may also run your own federation using a Master/Pacer federate and a Root Reference Frame Publisher, such as the "Space Master" and "Earth Environment" that Pitch can provide.

Overall process

You will need to go through the following steps, that are covered in this cookbook:

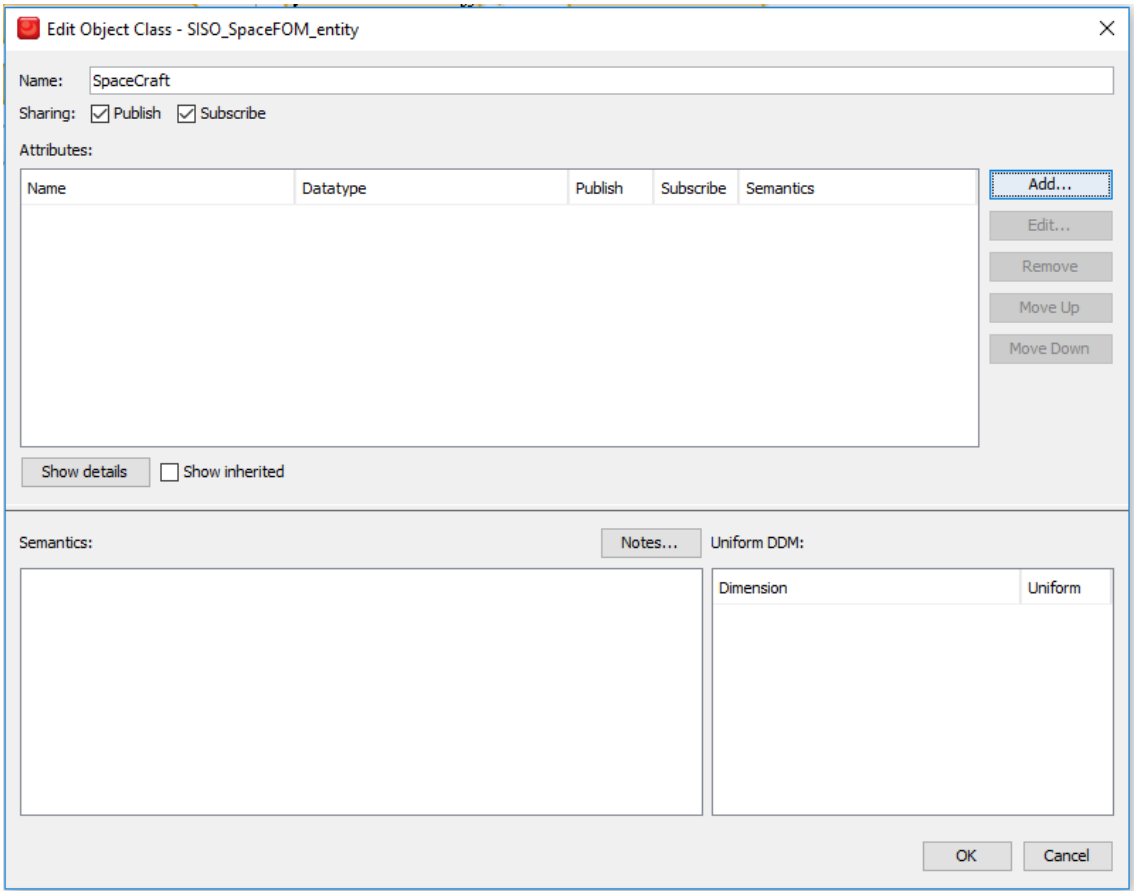
1. Use Visual OMT to create the object and interaction classes that your application needs.
2. Generate an HLA module based on the SISO Space FOM modules using Pitch Developer Studio.
3. Develop code that follows the SISO Space FOM specification for a "Late Joiner".

Creating object classes for your federation in Visual OMT

In this example we will create a spacecraft class together with an Astronaut class and a CameraSensor class. Open Pitch Visual OMT and expand SISO_SpaceFOM_entity and double click on Object Classes.

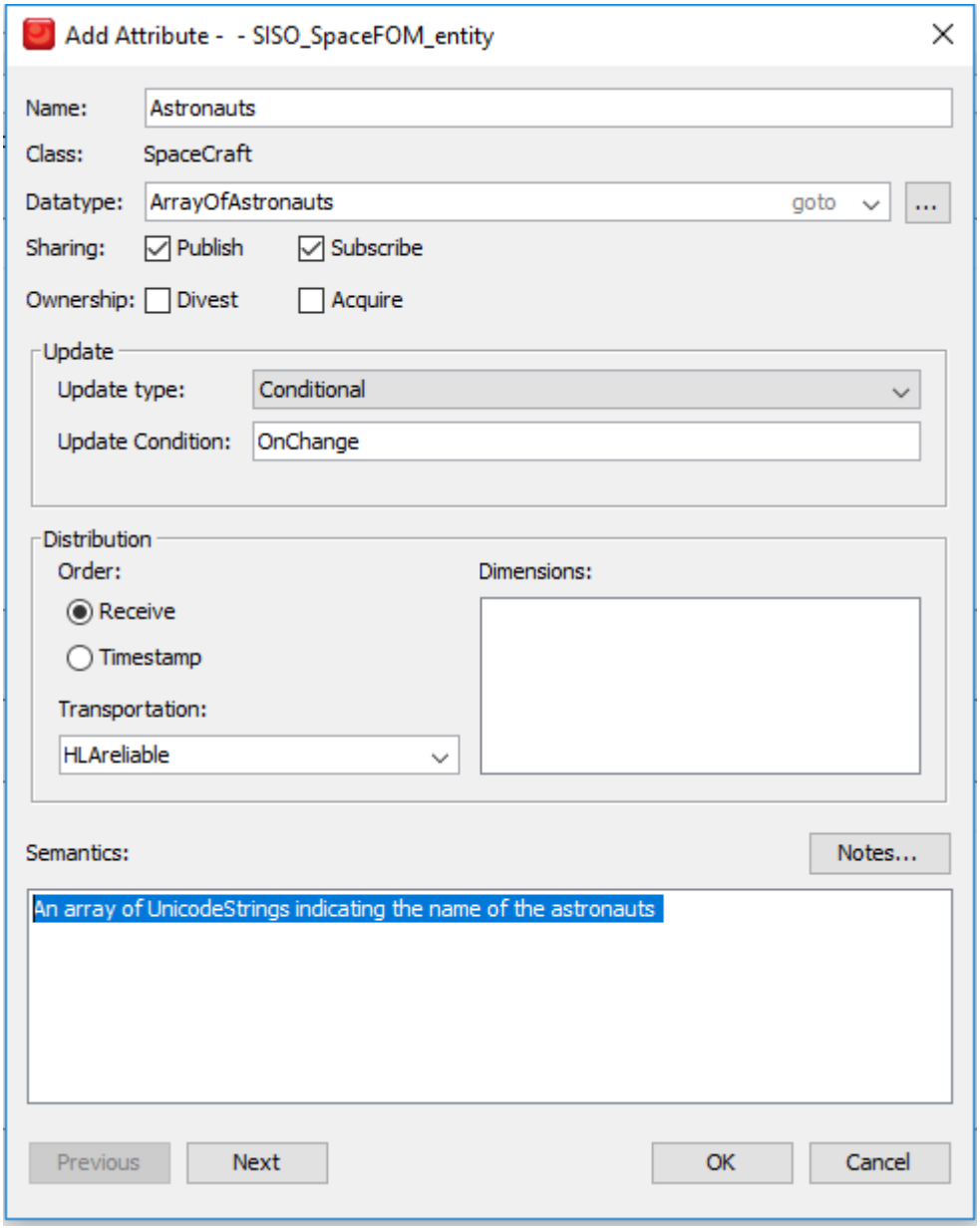
Creating the spacecraft

Right click on DynamicalEntity and select “Add Object Class” or drag and drop from the right margin of the object class. In this example we will create a SpaceCraft. Right click the new object class and edit it. Tick both publish and subscribe boxes.



Picture 1

A spacecraft most likely has a number of Astronauts, so let us add an array of Astronauts. Click Add.. to add an Attribute and name it Astronauts. Set the datatype to ArrayOfAstronauts (do not worry, we will create the datatype soon). Tick both publish and subscribe boxes. Set *Update type* to “Conditional” and the *Update Condition* to “OnChange”. Write a semanticm ex. “An array of UnicodeStrings indicating the name of the astronauts”.



Picture 2

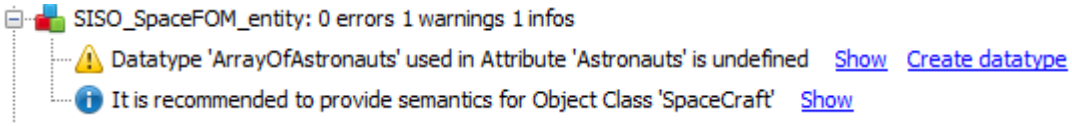
Add another Attribute. This time, name it Camera and let it have *Datatype* "HlaUnicodeString". As before, tick both boxes for publish and subscribe and set the *Update type* to "Static" and *Update Condition* to "NA". In the semantics, add a text, ex. "Name of the camera".

Your object class should look something like this:

SpaceCraft		ps	
Astronauts	ArrayOfAstronauts	ps	ro
Camera	HLAunicodeString	ps	ro

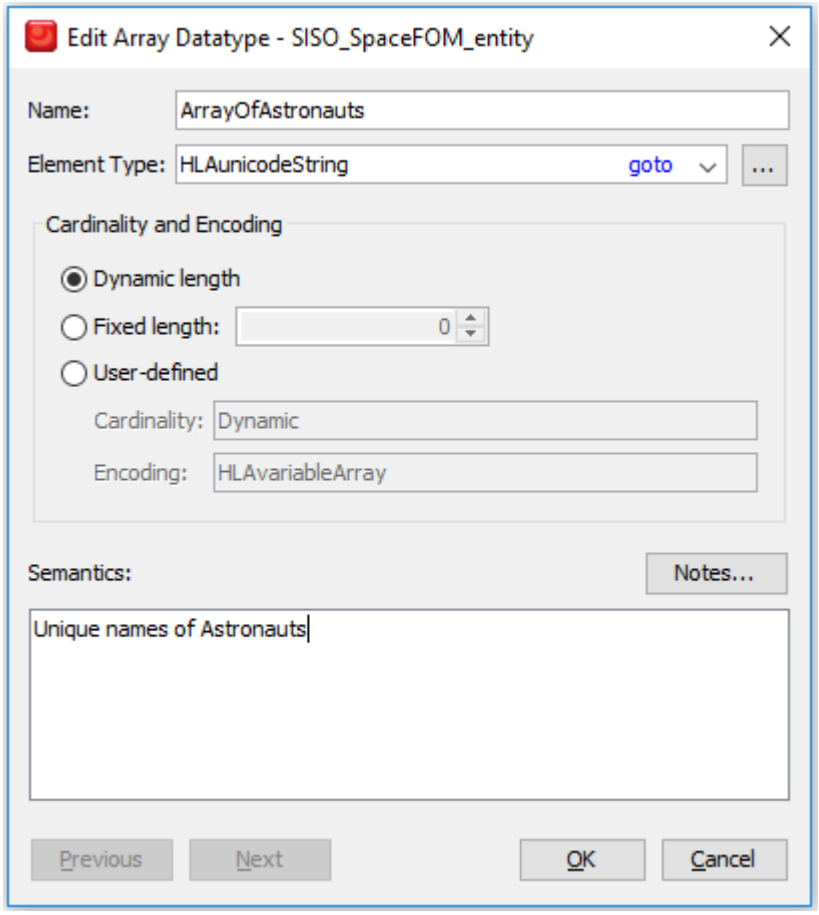
Picture 3

Now, in the bottom field, expand the module SISO_SpaceFOM_entity where you should have one warning as shown below:



Picture 4

Click *Create datatype*, chose Array datatype and click *Create and edit*. Fill in the datatype as shown in the picture below.

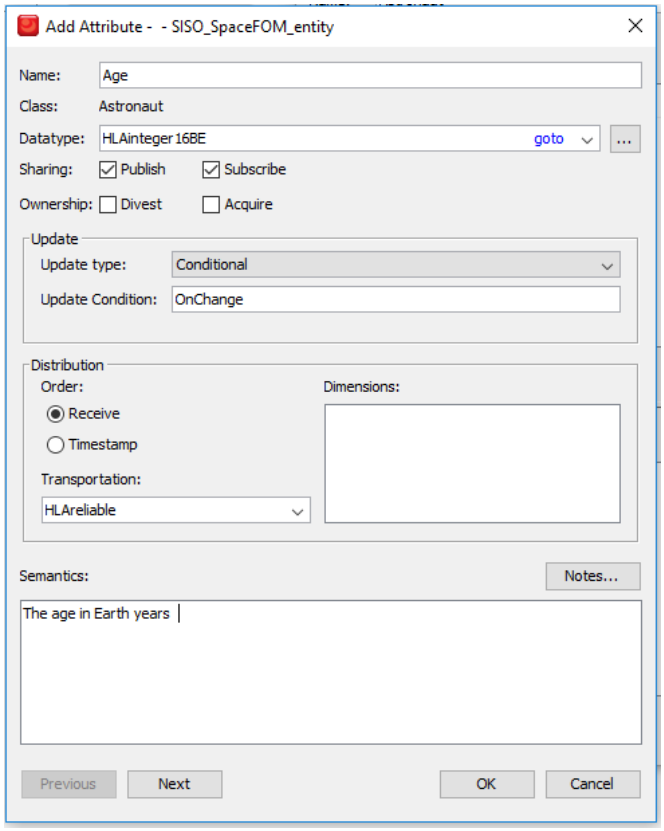


Picture 5

Now we have a spacecraft we can use for code generation. Let us continue with creating a object class for the astronauts.

Creating the astronaut

Go back to the Object Classes of SISO_SpaceFOM_entity. Now add a new object class, extending the *PhysicalEntity* class, call it Astronaut and edit it. As before, tick the publish and subscribe boxes and add an attribute called Age. Let it be an “HLAinteger 16BE”, see picture below



Picture 6

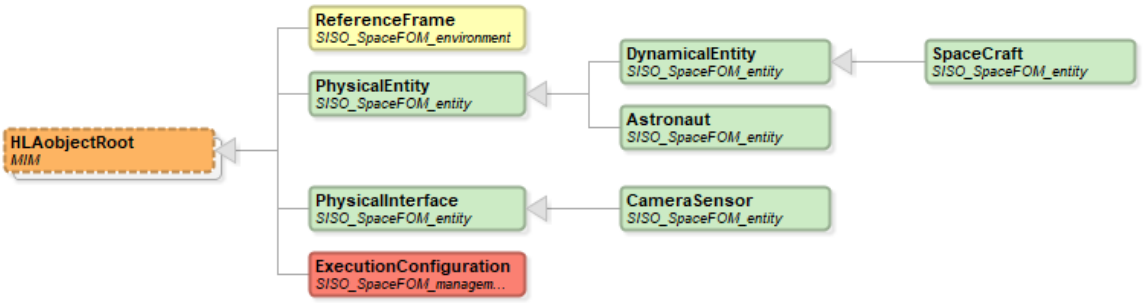
Add another attribute called Name and let it be the datatype “HLAUnicodeString”. The Update type should be static as the names of the astronaut should stay the same and hopefully be unique.

Creating the CameraSensor

Once again, go back to the Object Classes of SISO_SpaceFOM_entity, add a subclass to the *PhysicalInterface* class and name it “ArticulatedPart”. Now create a new object class from the newly created object class *ArticulatedPart*, call it “CameraSensor”. Edit the camera sensor object class and add an attribute called “ZoomLevel”, it should be of the datatype float and have a semantic explaining what it means.

Summary

When you are done, your FOM modules should look something like this:

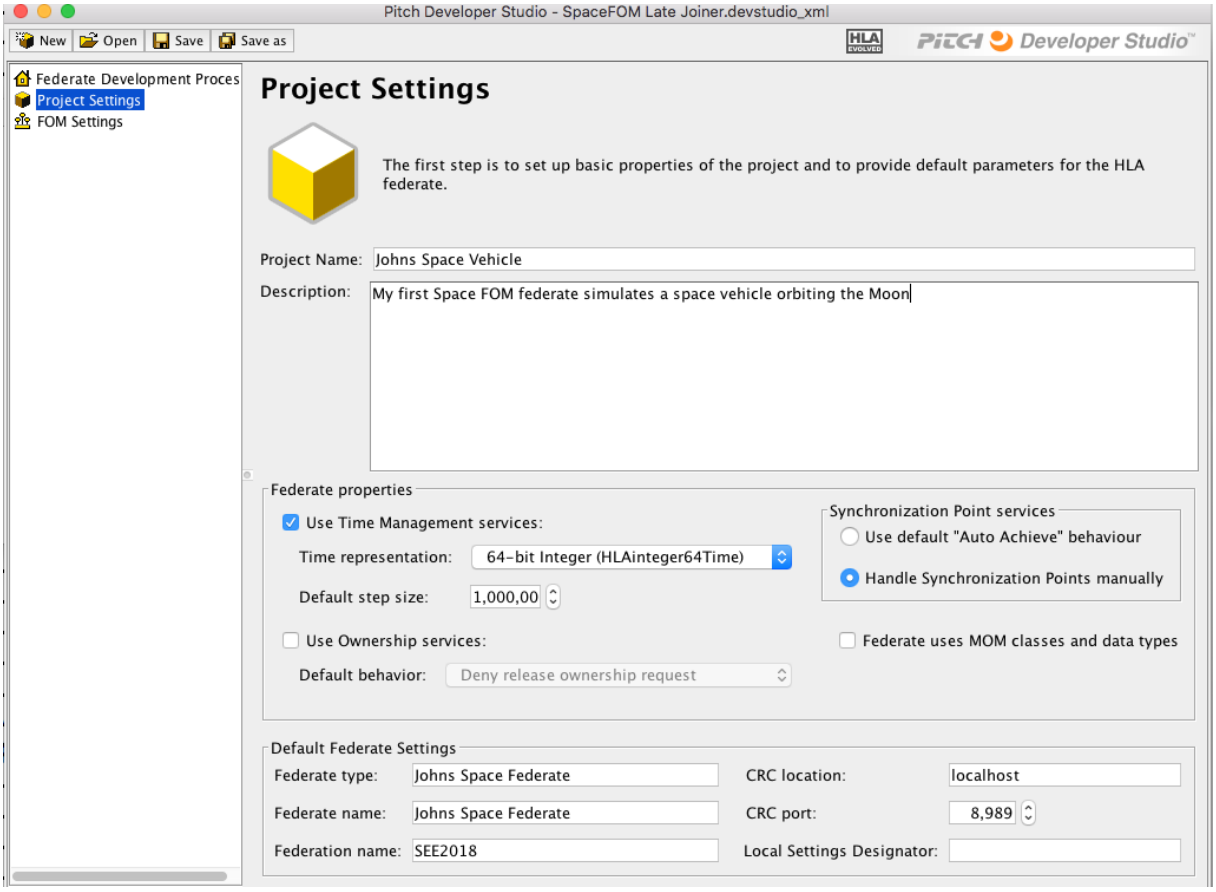


You can access this view by Tools->Project Overview->Object Classes (or the keyboard shortcut of Ctrl+G). Do not forget to save your work.

Generating the HLA Module in Pitch Developer Studio

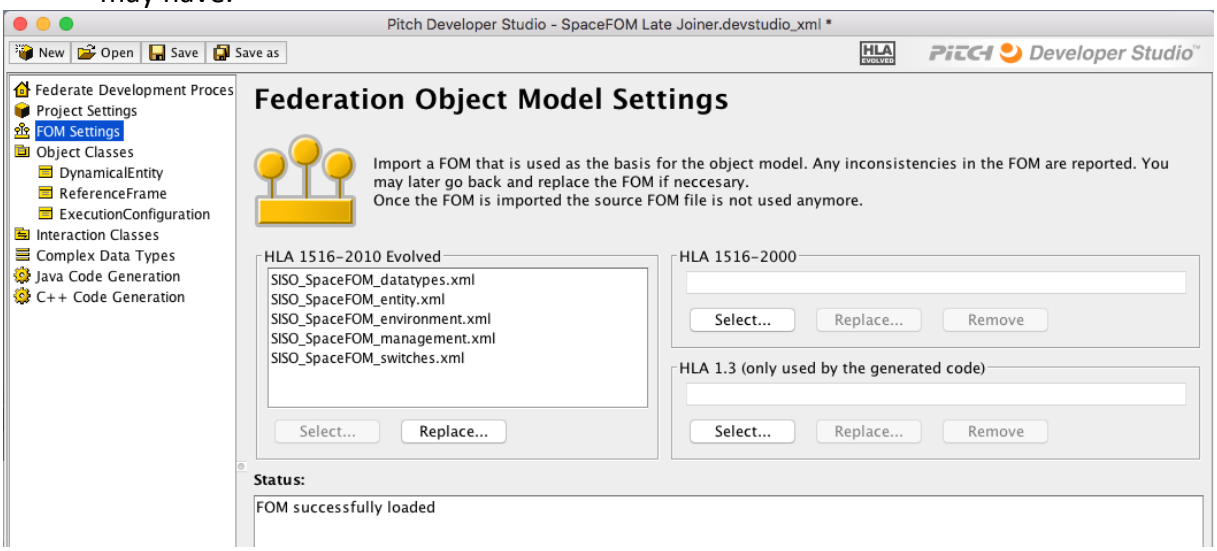
Follow these steps:

1. Start Pitch Developer Studio
2. In the project settings input your project name and description. Select to use Time Management Services with time representation HLAinteger64Time. Use the default step size of 1 000 000 microseconds, i.e. one second. Also select to handle Synchronization points manually. Suggest a name for your federate. The federate name and type can be the same. See the picture below.



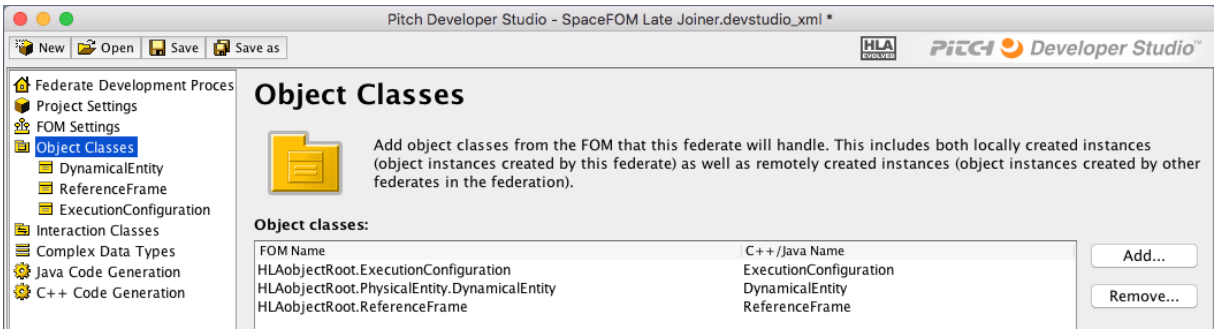
Picture 7

3. Go to the FOM settings and add the Space FOM modules (Data Types, Entity, Environment, Management, Switches) and any additional FOM modules that you may have.



Picture 8

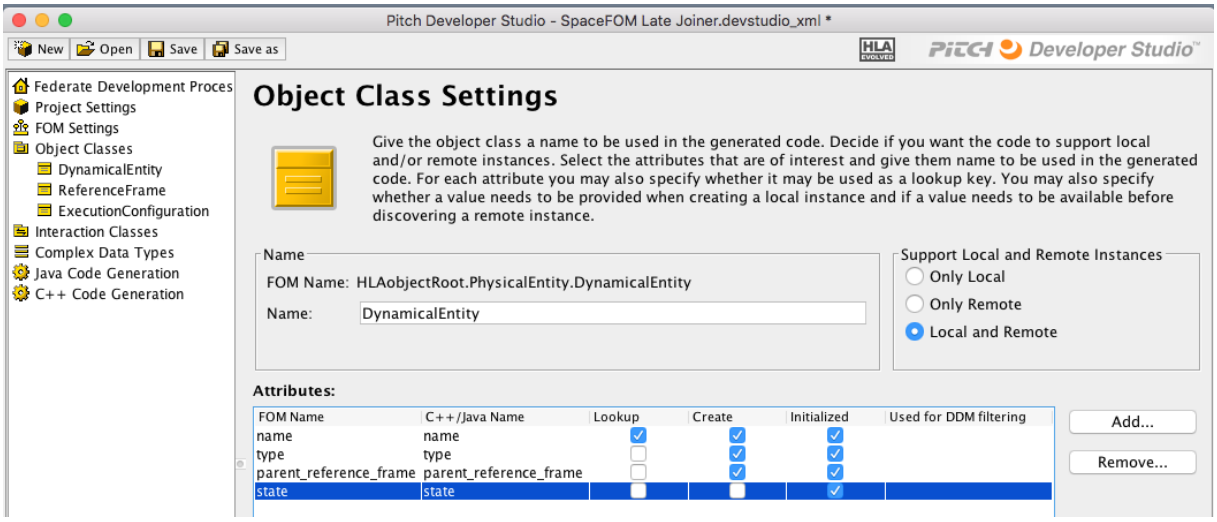
4. Go to the Object Classes settings and select the following classes:
    a. ExecutionConfiguration, since our federate needs to subscribe to the ExCO object instance
    b. PhysicalEntity.DynamicalEntity since our federate needs to publish and subscribe space vehicles. If you created a Space Vehicle subclass you may choose that instead.
    c. ReferenceFrame since our federate needs to discover reference frames in order to specify positions.



Picture 9

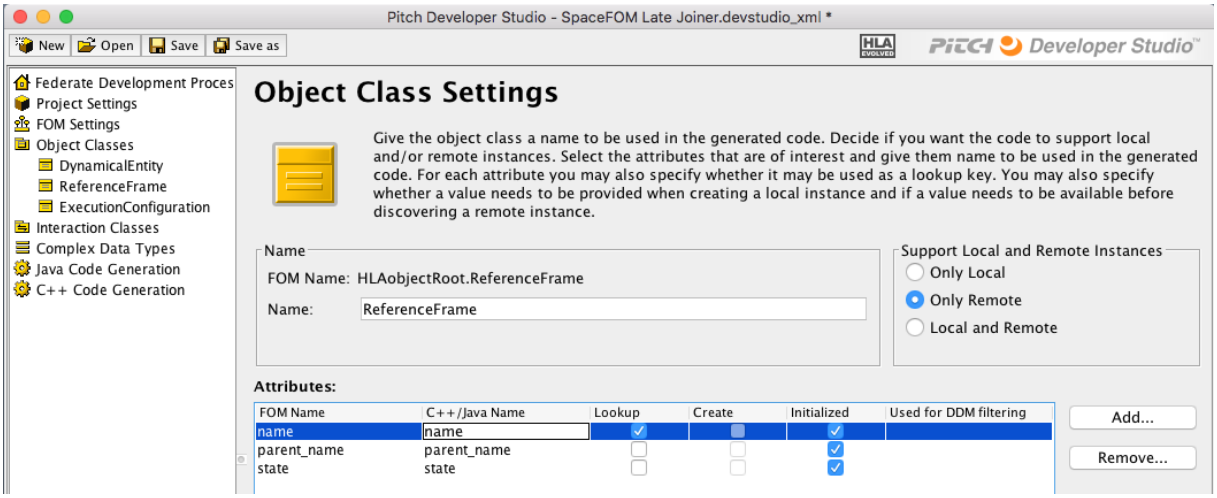
5. Now go to the DynamicalEntity settings.
    a. Select to support both local instances (created by our own federate) and remote instances (created by other federates).
    b. Add at least the attributes “name”, “type”, “state” and “parent_reference_frame” attributes.
    c. In the Lookup column, select that we want an automatic lookup table for the attribute “name”

d. In the Create column, select that we want to provide the following attribute values when creating a new DynamicalEntity: “name”, “type”, parent_reference_frame”.
e. In the Initialized column, select that we want the HLA module to let our code know about a remote object instance when we have received values for “name”, “type”, parent_reference_frame” and “state”. This will prevent us from getting C++/Java instances that don’t have a value (yet) for important attributes.



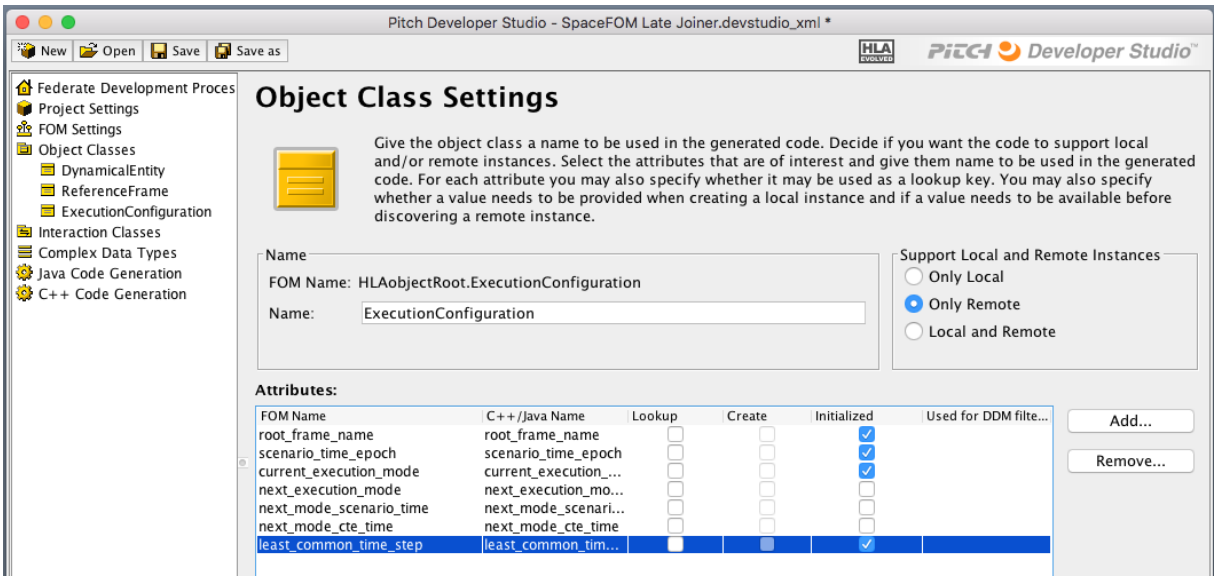
Picture 10

6. Now go the Reference Frame settings. In our federate we will only discover reference frames.
    a. Select to support only remote instances (created by other federates).
    b. Add the attributes “name”, “type”, “state” and “parent_reference_frame”.
    c. In the Lookup column, select that we want an automatic lookup table for the attribute “name”
    d. In the Initialized column, select that we want the HLA module to let our code know about a remote object instance when we have received values for “name”, “type”, parent_name” and “state”.



Picture 11

7. Now go the Execution Configuration settings. In our federate we will only discover the ExCO instance of this class.
    a. Select to support only remote instances (created by other federates).
    b. Add all attributes.
    c. In the Initialized column, select that we want the HLA module to let our code know about a remote object instance when we have received values for all attributes except next_execution_mode, next_mode_scenarion_time and next_mode_cte_time.



Picture 12

There is no need to set up any interactions in this simple case. In case you have a federate that wants to set up a mode transition, you may consider adding support for such an interaction.

Develop Code for a Late Joiner

Using Developer Studio 5.0.1 and later, you can start your federates without any time management enabled. This is done in java by adding the following line to your code before connecting your federate to the federation:

```
System.setProperty("[your-package-
name].hla.tuning.enableTimeManagementOnConnect", Boolean.toString(false));
```

To use this feature in C++, you will have to use the FederateConfig.txt (Read more about the FederateConfig.txt in the Developer Studio Users Guide.

Here is a step by step guide over what to do in your source code to develop a late joiner federate:

1. Create your HlaWorld and listeners (and add them to the HlaWorld), then connect.
2. Check that the synchronization point `mtr_shutdown` is not announced.
    a. If it is, shut down your federate immediately.
3. Wait for the synchronization point `initialization_completed` before doing anything else. *
4. Register your objects. (You will get callbacks while you wait for the `initialization_completed` synchronization point. The callbacks are most likely the ExCO object and other objects depending on your subscription. Note that you should not achieve the synchronization point `initialization_completed`.)
5. Enable time management. (This is done in Java and C++ with `_hlaWorld.enableTimeManagement()`).
6. Advance to the Highest Logical Time Boundry (HLTB) by asking for the Greatest Available Logical Time (GALT) and compute the HLTB by using the formula $HLTB = (floor\ (GALT/LCTS) + 1) * LCTS$.
7. Look at the ExCO's current and next execution mode. If a transition is in progress, make the transition during the next mode scenario time.

* To wait for a synchronization point, use the achieveSynchronizationPointAndWaitForSynchronized method, located under hlaWorld.getHlaSynchronizationManager().

Develop Code for an Early Joiner

Using Developer Studio 5.0.1 and later, you can start your federates without any time management enabled. This is done in java by adding the following line to your code before connecting your federate to the federation:

```
System.setProperty("[your-package-
name].hla.tuning.enableTimeManagementOnConnect", Boolean.toString(false));
```

To use this feature in C++, you will have to use the FederateConfig.txt (Read more about the FederateConfig.txt in the Developer Studio Users Guide.

Here is a step by step guide over what to do in your source code to develop a late joiner federate:

1. Create your HlaWorld and listeners (and add them to the HlaWorld), then connect.
    a. The minimum required listeners needed are:
        i. HlaWorldListener (in order to check if you connected and request time advancement).
        ii. HlaExecutionConfigurationManagerListener (in order to discover and save the HlaExecutionConfiguration/ExCO object).
        iii. HlaExecutionConfigurationValueListener (in order to listen to changes to the execution).
        iv. HlaSynchronizationListener (in order to listen to synchronization points and achieve the right ones).
2. Check that the synchronization point `mtr_shutdown` is not announced.
    a. If it is, shut down your federate immediately.
3. Check if the synchronization point `initialization_started` is announced and still active.
    a. If it is, then your federate is a late joiner.
    b. If not, continue to step 4.
4. Wait for the synchronization points `objects_discovered`, `root_frame_discovered`, `initialization_started` and all MPI synchronization points to be announced. (The MultiPhase Initialization (MPI) synchronization points may or may not exist depending on your scenario) *
5. Create and register your objects and wait for all objects you require for the scenario.
6. When you have registered all your objects and received all remote objects that you require, achieve the synchronization point `objects_discovered` and wait for synchronization. *
7. Wait for the ExCO to set the Epoch. **
8. Wait for the ExCO to set the `root_frame_name` attribute. **
9. Achieve the synchronization point `root_frame_discovered` and wait for synchronization. *
10. If you use MPI synchronization points, send and wait to receive updates you require for that specific MPI synchronization point.
11. Achieve the MPI synchronization point and wait for synchronization. *
    a. If more MPI synchronization points exist, repeat step 10.
8. Enable time management. (This is done in Java and C++ with `_hlaWorld.enableTimeManagement()`).
12. Achieve the syncrhonization point `initialization_started` and wait for synchronization.
13. Wait for ExCO to update the `next_execution_mode` and take steps accordingly.

* To wait for a synchronization point, use the achieveSynchronizationPointAndWaitForSynchronized method, located under hlaWorld.getHlaSynchronizationManager().
** To wait for something specific, use a CountDownLatch or something similar (depending on the programming language you are using).

Code examples

Java

Start sequence

```
createHlaWorld();
createListeners();
addListeners();
connect();
_connectCountDownLatch.await(); //Wait until we're connected
checkShutdown(); //Check if the synchronization point mtr_shutdown is
announced, then shut down
if (checkInitializationStarted()) {
   System.out.println(INITIALIZATION_STARTED + " already announced, joining
as late joiner");
   lateJoinerStart();
} else {
   System.out.println(INITIALIZATION_STARTED + " not announced, joining as
early joiner");
   earlyJoinerStart();
}

private void earlyJoinerStart(){
   waitForObjectsDiscoveredAndMPISynchPoints();
   createMyObjects(); //create objects and register them
   _objectsDiscoveredCountDownLatch.await(); //wait for all my required
objects to be discovered
   achieveSyncPointAndWait(OBJECTS_DISCOVERED);
   waitForEpoch();
   waitForRootReferenceFrame(); //Wait until the root reference frame name
is set by ExCO
   achieveSyncPointAndWait(ROOT_FRAME_DISCOVERED);
   MPISynchPoints();
   enableTimeManagement();
   achieveSyncPointAndWait(INITIALIZATION_STARTED);
   //listen to next execution mode and follow accordingly
}
```

C++

To be done