JSC XXXXX-DD.mm.20

Trick High Level Architecture
TrickHLA
User Guide

Simulation and Graphics Branch (ER7)
Software, Robotics and Simulation Division
Engineering Directorate

Package Release TrickHLA v3.0.0 - Beta

Document Revision 1.0
June 2020

National Aeronautics and Space Administration
Lyndon B. Johnson Space Center
Houston, Texas

JSC XXXXX-DD.mm.20

Trick High Level Architecture
TrickHLA
User Guide

Document Revision 1.0
June 2020

Edwin Z. Crues
and
Daniel E. Dexter

Simulation and Graphics Branch (ER7)
Software, Robotics and Simulation Division
Engineering Directorate

National Aeronautics and Space Administration
Lyndon B. Johnson Space Center
Houston, Texas

Abstract

The `TrickHLA` model provides an abstraction of the IEEE-1516 High Level Architecture (HLA) in the Trick Simulation Environment allowing a developer to concentrate on simulation development without needing to be an HLA expert. This document is a users guide for engineers and developers seeking to use `TrickHLA` in their simulations.

Contents

1 Introduction **1**

1.1 Identification of Document 1

1.2 Scope of Document 1

1.3 Purpose and Objectives of Document 1

1.4 Documentation Status and Schedule 1

1.5 Document Organization 2

2 Related Documentation **4**

2.1 Parent Documents 4

2.2 Applicable Documents 4

3 Preliminaries **5**

3.1 Background 5

3.2 Important Trick Concepts 5

3.3 Important HLA Concepts 6

3.4 The `simplesine` Model 8

3.5 Federation Object Model 11

4 An Example `simplesine` Simulation **14**

4.1 `SIM_simplesine_pubsub` 14

4.2 `S_define` 14

4.3 Input Files 16

4.4 Output 17

5 Joining a Federation **21**

5.1 `SIM_simplesine_hla_join` 21

5.2 Input Files . . . 23
5.3 Output . . . 25

6 Publishing and Subscribing **28**
6.1 `SIM_simplesine_hla_pub` . . . 28
6.2 `SIM_simplesine_hla_sub` . . . 29
6.3 Publisher input file . . . 30
6.4 Subscriber input file . . . 31
6.5 Output . . . 33

7 Lag Compensation **36**
7.1 What is a Lag Compensator? . . . 36
7.2 Publisher-resident Compensation . . . 39
7.3 Subscriber-resident Compensation . . . 42

8 Sending and Receiving Interactions **45**
8.1 What is an interaction handler? . . . 45
8.2 `SIM_simplesine_hla_sendInt` . . . 48
8.3 Sender input . . . 48
8.4 `SIM_simplesine_hla_receiveInt` . . . 49
8.5 Receiver input . . . 49
8.6 Output . . . 50

9 Ownership Transfer **52**
9.1 What is an ownership handler? . . . 52
9.2 `SIM_simplesine_hla_own` . . . 54
9.3 Active input file . . . 54
9.4 Passive input file . . . 55
9.5 Output . . . 56

10 Data Encoding and Packing **60**
10.1 What is a *packing* class? . . . 60
10.2 `SIM_simplesine_hla_pack` . . . 63
10.3 `SIM_simplesine_hla_unpack` . . . 64

10.4 Output . . . 64

11 Initialization **66**

11.1 What is multiphase initialization? . . . 66

11.2 `DSESSimConfig` . . . 66

12 Timeline **70**

12.1 What is the *TrickHLATimeline* class? . . . 70

12.2 `S_define` file . . . 72

12.3 `input` file . . . 72

13 Object Deleted Notification **73**

13.1 What is an *ObjectDeleted* class? . . . 73

13.2 `S_define` file . . . 75

13.3 `input` file . . . 75

13.4 Output . . . 75

14 Upgrading Your Federate To Initiate a Federation Save **77**

14.1 Trick `input` file update . . . 77

14.2 `S_define` file updates . . . 77

15 Federation Save **81**

15.1 Interactive Save . . . 81

15.2 Programmatic Save . . . 82

16 Federation Restore **84**

16.1 Interactive Restore . . . 84

16.2 Programmatic Restore . . . 84

17 Conditional sending of cyclic data **86**

17.1 How do you send cyclic data conditionally? . . . 86

17.2 `SIM_sine_conditional_cyclic` . . . 90

17.3 Input . . . 91

17.4 Output . . . 93

A `simplesine` Files **100**

A.1 simplesine.h 100
A.2 simplesine_proto.h 101
A.3 simplesine_InteractionHandler.hh 102
A.4 simplesine_LagCompensator.hh 102
A.5 simplesine_Packing.hh 103

B Interaction send/receive input files **104**
B.1 Complete sender input file 104
B.2 Complete receiver input file 107

Chapter 1

Introduction

The objective of `TrickHLA` is to simplify the process of providing simulations built with the Trick Simulation Environment[8] with the ability to participate in distributed executions using the High Level Architecture (HLA)[11]. This allows a simulation developer to concentrate on the simulation and not have to be an HLA expert. `TrickHLA` is data driven and provides a simple API making it relatively easy to take an existing Trick simulation and make it HLA capable.

1.1 Identification of Document

This document describes the use of the `TrickHLA` developed for use in the Trick Simulation Environment. This document adheres to the documentation standards defined in NASA Software Engineering Requirements Standard [7].

1.2 Scope of Document

This document provides information on the use of the `TrickHLA`.

1.3 Purpose and Objectives of Document

The purpose of this document is to describe how to incorporate the `TrickHLA` into a dynamic Trick simulation and used by other simulation models.

1.4 Documentation Status and Schedule

The information in this document is current with the TrickHLA v3.0.0 - Beta implementation of the `TrickHLA`. Updates will be kept current with module changes.

Author	Date	Description
Edwin Z. Crues	June 2020	Initial Version

Revised by	Date	Description

1.5 Document Organization

This document is organized into the following sections:

Chapter 1: Introduction – Identifies this document, defines the scope and purpose, present status, and provides a description of each major section.

Chapter 2: Related Documentation – Lists the related documentation that is applicable to this project.

Chapter 3: Preliminaries – Discusses some Trick, HLA and `TrickHLA` concepts that are used in the subsequent chapters.

Chapter 4: An Example `simplesine` Simulation – Introduces the `simplesine` model in the context of a non-HLA simulation.

Chapter 5: Joining a Federation – Illustrates how to make a Trick simulation become an HLA federate and join an HLA federation.

Chapter 6: Publishing and Subscribing – Shows how Trick simulations may publish and subscribe data.

Chapter 7: Lag Compensation – Demonstrates how HLA-induced lags due to time management may be removed either by publihsers of the data or by subscribers.

Chapter 8: Sending and Receiving Interactions – Illustrates how to send and receive HLA interactions.

Chapter 9: Ownership Transfer – Shows how to use the `TrickHLA` mechanisms for *pushing* and *pulling* ownership of HLA data.

Chapter 10: Data Encoding and Packing – Shows how simulation developers may pack and unpack HLA data.

Chapter 11: Initialization – Explains how to use `TrickHLA` to implement multiphase federation initialization.

Chapter 13: Object Deleted Notification – Explains how to use `TrickHLA` to implement callbacks when an object was deleted from the federation.

Chapter 14: How to setup your trick federate to initiate a federation save – Explains how to upgrade your trick model to utilize `TrickHLA` routines to save the federate.

Chapter 15: Federate Save – Explains how to initiate a federation save from the trick model.

Chapter 16: Federate Restore – Explains how to initiate a federation restore from the trick model.

Bibliography – Informational references associated with this document.

Appendix A: `simplesine` Files – Provides listings of some of the `simplesine` source files.

Appendix B: Interaction send/receive input files – Provides listings of the input files for the simulations that send and receive HLA interactions.

Chapter 2

Related Documentation

2.1 Parent Documents

The following documents are parent to this document:

- *Trick High Level Architecture (TrickHLA)* [2]

2.2 Applicable Documents

The following documents are referenced herein and are directly applicable to this document:

- *Distributed Space Exploration Simulation Multiphase Initialization Design* [5]
- *Integrated Mission Simulation Multiphase Initialization Design* [6]
- *TrickHLA Product Requirements* [3]
- *TrickHLA Product Specification* [4]
- *TrickHLA Inspection, Verification, and Validation* [1]
- *Trick Simulation Environment: Installation Guide* [9]
- *Trick Simulation Environment: Tutorial* [10]
- *Trick Simulation Environment: Documentation* [8]
- *NASA Software Engineering Requirements* [7]

Chapter 3

Preliminaries

3.1 Background

`TrickHLA` is a *glue layer* between Trick and HLA. As such, using it to develop distributed simulations requires that you have an understanding of Trick itself and to a lesser extent be familiar with HLA terminology. This section reviews the concepts that the `TrickHLA` model is based on.

3.2 Important Trick Concepts

`S_define` Files. Trick is a simulation development environment that is used to build (compile, link, etc...) executable images for your simulations. The details of what your simulation consists of (which things are being simulated) and what dynamic scenario is simulated (what happens during the simulation) are specified by you in the so-called `S_define` file.

This file contains declarations of *sim objects* (the things) and *jobs* (what happens). The sim objects correspond to C/C++ data structures defined in models written by you, and the jobs are C/C++ functions (or methods) which are also part of the models.

`TrickHLA` is a Trick model and hence consists of sim objects and jobs that you assemble into your `S_define` file alongside the objects and jobs that make up your particular simulation. Indeed, integrating `TrickHLA` with your simulation consists mostly (but not completely) of pasting in an `S_define` snippet which provides much of the logic (objects and jobs) necessary for your simulation to participate in an HLA federation.

Input Files. Trick is based on a philosophy of *data driven* simulation. Trick models are aggressively parameterized, and the parameters are intended to be driven from initial data resident in input files. This permits significant variations of a particular simulation to be run without actually rebuilding (recompiling and relinking) the simulation itself.

A Trick input file is a text file consisting of name/value pairs of data: the names specify specific model parameters, and the values specify the initial data to be assigned to those paramters. Trick has an input processor which parses the input file and makes sure that the various variables in the

simulation are initialized accordingly.

`TrickHLA` is a Trick model, and so configuring it involves setting certain `TrickHLA` parameters in your sim input files.

Enabling/disabling jobs. In some of the simulations in this document, we will want to enable/disable certain Trick jobs without actually removing them from the `S_define` file, since changing the `S_define` file requires that the simulation be recompiled. This can be done from the input file by using the `JOB` directive:

```
JOB job-name = [On|Off];
```

where `job-name` can be found in the list of all job-related parameters in the Trick-generated file, `S_default.dat`.[1] Thus, to enable/disable a particular job, you just change the input value in the directive between `On` and `Off`. You need one such directive for each job you wish to disable. Of course, all jobs without a corresponding `JOB` directive are enabled.

Calling jobs at select times. In some of the simulations in this document, rather than invoking Trick jobs periodically from the `S_define` file, we want to invoke them at select times during the run. This is handled by the `READ` and `CALL` directives:

```
READ = T;
CALL job-name;
```

`job-name` can be found in the list of all job-related parameters in `S_default.dat`. This syntax tells Trick to call the specified job at time `T`. If such a line occurs once in the input file (for only one value of `T`), then Trick will only invoke the specified job once at time, $t =$ `T`.

One subtle point must be made about the `CALL` directive. If your input file uses `CALL` to invoke a job that is otherwise *not* part of the Trick job schedule (as defined in the `S_define` file), then you **must** declare the job in the `S_define` with a frequency of zero. This ensures that Trick generates the necessary code for the job in spite of the fact that it is not in fact called by the Trick scheduler. If you do not do this, you will get a runtime error during the execution of your simulation.

3.3 Important HLA Concepts

Federations and Federates. An association of possibly distributed processes cooperating using HLA is called a *federation*. Processes may *join* a federation to participate, and they may also *resign*. The first federate to join a federation generally *creates* it, and the last one to resign usually *destroys* it.

[1] This file is one of the files created by `CP` when you build your simulation. It is located in the same directory as the `S_define` file.

Objects and Interactions. There are two kinds of data in HLA: *objects* and *interactions.*

Objects are used to model data that are persistent over time, entities which will evolve as the federation executes. Objects are composed of *attributes,* and are as such very similar to the classes of traditional object oriented languages. Objects provide the abstract framework for how persistent data are structured. Specific occurences of the data in a federation are refered to as *instances*, and have federation-wide unique names. It is these instances that persist during the execution of the federation.

Interactions are used to model events which contain data but are not persistent; they are notifications. Interactions are composed of *parameters.*[2] Specific occurences of interactions are delivered to federates by the HLA API as they are sent and/or received, and there is no concept of an "instance" of an interaction.

Publish/subscribe. Federates that can generate values for particular instance attributes are said to *publish* them.[3] Publishing is on a per-attribute basis. Thus several federates may be involved in generating the values for the attributes of a particular instance. Furthermore, many federates may declare that they are publishers of a particular attribute (i.e., that they have the ability to generate new values for it); however, only one may actually generate new values: this one federate is referred to as the *owner*. When an owner generates new values for an attribute, it is said to *update* them. When a non-owner receives new values for an attribute from the owner, the receiver is said to *reflect* the values. During the execution of a federation, ownership of a particular attribute may move from one federate to another. This is called *ownership transfer*. The act of giving up ownership of an attribute is called *divestiture*. The act of assuming ownership of an attribute *acquisition*.

Send/receive. A federate may *send* a interaction to any other interested federates. When the interaction arrives at the interested federates, they are said to *receive* it. These are on a per-interaction basis (not per-parameter). When an interaction is sent, values for each of its parameters are specified by the sender, and the entire interaction with each of its parameters is delivered to any receivers.

Federation Object Model. HLA federations each have a specific *federation object model (FOM)* that defines the structure of the types of objects and interactions that may be exchanged by participating federates. The information included in the FOM specifies the names and data types of the shared object classes and attributes and interactions and parameters. (It does not document specific object instances, however.) The format of the FOM is not specified by the HLA standard, but the Pitch implementation of HLA (currently used for NASA/DSES HLA simulations) uses an XML format.

Runtime Infrastructure. Most distributed computing envronments require some additional processes in order to help coordinate communication between federates. For HLA, this component is called the *runtime infrastructure (RTI)*. The Pitch RTI (currently used for NASA/DSES HLA

[2] Parameters are to interactions as attributes are to objects.

[3] Note that the use of the term publish in HLA only indicates a federate's *ability* to generate values and not the actual act of generating them.

simulations) is a single process which runs at a well-known TCP port number on a well-known host on the network. Each federate much be configured with this RTI host/port information in order to join the federation.

Time Management. One of the unique capabilities HLA provides is a mechanism for keeping all the federates in a federation synchronized. In the HLA context, this means making sure that all federates see the same sequence of data (instance attributes and interaction parameters) at the same federation-time. The HLA services behind this are collectively referred to as *time management*.

The HLA concept of *lookahead* makes this possible: each federate declares a lookahead (measured in units of time), and any message sent by that federate (attribute update or interaction send) must have a time stamp greater than or equal to the current time plus the lookahead, i.e., the federate must be able to extrapolate the value slightly into the future in order to update its value. When data is delivered in this synchronized manner, it is said to be *time stamp ordered*. (If this synchronization is disabled, data delivery is said to be *receive ordered*).

Federates participate in the HLA time management services by specifying whether or not they are *time regulating* and *time constrained*. Time regulating federates may generate TSO data, and the advance of federation time proceeds only with the explicit agreement of such federates. Time contrained federates may receive TSO data, but they do not have a voice in whether or not time in the federation may advance. Federates that are only time regulating are data sources. Federates that are only time constrained are passive listeners to the data. The most common case of for a federate to be both, in which case it may generate and receive data. HLA only increments federation time due to explicit requests from federates – *time advance requests*. When the HLA runtime infrastructure acknowledges such requests, it is said to provide *time advance grants*.

3.4 The `simplesine` Model

In the following chapters, we use a simple sine wave model, `simplesine`, to illustrate `TrickHLA` in action. Understanding the model and how it is used in a Trick `S_define` file is important for those chapters. This section introduces `simplesine` with that in mind.

3.4.1 Description

The system modeled by `simplesine` is an undamped harmonic oscillator, the dynamics of which are governed by the differential equation

$$\ddot{x} + w^2 x = 0, \tag{3.1}$$

which has an analytic solution of the form

$$x(t) = A \sin(\omega t + \phi), \tag{3.2a}$$
$$\dot{x}(t) = A\omega \cos(\omega t + \phi). \tag{3.2b}$$

The relevant model data are the dynamic *state*, $(x, \dot{x})$ and the constant system *parameters*, (A, ϕ, ω), where the parameters are specified as inputs and the state is calculated dynamically as simulation

outputs.[4]

The `simplesine` model has functions that may be used to calculate the state analytically based on equations 3.2. It can also propagate the state approximately based on numerical integration of the differential equations. The integration involves the calculation of the derivative of the 2-vector $\boldsymbol{z}$ defined as

$$\boldsymbol{z}(t) \equiv \left\{ \begin{array}{c} x(t) \\ \dot{x}(t) \end{array} \right\} = \left\{ \begin{array}{c} A\sin(\omega t + \phi) \\ A\omega\cos(\omega t + \phi) \end{array} \right\} \tag{3.3}$$

So that

$$\dot{\boldsymbol{z}}(t) = \left\{ \begin{array}{c} \dot{x}(t) \\ \ddot{x}(t) \end{array} \right\} = \left\{ \begin{array}{c} \dot{x}(t) \\ -\omega^2 x(t) \end{array} \right\} = \left\{ \begin{array}{c} Aw\cos(\omega t + \phi) \\ -Aw^2\omega\sin(\omega t + \phi) \end{array} \right\} \tag{3.4}$$

3.4.2 Model

The source code for the `simplesine` model is organized into three directories: `data`, `include` and `src`. The files in these directories are discussed below.

3.4.2.1 Include Files

The `simplesine` C/C++ include files are in directory `simplesine/include`. A list of the files is shown below.

filename
`simplesine.h`
`simplesine_InteractionHandler.h`
`simplesine_LagCompensator.h`
`simplesine_Packing.h`
`simplesine_proto.h`

`simplesine.h` This file declares the fundamental `simplesine` data structures. There is a single data structure that in turn holds state- and parameter-related data structures. In the simulations that follow, the `simplesine_T` structure will be frequently declared as a `sim_object` in the `S_define` files when the simulations need to model a sine wave.

The `simplesine_T` structure consists of a *state* substructure, which holds x and $\dot{x}$ and a parameters substructure which holds the constant *parameters,* A, ϕ and ω. In this model, only the parameters may be set from the input processor. The state may only be calculated by a Trick job. The purpose of this is to ensure that the state and parameters are never initialized to inconsistent values.[5]

The complete file is shown in Appendix A.1.

[4] In this model, the initial state cannot be specified as inputs explicitly but rather through the parameters A and ϕ.

[5] Of course, this requires that developers remember to explicitly call `simplesine_calc()` as an initialization job for every `simplesine` sim variable.

`simplesine_proto.h` This file declares the C functions exported by the `simplesine` model. These functions may be used in an `S_define` file as Trick jobs. Functions of particular interest are

- `simplesine_calc()`, which calculates the state, $(x, \dot{x})$ according to equations 3.2,[6]
- `simplesine_deriv()` and `simplesine_integ()`, which are used to numerically integrate equations 3.3 using the standard Trick integration scheme,
- `simplesine_copyXXX()`, several routines which copy `simplesine` data from one data structure to another, and
- `simplesine_calcError()`, which calculates the error between one `simplesine` state and the true values based on equations 3.2.

The complete file is shown in Appendix A.2.

`simplesine_InteractionHandler.hh` This file declares the `simplesine` C++ class which acts as a `TrickHLA` *interaction handler*. It is a subclass of `TrickHLAInteractionHandler`, which is the `TrickHLA` class which defines how interactions are sent and received.

The complete file is shown in Appendix A.3.

`simplesine_LagCompensator.hh` This file declares the `simplesine` C++ class which acts as a `TrickHLA` *lag compensator*. It is a subclass of `TrickHLALagCompensator`, which is the `TrickHLA` class which defines how federates may compensate for HLA-time lags created as a result of sending data to remote federates and transfering ownership between federates.

The complete file is shown in Appendix A.4.

`simplesine_Packing.hh` This file declares the `simplesine` C++ class which may be optionally used by developers to *pack* outbound data prior to sending via HLA and *upack* is upon receipt from HLA. It is a subclass of `TrickHLAPacking`, which has `pack()` and `unpack()` virtual methods that implement the application-specific packing and unpacking logic.

The complete file is shown in Appendix A.5.

3.4.2.2 Source Files

The `simplesine` C/C++ source files are in the directory `simplesine/src`.

The implementation of the `simplesine` functions and classes is in `.c` and `.cpp` files located in the `simplesine/src` directory. The C functions are mainly implemented one function per file[7], and the C++ classes are implemented one class per file. The file names are shown in the table below:

[6] Since only the `simplesine` parameters have default values, this job may be used as an initialization job to initialize the state from the parameters.

[7] The copy functions (`simplesine_copyParams()`, `simplesine_copyParams()`, and `simplesine_copyParams()`) are located in a single file, `simplesine_copy.c`.

filename	*implements what?*
`simplesine_calc.c`	`simplesine_copy()`
`simplesine_calcError.c`	`simplesine_calcError()`
`simplesine_compensate.c`	`simplesine_compensate()`
`simplesine_copy.c`	the `simplesine_copyXXX()` functions
`simplesine_deriv.c`	`simplesine_deriv()`
`simplesine_integ.c`	`simplesine_integ()`
`simplesine_propagate.c`	`simplesine_propagate()`
`simplesine_InteractionHandler.cpp`	the `TrickHLA` interaction handler class
`simplesine_LagCompensator.cpp`	the `TrickHLA` lag compensator class
`simplesine_Packing.cpp`	the `TrickHLA` packing/unpacking class

3.4.2.3 Data Files

This `simplesine/data` directory consists Trick input files for default `simplesine` data.

Depending on how you build your `S_define` file, these files may be used as "fallback" initializations for your data. In the simulations that follow, the actual initial values will often override the defaults in these files.[8]

The files are shown below.

filename	*description*
`integ.d`	Default numerical integration parameters.
`simplesine.d`	Default `simplesine` parameters with uninitialized state.
`simplesine_params.d`	Default `simplesine` parameters.

3.5 Federation Object Model

In the chapters that follow, various object instances and interactions are used. Some of the simulations exchange data by putting `simplesine` state and parameters in a class instance. Others exchange data by putting the parameters into an interaction.

This section shows the FOM snippets that define the relevant HLA class and interaction.

3.5.1 FOM structure

The Pitch FOM is an XML file that has a structure shown in Listing 3.1. For each object class used by the simulation, there must be a corresponding ⟨`objectClass`⟩ element containing as many ⟨`attribute`⟩ subelements as it has attributes. Similarly for each interaction class used by the simulation, there must be a corresponding ⟨`interactionClass`⟩ element containing as many ⟨`parameter`⟩ subelements as it has parameters.[9]

```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <!DOCTYPE objectModel SYSTEM "hla.dtd">
3 <objectModel ...>
```

[8] Sine the `simplesine` state data are output-only, they cannot be set directly from the Trick input processor. Consequently, there is no default data file for `simplesine_state_T`.

[9] A full FOM includes other elements not shown in the listing and also declares object and interaction classes used by the underlying HLA infrastructure and not by the federates themselves. This document does not address how to build a FOM file from scratch.

```
5   <!-- Declaration of all object classes known to the federation -->
6   <objects>
7     <objectClass name="..." ...>
8       <attribute name="..." ... />
9       ...
10    </objectClass>
11    ...
12  </objects>

14  <!-- Declaration of all interactions classes known to the federation -->
15  <interactions>
16    <interactionClass name="..." ...>
17      <parameter dataType="..." name="..." />
18      ...
19    </interactionClass>
20  </interactions>

22  ...
```

Listing 3.1: FOM structure

3.5.2 Object class declaration

The XML definition of a class consists of a single XML element, ⟨ `objectClass` ⟩ containing one child element ⟨ `attribute` ⟩ for each attribute.

The class used in some of the following simulations is named `SimplesineStateAndParameters` and is defined by the following XML snippet. Lines 2-7 declare one attribute for each of the current time, the state and parameter values (t, x, $\dot{x}$, A, ϕ and ω).

```
1  <objectClass name="SimplesineStateAndParameters" sharing="Neither">
2    <attribute dimensions="NA" name="Time" order="TimeStamp" transportation="HLAreliable"/>
3    <attribute dimensions="NA" name="Value" order="TimeStamp" transportation="HLAreliable"/>
4    <attribute dimensions="NA" name="dvdt" order="TimeStamp" transportation="HLAreliable"/>
5    <attribute dimensions="NA" name="Phase" order="TimeStamp" transportation="HLAreliable"/>
6    <attribute dimensions="NA" name="Frequency" order="TimeStamp" transportation="HLAreliable"/>
7    <attribute dimensions="NA" name="Amplitude" order="TimeStamp" transportation="HLAreliable"/>
8  </objectClass>
```

Listing 3.2: FOM snippet defining a class

3.5.3 Interaction class declaration

Similarly, the interactions used in some of the following simulations is named `SimplesineParameters` and is defined in the following XML snippet. Lines 3-5 declare one parameter for each of the state and parameter values (A, ϕ and ω).

```
1  <interactionClass dimensions="NA" name="SimplesineParameters"
2                    order="TimeStamp" sharing="Neither" transportation="HLAreliable">
3    <parameter dataType="HLAfloat64LE" name="A"/>
4    <parameter dataType="HLAfloat64LE" name="w"/>
5    <parameter dataType="HLAfloat64LE" name="phi"/>
6  </interactionClass>
```

Listing 3.3: FOM snippet defining a interaction

3.5.4 Simulation Configuration declaration

The following XML snippet defines the HLA class used to capture `TrickHLA` simulation configuration information. `TrickHLA` requires that a simulation configuration class be defined in the FOM, and the snippet below shows the class definition used by DSES simulations. (This document does not address how to design simulations which use a different simulation configuration class.)

```
<objectClass name="SimulationConfiguration" sharing="PublishSubscribe">
  <attribute dataType="HLAlogicalTime" dimensions="NA" name="run_duration" order="Receive"
             semantics="Duration␣of␣run" sharing="PublishSubscribe"
             transportation="HLAreliable" updateType="Static"/>
  <attribute dataType="HLAinteger32LE" dimensions="NA" name="number_of_federates" order="Receive"
             semantics="Number␣of␣required␣federates␣for␣run" sharing="PublishSubscribe"
             transportation="HLAreliable" updateType="Static"/>
  <attribute dataType="HLAinteger32LE" dimensions="NA" name="start_year" order="Receive"
             semantics="Year␣at␣start␣of␣run" sharing="PublishSubscribe"
             transportation="HLAreliable" updateType="Static"/>
  <attribute dataType="HLAinteger32LE" dimensions="NA" name="start_seconds" order="Receive"
             semantics="Starting␣time␣of␣run␣in␣seconds-of-year" sharing="PublishSubscribe"
             transportation="HLAreliable" updateType="Static"/>
  <attribute dataType="HLAunicodeString" dimensions="NA" name="owner" order="Receive"
             semantics="Federation␣publishing␣object" sharing="PublishSubscribe"
             transportation="HLAreliable"/>
  <attribute dataType="HLAunicodeString" dimensions="NA" name="scenario" order="Receive"
             semantics="Scenario␣being␣simulated." sharing="PublishSubscribe"
             transportation="HLAreliable"/>
  <attribute dataType="HLAunicodeString" dimensions="NA" name="mode" order="Receive"
             semantics="Mode␣of␣simulation␣run." sharing="PublishSubscribe"
             transportation="HLAreliable"/>
  <attribute dataType="HLAunicodeString" dimensions="NA" name="required_federates" order="Receive"
             semantics="Comma-separated␣list␣of␣required␣federates." sharing="PublishSubscribe"
             transportation="HLAreliable"/>
</objectClass>
```

Listing 3.4: FOM snippet defining a interaction

Chapter 4

An Example simplesine Simulation

In the following chapters, we use the `simplesine` model extensively as part of simulations to illustrate `TrickHLA` in action. In this chapter, we introduce a non-HLA `simplesine` simulation as a way to explain the basics of how the model may be used with Trick, and we do so without any `TrickHLA` distractions.

4.1 SIM_simplesine_pubsub

In this simulation, there are two `sim_objects`: a *publisher* and a *subscriber*.[1] The publisher generates a sine wave using the analytic equations and periodically copies the state to the subscriber. The subscriber propagates the state approximately between updates from the publisher. Based on the analytic equations, the subscriber also calculates an error in the approximate propagation.

The motivation for having the publisher and subscriber propagate the state differently is to illustrate a technique that is useful for HLA simulations. The owner of some data might simulate it at a very high rate but send updates at a lower frequency. In between these updates, a subscriber may use an approximate method to extrapolate the data until the next update arrives.[2]

4.2 S_define

The `SIM_simplesine_pubsub` `S_define` file is shown below. It consists of

- Some `#define` statements that set relevant frequencies for state propagation and data copying.
- The `sim_objects` – the usual Trick `sys` object and publisher and subscriber objects.

[1] This publisher/subscriber terminology here is only suggestive. There is no distributed computing going on. Data only moves from one `sim_object` to another within a single process. Nevertheless, the simulation is a fair way to introduce how `simplesine` data structure and functions are used in Trick `S_define` and input files.

[2] Extrapolation alternatives include doing nothing, in which case the data increments in discontinuous steps as the remote data arrive; dead reckoning, in which the data are extrapolated based on derivatives; or numerical integration of an approximate (lower fidelity) model, in which case the discontinuities on the subscriber side are hopefully reduced sufficiently to allow the subscriber's simulation to proceed.

- An `integrate` statement which enables Trick numerical integration for the subscriber.

```
#define PROPAGATE_TIMESTEP 0.25
#define COPY_TIMESTEP 5.0

sim_object
{
  sim_services/include: EXECUTIVE exec (sim_services/include/executive.d) ;

  (automatic) sim_services/input_processor:
    input_processor( INPUT_PROCESSOR* IP = &sys.exec.ip ) ;
} sys ;


sim_object
{
  simplesine: simplesine_T simplesine (simplesine/data/simplesine.d);

  (initialization) simplesine:
    simplesine_calc(
      simplesine_T* P = &publisher.simplesine,
      double t = sys.exec.out.time );

  // Propagate the state using the analytic equations.
  (PROPAGATE_TIMESTEP, scheduled) simplesine:
    simplesine_calc(
      simplesine_T* P = &publisher.simplesine,
      double t = sys.exec.out.time );

  // Copy the propagated state to the subscriber.
  (COPY_TIMESTEP, scheduled) simplesine:
    simplesine_copyState(
      simplesine_state_T* fromP = &publisher.simplesine.state,
      simplesine_state_T* toP = &subscriber.simplesine.state );
} publisher ;


sim_object
{
  sim_services/include: INTEGRATOR integ (simplesine/data/integ.d);
  simplesine: simplesine_T simplesine (simplesine/data/simplesine.d);
  simplesine: simplesine_T err (simplesine/data/simplesine.d);

  (initialization) simplesine:
    simplesine_calc(
      simplesine_T* P = &subscriber.simplesine,
      double t = sys.exec.out.time );

  // deriv/integ jobs to propagate the state differential equation
  (derivative) simplesine:
    simplesine_deriv( simplesine_T* p = &subscriber.simplesine );
  (integration) simplesine:
    simplesine_integ(
      INTEGRATOR* I = &subscriber.integ,
      simplesine_T* p = &subscriber.simplesine );

  // calculate the error between the integrated state and the true state
  (PROPAGATE_TIMESTEP, scheduled) simplesine:
    simplesine_calcError(
      double t = sys.exec.out.time,
      simplesine_T* s = &subscriber.simplesine,
      simplesine_state_T* err = &subscriber.err.state );
} subscriber ;

integrate (PROPAGATE_TIMESTEP) subscriber;
```

Listing 4.1: `SIM_simplesine_pubsub` `S_define` file

4.3 Input Files

The input files for this simulation are located in the `RUN_1` directory and are summarized below.

- `input_noCopy_noInteg`. In this input file, the publisher never sends data to the subscriber, and the subscriber does not propagate its local state. The motivation here is to illustrate that nothing really happens on the subscriber side until the publisher sends data.
- `input_noInteg`. This input file illustrates the arrival of data at the subscriber from the publisher. By not propagating the subscriber state, the discrete arrival of updates is readily evident. In some simulations in the subsequent chapter, we will use this technique to illustrate the arrival of HLA data.
- `input`. This file illustrates the simulation running as it is intended: the publisher sends data periodically to the subscriber, and the subscriber integrates those data in between updates. In this case the harmonical oscillator is so simple that the numerical integration is a very good approximation to the true system.

The first two files illustrate how to disable specific Trick jobs from the input file using the `JOB` directive. In `input_noCopy_noInteg`, the publisher's publisher-to-subscriber copy job is disabled as well as the subscriber's numerical integration jobs. In `input_noInteg`, the subscriber's numerical integration jobs are disabled. The input files are shown below.

```
1  #include "S_default.dat"
2  #include "Log_data/states.d"
3  #include "Modified_data/realtime.d"
4  #include "Modified_data/publisher.d"
5  #include "Modified_data/subscriber.d"

7  JOB publisher.simplesine_copyState(&publisher.simplesine) = Off;

9  JOB subscriber.simplesine_deriv(&subscriber.simplesine) = Off;
10 JOB subscriber.simplesine_integ(&subscriber.integ) = Off;

12 stop = 32.5;
```

Listing 4.2: `SIM_simplesine_pubsub` input file, `input_noCopy_noInteg`

```
1  #include "S_default.dat"
2  #include "Log_data/states.d"
3  #include "Modified_data/realtime.d"
4  #include "Modified_data/publisher.d"
5  #include "Modified_data/subscriber.d"

7  JOB publisher.simplesine_copyState(&publisher.simplesine) = Off;

9  stop = 32.5;
```

Listing 4.3: `SIM_simplesine_pubsub` input file, `input_noInteg`

```
#include "S_default.dat"
#include "Log_data/states.d"
#include "Modified_data/realtime.d"
#include "Modified_data/publisher.d"
#include "Modified_data/subscriber.d"

stop = 32.5;
```

Listing 4.4: `SIM_simplesine_pubsub` input file, `input`

4.4 Output

Output from the simulation with `input_noCopy_noInteg` is shown in Figure 4.1. The plot shows the evolution of the sine wave for approximately ten cycles. The publisher state ($x(t)$ and $\dot{x}(t)$) evolve as expected. The subscriber state is flatlined at its initial conditions, since data never arrive from the publisher and the subscriber's numerical propagation is disabled.

Output from the simulation with `input_noInteg` is shown in Figure 4.2. It clearly shows the discrete transfer of data from the publisher to the subscriber. Between the data updates, the subscriber state remains constant, since there is still no numerical intregration on the subscriber side. This manifests itself in errors which grow significantly until the next update arrives, at which point the errors reset to zero.

Output from the simulation with `input` is shown in Figure 4.3. In this case, the subscriber state is "smoothed" in between data updates, since the subscriber numerical integration has been enabled. Note that there are still errors which grow between updates; however, in this case the magnitude of those error has diminished by several orders of magnitude.

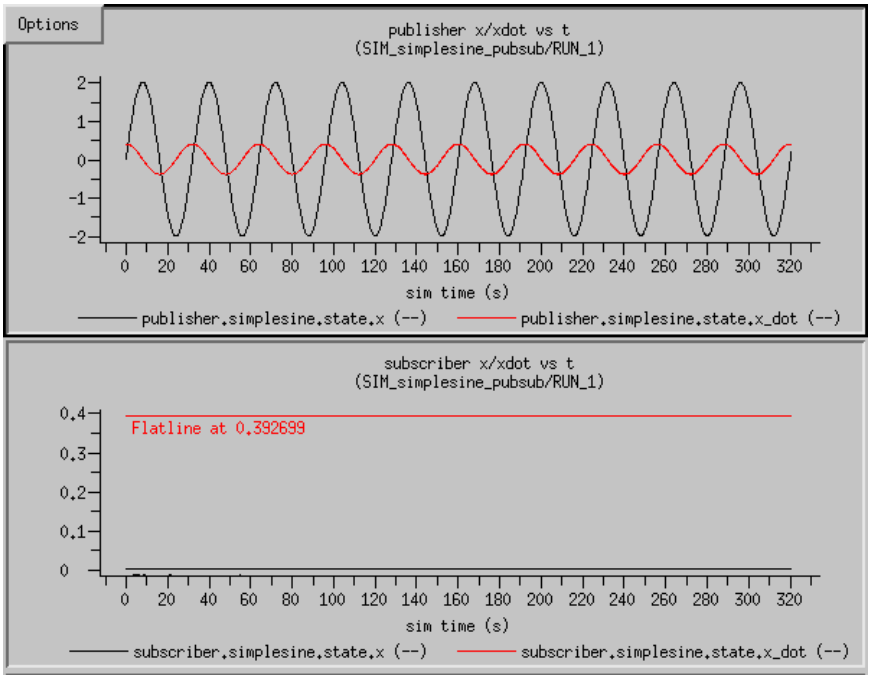

Figure 4.1: Output from `SIM_simplesine_pubsub` using input file `input_noCopy_noInteg`

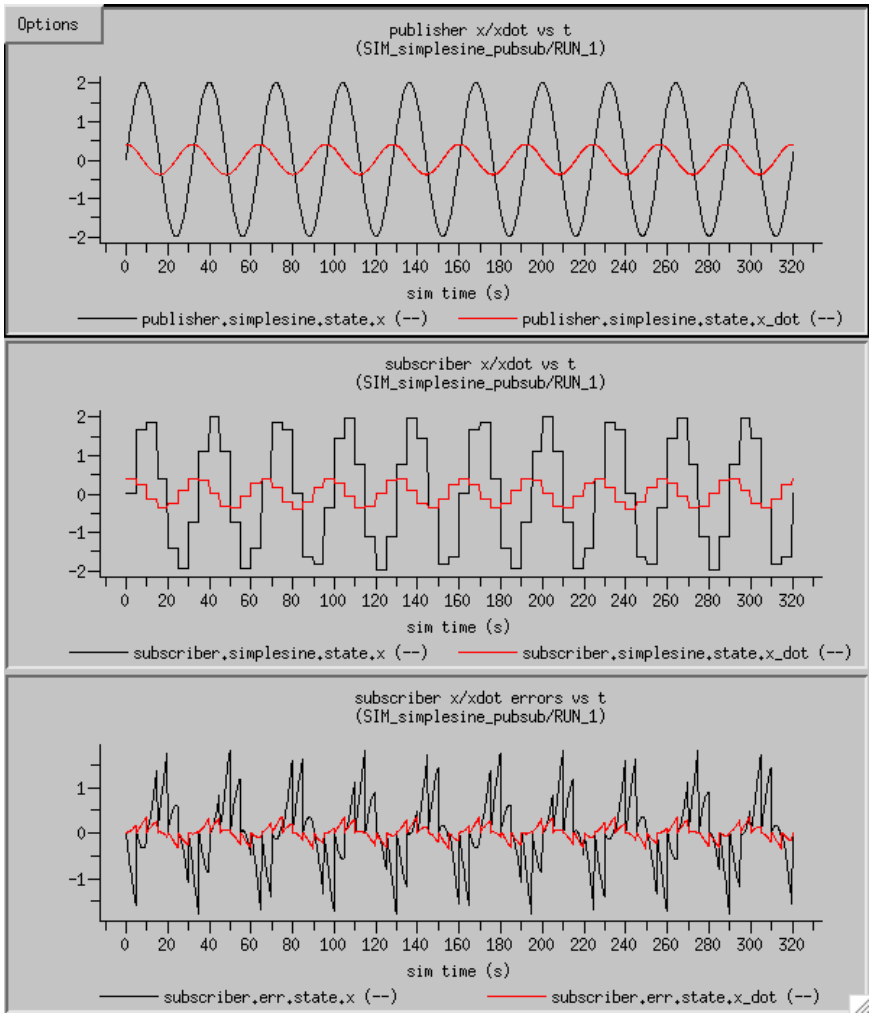

Figure 4.2: Output from `SIM_simplesine_pubsub` using input file `input_noInteg`

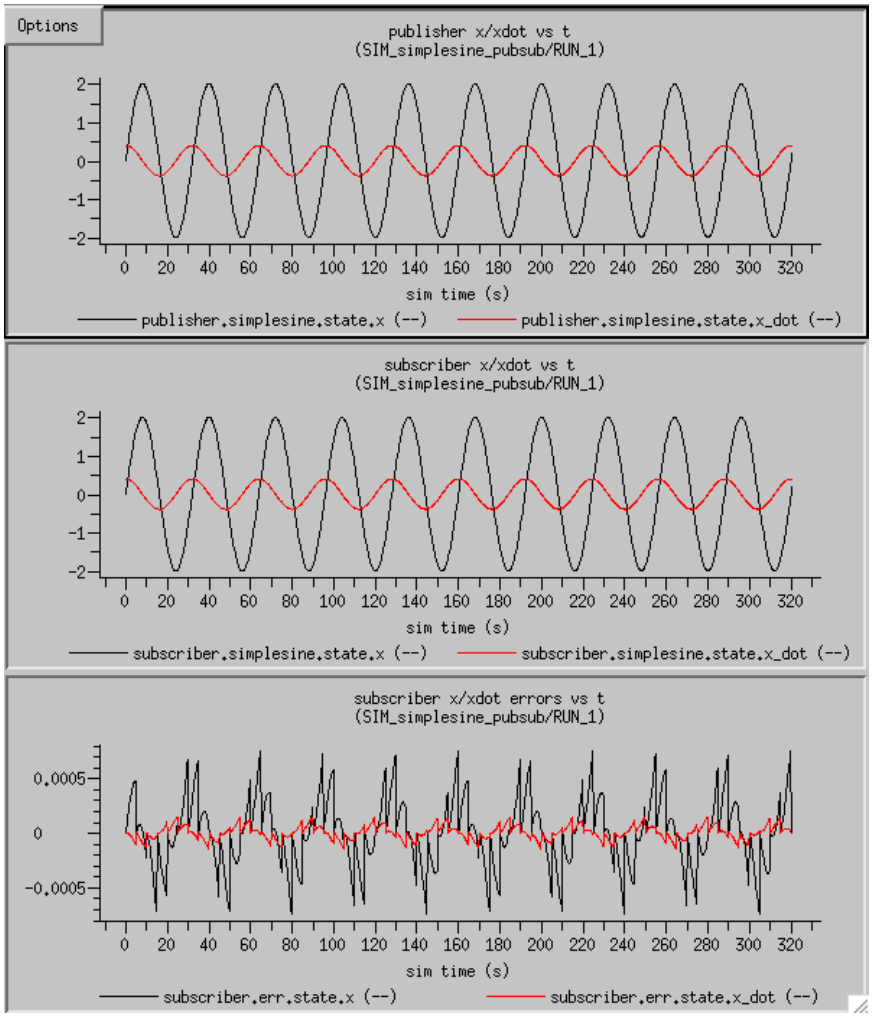

Figure 4.3: Output from `SIM_simplesine_pubsub` using input file `input`

Chapter 5

Joining a Federation

In this chapter, we present a Trick simulation which uses `TrickHLA` to join an HLA federation.[1] If the specified federation exists, the simulation joins it. If the federation does not exist, `TrickHLA` takes care of creating it.

The simulation is virtually identical to `SIM_simplesine_pubsub` presented in Section 3.4, except that this simulation joins (or creates) an HLA federation. In particular, there is no distributed publish/subscribe; the publisher and subscriber are both resident in this single simulation process, and they exchange data to each other locally.

The main tasks to HLA-enable an existing Trick simulation are to insert some "standard" `TrickHLA` `sim_objects` into the `S_define` file, and configure the input file with new `TrickHLA` parameters. The following sections illustrate what is involved.

5.1 SIM_simplesine_hla_join

The first step in enabling `TrickHLA`, is inserting two new `sim_objects` into the `S_define` file: one which contains the main `TrickHLA` execution framework (a bunch of jobs that automate the `TrickHLA` process) and one which contains object and jobs related to simulation configuration and initialization. The first of these, `THLA`, is shown below and can be pasted in verbatim.[2] In most cases, very few modifications to this `sim_object` should be necessary.

Examination of the code reveals that `TrickHLA` is composed of three objects: the *mangager*, the *federate*, and the *federate ambassador*. The `TrickHLA` infrastructure is the base manager and federate jobs that follow. In most cases, it is possible to view these objects and their jobs as black boxes: just paste the object into the `S_define` file.

```
1 sim_object {
2    TrickHLA: TrickHLAFreezeInteractionHandler freeze_ih;
3    TrickHLA: TrickHLAFedAmb federate_amb;
4    TrickHLA: TrickHLAFederate federate;
```

[1] The `TrickHLA` initialization process is more complex than this simulation illustrates. The only focus here is on using `TrickHLA` to enable HLA in a Trick simulation. The full `TrickHLA` *multi-phase initialzation process* is discussed in Section 11 on page 66.

[2] Some comments have been removed from this object to reduce page space.

```
   TrickHLA: TrickHLAManager manager;
   double checkpoint_time;
   char  checkpoint_label[256];

   // Initialization jobs
   P1 (initialization) TrickHLA: THLA.manager.print_version();
   P1 (initialization) TrickHLA: THLA.federate.fix_FPU_control_word();
   P60 (initialization) TrickHLA: THLA.federate_amb.initialize(
      In TrickHLAFederate * federate = &THLA.federate,
      In TrickHLAManager * manager = &THLA.manager );
   P60 (initialization) TrickHLA: THLA.federate.initialize(
      Inout TrickHLAFedAmb * federate_amb = &THLA.federate_amb );
   P60 (initialization) TrickHLA: THLA.manager.initialize(
      In TrickHLAFederate * federate = &THLA.federate );
   P65534 (initialization) TrickHLA: THLA.manager.initialization_complete();
   P65534 (initialization) TrickHLA: THLA.federate.check_pause_at_init(
      In const double check_pause_delta = THLA_CHECK_PAUSE_DELTA );

   // Checkpoint related jobs
   P1 (checkpoint)       TrickHLA: THLA.federate.setup_checkpoint();
   (freeze)              TrickHLA: THLA.federate.perform_checkpoint();
   P1 (pre_load_checkpoint) TrickHLA: THLA.federate.setup_restore();
   (freeze)              TrickHLA: THLA.federate.perform_restore();

   // Freeze jobs
   (freeze)  TrickHLA: THLA.federate.check_freeze();
   (unfreeze) TrickHLA: THLA.federate.exit_freeze();

   // Scheduled jobs
   P1 (THLA_DATA_CYCLE_TIME, environment) TrickHLA: THLA.federate.wait_for_time_advance_grant();
   P1 (THLA_INTERACTION_CYCLE_TIME, environment) TrickHLA: THLA.manager.process_interactions();
   P1 (THLA_DATA_CYCLE_TIME, environment) TrickHLA: THLA.manager.process_deleted_objects();
   P1 (0.0, environment) TrickHLA: THLA.manager.start_federation_save(
      In const char * file_name = THLA.checkpoint_label );
   P1 (0.0, environment) TrickHLA: THLA.manager.start_federation_save_at_sim_time(
      In double freeze_sim_time = THLA.checkpoint_time,
      In const char * file_name = THLA.checkpoint_label );
   P1 (0.0, environment) TrickHLA: THLA.manager.start_federation_save_at_scenario_time(
      In double freeze_scenario_time = THLA.checkpoint_time,
      In const char * file_name = THLA.checkpoint_label );
   P1 (THLA_DATA_CYCLE_TIME, environment) TrickHLA: THLA.manager.receive_cyclic_data(
      In double current_time = sys.exec.out.time );
   P65534 (THLA_DATA_CYCLE_TIME, logging) TrickHLA: THLA.manager.send_cyclic_data(
      In double current_time = sys.exec.out.time );
   P65534 (THLA_DATA_CYCLE_TIME, logging) TrickHLA: THLA.manager.send_requested_data(
      In double current_time = sys.exec.out.time );
   P65534 (THLA_DATA_CYCLE_TIME, logging) TrickHLA: THLA.manager.process_ownership();
   P65534 (THLA_DATA_CYCLE_TIME, logging) TrickHLA: THLA.federate.time_advance_request();
   P65534 (THLA_DATA_CYCLE_TIME, logging) TrickHLA: THLA.federate.check_freeze_time();
   P65534 (THLA_DATA_CYCLE_TIME, THLA_CHECK_PAUSE_JOB_OFFSET, logging) TrickHLA: THLA.federate.check_pause(
      In const double check_pause_delta = THLA_CHECK_PAUSE_DELTA );
   P65534 (THLA_DATA_CYCLE_TIME, THLA_CHECK_PAUSE_JOB_OFFSET, logging) TrickHLA: THLA.federate.enter_freeze();

   // Shutdown jobs
   P65534 (shutdown) TrickHLA: THLA.manager.shutdown();
} THLA;
```

Listing 5.1: The `THLA sim_object`

In addition, the following `THLA_INIT` object should be put into the `S_define` file. The version shown here is sufficient for now.[3]

[3] We will discuss this object in more detail when we address `TrickHLA` multi-phase initialization.

```
sim_object {
   // The DSES simulation configuration.
   simconfig: DSESSimConfig dses_config;

   // Generally, initialization jobs will go here, but not for this example.

   // Clear remaining initialization sync-points.
   P100 (initialization) TrickHLA: THLA.manager.clear_init_sync_points();
} THLA_INIT;
```

Listing 5.2: The `THLA_INIT sim_object`

5.2 Input Files

The second step in enabling `TrickHLA` is to set its parameters in a Trick input file. The input file for `SIM_simplesine_hla_join` is shown below.

```
#include "S_properties"
#include "S_default.dat"
#include "Log_data/states.d"
#include "Modified_data/realtime.d"
#include "Modified_data/publisher.d"
#include "Modified_data/subscriber.d"

stop =32.5;

//
// Basic RTI/federation connection info
//

// Configure the CRC for the Pitch RTI.
THLA.federate.local_settings = "crcHost␣=␣localhost\n␣crcPort␣=␣8989";

THLA.federate.name         = "pubsub_join";
THLA.federate.FOM_modules  = "FOM.xml";
THLA.federate.federation_name = "simplesine";

THLA.federate.lookahead_time = THLA_DATA_CYCLE_TIME;
THLA.federate.time_regulating = true;
THLA.federate.time_constrained = true;
THLA.federate.multiphase_init_sync_points = "Phase1,␣Phase2";

THLA.federate.enable_known_feds  = true;
THLA.federate.known_feds_count   = 1;
THLA.federate.known_feds         = alloc(THLA.federate.known_feds_count);
THLA.federate.known_feds[0].name = "pubsub_join";
THLA.federate.known_feds[0].required = true;

// TrickHLA debug messages.
THLA.manager.debug_handler.debug_level = THLA_LEVEL2_TRACE;

//
// SimConfig initialization
//

// DSES simulation configuration.
THLA_INIT.dses_config.owner          = "pubsub_join";
THLA_INIT.dses_config.run_duration   = 15.0;
THLA_INIT.dses_config.num_federates  = 1;
THLA_INIT.dses_config.required_federates = "pubsub_join";
THLA_INIT.dses_config.start_year     = 2007;
```

```
45  THLA_INIT.dses_config.start_seconds  = 0;
46  THLA_INIT.dses_config.scenario       = "Nominal";
47  THLA_INIT.dses_config.mode           = "Unknown";

49  // Simulation Configuration for DSES Multi-phase Initialization.
50  THLA.manager.sim_config.FOM_name     = "SimulationConfiguration";
51  THLA.manager.sim_config.name         = "SimConfig";
52  THLA.manager.sim_config.packing      = &THLA_INIT.dses_config;
53  THLA.manager.sim_config.attr_count   = 8;
54  THLA.manager.sim_config.attributes   = alloc(THLA.manager.sim_config.attr_count);

56  THLA.manager.sim_config.attributes[0].FOM_name = "owner";
57  THLA.manager.sim_config.attributes[0].trick_name = "THLA_INIT.dses_config.owner";
58  THLA.manager.sim_config.attributes[0].publish = true;
59  THLA.manager.sim_config.attributes[0].subscribe = true;
60  THLA.manager.sim_config.attributes[0].rti_encoding = THLA_UNICODE_STRING;

62  THLA.manager.sim_config.attributes[1].FOM_name = "run_duration";
63  THLA.manager.sim_config.attributes[1].trick_name = "THLA_INIT.dses_config.run_duration_microsec";
64  THLA.manager.sim_config.attributes[1].publish = true;
65  THLA.manager.sim_config.attributes[1].subscribe = true;
66  THLA.manager.sim_config.attributes[1].rti_encoding = THLA_LITTLE_ENDIAN;

68  THLA.manager.sim_config.attributes[2].FOM_name = "number_of_federates";
69  THLA.manager.sim_config.attributes[2].trick_name = "THLA_INIT.dses_config.num_federates";
70  THLA.manager.sim_config.attributes[2].publish = true;
71  THLA.manager.sim_config.attributes[2].subscribe = true;
72  THLA.manager.sim_config.attributes[2].rti_encoding = THLA_LITTLE_ENDIAN;

74  THLA.manager.sim_config.attributes[3].FOM_name = "required_federates";
75  THLA.manager.sim_config.attributes[3].trick_name = "THLA_INIT.dses_config.required_federates";
76  THLA.manager.sim_config.attributes[3].publish = true;
77  THLA.manager.sim_config.attributes[3].subscribe = true;
78  THLA.manager.sim_config.attributes[3].rti_encoding = THLA_UNICODE_STRING;

80  THLA.manager.sim_config.attributes[4].FOM_name = "start_year";
81  THLA.manager.sim_config.attributes[4].trick_name = "THLA_INIT.dses_config.start_year";
82  THLA.manager.sim_config.attributes[4].publish = true;
83  THLA.manager.sim_config.attributes[4].subscribe = true;
84  THLA.manager.sim_config.attributes[4].rti_encoding = THLA_LITTLE_ENDIAN;

86  THLA.manager.sim_config.attributes[5].FOM_name = "start_seconds";
87  THLA.manager.sim_config.attributes[5].trick_name = "THLA_INIT.dses_config.start_seconds";
88  THLA.manager.sim_config.attributes[5].publish = true;
89  THLA.manager.sim_config.attributes[5].subscribe = true;
90  THLA.manager.sim_config.attributes[5].rti_encoding = THLA_LITTLE_ENDIAN;

92  THLA.manager.sim_config.attributes[6].FOM_name = "scenario";
93  THLA.manager.sim_config.attributes[6].trick_name = "THLA_INIT.dses_config.scenario";
94  THLA.manager.sim_config.attributes[6].publish = true;
95  THLA.manager.sim_config.attributes[6].subscribe = true;
96  THLA.manager.sim_config.attributes[6].rti_encoding = THLA_UNICODE_STRING;

98  THLA.manager.sim_config.attributes[7].FOM_name = "mode";
99  THLA.manager.sim_config.attributes[7].trick_name = "THLA_INIT.dses_config.mode";
100 THLA.manager.sim_config.attributes[7].publish = true;
101 THLA.manager.sim_config.attributes[7].subscribe = true;
102 THLA.manager.sim_config.attributes[7].rti_encoding = THLA_UNICODE_STRING;

104 //
105 // Object info
106 //
107 THLA.manager.obj_count = 0;

109 //
110 // Interaction info
```

```
111 //
112 THLA.manager.inter_count = 0;
```

Listing 5.3: `SIM_simplesine_hla_join` input file

Lines 2-8 are identical to the input file for `SIM_simplesine_pubsub`. Everything else is new.

Lines 16-18 initialize the federation, supplying values for this federate's name, the name of the federation to join, the host/port information for the HLA RTI,[4] and the name of the FOM file (located in the same directory as the `S_define` file).

Lines 20-22 are related to HLA time management: the *lookahead time*, and two flags indicating whether the simulation is *time regulating* and *time constrained*. All the examples in this document use a lookahead of just under 1sec and are both time constrained and time regulating.

Lines 24-28 itemizes a list of *known federates*. This is the `TrickHLA` mechanism for ensuring that federates wait for everyone to join before proceeding. In this example, there is only a single federate (`pubsub_join`), but in cases where there are several, the array would be allocated to hold them all, specifying the name of each and whether they are required to be present before the others may proceed.

Line 31 is setting the global debug level flags and lead to gradually more or less verbose output.

Lines 38-45 initialize the federation's `SimulationConfiguration` object. There is one (and only one) of these instances for all the federates in the federation, but each one (if it uses `TrickHLA`) will nevertheless set these values. The `TrickHLA` infrastructure ensures that even though many federates attempt to publish the object, it only gets created by one of them. One point worth noting: the value on line 39 for the `.run_duration` parameter ensures that the simulation does not run longer than the specified duration, even if that duration is less than the value specified with the `STOP` directive (on line 7).

Lines 48-101 are also related to the `SimulationConfiguration` object as well as multi-phase initialization. (We do not discuss multi-phase initialization in this example.)

Finally, lines 106 and 111 specify that this simulation has no objects to publish or subscribe and no interactions to send or receive. Subsequent examples will elaborate on the `TrickHLA` publish/-subscribe mechanisms.

5.3 Output

The relevant output for this simulation is the output stream from the running simulation, which (among other things) verifies that the simulation did indeed create and join an HLA federation. An abbreviated version of the output is shown below.

```
1 ...
2 | |wormhole|1|0.00|2007/07/25,17:59:09| TrickHLAFedAmb::initialize()
3 | |wormhole|1|0.00|2007/07/25,17:59:09| TrickHLAFederate::initialize()
4 | |wormhole|1|0.00|2007/07/25,17:59:09| TrickHLAManager::initialize()
5 ...
```

[4] The standard port number is 8989. During development the host is often set to localhost, but in general the RTI will be running on some agreed location on the network.

```
| |wormhole|1|0.00|2007/07/25,17:59:10|
 TRIVIAL: Trick Federation "simplesine": CREATING FEDERATION EXECUTION
| |wormhole|1|0.00|2007/07/25,17:59:10|
 ADVISORY: Trick Federation "simplesine": SUCCESSFULLY CREATED FEDERATION EXECUTION
| |wormhole|1|0.00|2007/07/25,17:59:10|
 TRIVIAL: Trick Federation "simplesine": JOINING FEDERATION EXECUTION
| |wormhole|1|0.00|2007/07/25,17:59:10|
 ADVISORY: Trick Federation "simplesine": JOINED FEDERATION EXECUTION
Federate Handle = 2
...
| |wormhole|1|0.00|2007/07/25,17:59:10| TrickHLAFederate::wait_for_required_federates_to_join()
WAITING FOR 1 REQUIRED FEDERATES:
    1: Waiting for Federate 'pubsub_join'
...
| |wormhole|1|0.00|2007/07/25,17:59:10| TrickHLAFederate::wait_for_required_federates_to_join()
WAITING FOR 1 REQUIRED FEDERATES:
    1: Found required Federate 'pubsub_join'
...
| |wormhole|1|0.00|2007/07/25,17:59:10| TrickHLAManager::initialization_complete()
       Simulation has started and is now running...
...
Federate "pubsub_join" Time granted to: 1
...
Federate "pubsub_join" Time granted to: 16
...
 TRIVIAL: Trick Federation "simplesine": RESIGNING FROM FEDERATION
...
 ADVISORY: Trick Federation "simplesine": RESIGNED FROM FEDERATION
...
Federation destroyed
...
SIMULATION TERMINATED IN
  PROCESS: 1
  JOB/ROUTINE: 11/sim_services/mains/master.c
DIAGNOSTIC:
Simulation reached input termination time.

LAST JOB CALLED: THLA.THLA.federate.time_advance_request()
             TOTAL OVERRUNS:          0
PERCENTAGE REALTIME OVERRUNS:     0.000%


      SIMULATION START TIME:      0.000
       SIMULATION STOP TIME:     15.000
    SIMULATION ELAPSED TIME:     15.000
        ACTUAL ELAPSED TIME:     15.000
       ACTUAL CPU TIME USED:      0.040
   SIMULATION / ACTUAL TIME:      1.000
      SIMULATION / CPU TIME:    375.056
  ACTUAL INITIALIZATION TIME:      0.000
     INITIALIZATION CPU TIME:      0.719
*** DYNAMIC MEMORY USAGE ***
     CURRENT ALLOCATION SIZE: 1569508
       NUM OF CURRENT ALLOCS:   483
         MAX ALLOCATION SIZE: 1569508
           MAX NUM OF ALLOCS:   483
       TOTAL ALLOCATION SIZE: 1794096
         TOTAL NUM OF ALLOCS:  3301
```

Listing 5.4: `SIM_simplesine_hla_join` output

Lines 1-14 show the simulation running through its startup process, initializing the `TrickHLA` objects which in turn create and join the HLA federation names *simplesine*.

Lines 16-25 show the simulation waiting for all the required federates to join, which in this case is

just this simulation.

Lines 27-29 show the beginning and end of the time grants issues by HLA, starting at $t = 1$ and going thru the end of the simulation duration, as t extended past the specified limit of 15.

Lines 31-35 show the simulation resigning from the federation and destroying it (since no federates remain).

The remaining lines are standard Trick output generated at the end of a run. Note that in spite of the fact that the input file specified `stop = 32.5` as shown on line 7 of the input file (Listing 5.3 on page 23), the simulation actually terminated earlier due to the simulation configuration `.run_duration = 15.0` as specified in the input file.

Chapter 6

Publishing and Subscribing

In this chapter, we present two Trick simulations: one that publishes sine wave data via HLA and one that subscribes. Together, these represent a distributed, HLA version of the simple publish/-subscribe simulation that was discussed in Section 4.2 on page 14.

6.1 SIM_simplesine_hla_pub

This section discusses the `S_define` file for a publisher. The file is very similar to that discussed in Section 5.1 on page 21, except that there is no subscriber: the subscriber `sim_object` is gone, the publisher no longer has a job to copy data to the subscriber, and there is no Trick `integrate` directive for the subscriber's numerical integraion.

An abbreviated version file is shown below. (The full `THLA` and `THLA_INIT` sim objects are identical to those in the `SIM_simplesine_hla_join` file.)

```
1  #include "S_properties"
2
3  #define PROPAGATE_TIMESTEP 0.25
4
5  sim_object
6  {
7    sim_services/include: EXECUTIVE exec (sim_services/include/executive.d) ;
8
9    (automatic) sim_services/input_processor:
10     input_processor( INPUT_PROCESSOR* IP = &sys.exec.ip ) ;
11 } sys ;
12
13 sim_object
14 {
15   simplesine: simplesine_T simplesine (simplesine/data/simplesine.d);
16
17   (initialization) simplesine:
18     simplesine_calc(
19       simplesine_T* P = &publisher.simplesine,
20       double t = sys.exec.out.time );
21   (PROPAGATE_TIMESTEP, scheduled) simplesine:
22     simplesine_calc(
23       simplesine_T* P = &publisher.simplesine,
24       double t = sys.exec.out.time );
25 } publisher ;
```

```
27 #include "S_modules/THLA.sm"

29 sim_object {
30   ...
31 } THLA_INIT;
```

Listing 6.1: SIM_simplesine_hla_pub S_define

6.2 SIM_simplesine_hla_sub

This section discusses the S_define file for a TrickHLA publisher. Like the publisher above, this file is similar to the S_define file for the non-HLA publisher/subscriber in Section 4.2 on page 14, except that this file has no publisher object. An abbreviated version of the subscriber's S_define is shown below.

```
1  #include "S_properties"

3  #define PROPAGATE_TIMESTEP 0.25
4  #define COPY_TIMESTEP 5.0

6  sim_object
7  {
8    sim_services/include: EXECUTIVE exec (sim_services/include/executive.d) ;

10   (automatic) sim_services/input_processor:
11     input_processor( INPUT_PROCESSOR* IP = &sys.exec.ip ) ;
12 } sys ;


15 sim_object
16 {
17   sim_services/include: INTEGRATOR integ (simplesine/data/integ.d);
18   simplesine: simplesine_T simplesine;
19   simplesine: simplesine_T err;

21   (initialization) simplesine:
22     simplesine_calc(
23       simplesine_T* P = &subscriber.simplesine,
24       double t = sys.exec.out.time );

26   (derivative) simplesine:
27     simplesine_deriv( simplesine_T* p = &subscriber.simplesine );

29   (integration) simplesine:
30     simplesine_integ(
31       INTEGRATOR* I = &subscriber.integ,
32       simplesine_T* p = &subscriber.simplesine );

34   (PROPAGATE_TIMESTEP, scheduled) simplesine:
35     simplesine_calcError(
36       double t = sys.exec.out.time,
37       simplesine_T* s = &subscriber.simplesine,
38       simplesine_state_T* err = &subscriber.err.state );

40 } subscriber ;

42 integrate (PROPAGATE_TIMESTEP) subscriber;

44 #include "S_modules/THLA.sm"
```

```
sim_object {
  ...
} THLA_INIT;
```

Listing 6.2: SIM_simplesine_hla_sub S_define

6.3 Publisher input file

The publisher's input file is shown below. It is very similar to the input file used for the join example in the previous chapter. The differences are

- this federate has a different name (*publisher*),
- the simulation waits for the *subscriber* federate to join the federation, and
- one object class is defined to the data which this simulation publishes.

The last difference is the most significant. To add object classes to a simulation, all that is required is that you declare them in the input file.[1] For this simulation, the relevant additions to the input file are shown below.

```
// This federate has one object which is publishes.
THLA.manager.obj_count = 1;
THLA.manager.objects = alloc(THLA.manager.obj_count);

// Configure the object this federate owns and will publish.
THLA.manager.objects[0].FOM_name        = "SimplesineStateAndParameters";
THLA.manager.objects[0].name            = "simplesineStateAndParameters";
THLA.manager.objects[0].create_HLA_instance = true;
THLA.manager.objects[0].attr_count      = 6;
THLA.manager.objects[0].attributes      = alloc(THLA.manager.objects[0].attr_count);

THLA.manager.objects[0].attributes[0].FOM_name = "Time";
THLA.manager.objects[0].attributes[0].trick_name = "sys.exec.out.time";
THLA.manager.objects[0].attributes[0].config   = THLA_CYCLIC;
THLA.manager.objects[0].attributes[0].publish  = true;
THLA.manager.objects[0].attributes[0].locally_owned = true;
THLA.manager.objects[0].attributes[0].rti_encoding = THLA_LITTLE_ENDIAN;

THLA.manager.objects[0].attributes[1].FOM_name = "Value";
THLA.manager.objects[0].attributes[1].trick_name = "publisher.simplesine.state.x";
THLA.manager.objects[0].attributes[1].config   = THLA_INITIALIZE + THLA_CYCLIC;
THLA.manager.objects[0].attributes[1].publish  = true;
THLA.manager.objects[0].attributes[1].locally_owned = true;
THLA.manager.objects[0].attributes[1].rti_encoding = THLA_LITTLE_ENDIAN;

THLA.manager.objects[0].attributes[2].FOM_name = "dvdt";
THLA.manager.objects[0].attributes[2].trick_name = "publisher.simplesine.state.x_dot";
THLA.manager.objects[0].attributes[2].config   = THLA_CYCLIC;
THLA.manager.objects[0].attributes[2].publish  = true;
THLA.manager.objects[0].attributes[2].locally_owned = true;
THLA.manager.objects[0].attributes[2].rti_encoding = THLA_LITTLE_ENDIAN;

THLA.manager.objects[0].attributes[3].FOM_name = "Phase";
```

[1] Of course, the object class in question must already be declared in the FOM.

```
34 THLA.manager.objects[0].attributes[3].trick_name = "publisher.simplesine.params.phi";
35 THLA.manager.objects[0].attributes[3].config   = THLA_CYCLIC;
36 THLA.manager.objects[0].attributes[3].publish  = true;
37 THLA.manager.objects[0].attributes[3].locally_owned = true;
38 THLA.manager.objects[0].attributes[3].rti_encoding = THLA_LITTLE_ENDIAN;

40 THLA.manager.objects[0].attributes[4].FOM_name = "Frequency";
41 THLA.manager.objects[0].attributes[4].trick_name = "publisher.simplesine.params.w";
42 THLA.manager.objects[0].attributes[4].config   = THLA_CYCLIC;
43 THLA.manager.objects[0].attributes[4].publish  = true;
44 THLA.manager.objects[0].attributes[4].locally_owned = true;
45 THLA.manager.objects[0].attributes[4].rti_encoding = THLA_LITTLE_ENDIAN;

47 THLA.manager.objects[0].attributes[5].FOM_name = "Amplitude";
48 THLA.manager.objects[0].attributes[5].trick_name = "publisher.simplesine.params.A";
49 THLA.manager.objects[0].attributes[5].config   = THLA_CYCLIC;
50 THLA.manager.objects[0].attributes[5].publish  = true;
51 THLA.manager.objects[0].attributes[5].locally_owned = true;
52 THLA.manager.objects[0].attributes[5].rti_encoding = THLA_LITTLE_ENDIAN;
```

Listing 6.3: `SIM_simplesine_hla_pub` input file

This set of inputs tells `TrickHLA` that the simulation will be publishing an object with six attributes.

Lines 2-3 indicate that only a single object is involved.

Lines 6-7 specify that the object instance will be named `simplesineStateAndParameters` and that it is an instance of the class `SimplesineStateAndParameters` (which must be defined in the FOM).

Line 8 indicates that the instance will be owned by this simulation.

Lines 10-11 specify that the simulation is interested in six attributes of the instance – in this case all of them.

The subsequent lines specify per-attribute settings: the name of the attribute as specified in the class definition in the FOM, the name of the Trick variable associated with the attribute, whether the attribute is owned locally, whether this simulation intends to publish values for the attribute, and how the data is to be encoded when it is sent over the network. Notice that the trick variables specified are `simplesine` parameters and state located in the publisher object.

By associating Trick variables with attributes in this fashion, whenever the `TrickHLA` infrastructure sends out updates (as defined in the `THLA` object in the `S_define` file), the current value of the corresponding Trick variable will be used. Simulation developers need do nothing explicit to send data – the `TrickHLA` infrastructure handles that task.

6.4 Subscriber input file

Like the publisher, the subscriber's input file is similar to the join example in the previous chapter. The differences are

- this federate has a different name (*subscriber*).
- the simulation waits for the *publisher* federate to join the federation, and
- an object class is defined for the data to which this federate subscribes.

Indeed, the subscriber inputs are very similar to the publisher inputs above with the exception that local ownership is turned off, publishing is turned off, and subscribing is turned on.

The relevant object declarations are shown below.

```
// This federate has only one object, and it subscribes to it.
THLA.manager.obj_count = 1;
THLA.manager.objects = alloc(THLA.manager.obj_count);

// objects subscribed to
THLA.manager.objects[0].FOM_name        = "SimplesineStateAndParameters";
THLA.manager.objects[0].name            = "simplesineStateAndParameters";
THLA.manager.objects[0].create_HLA_instance = false;
THLA.manager.objects[0].attr_count      = 6;
THLA.manager.objects[0].attributes      = alloc(THLA.manager.objects[0].attr_count);

THLA.manager.objects[0].attributes[0].FOM_name = "Time";
THLA.manager.objects[0].attributes[0].trick_name = "sys.exec.out.time";
THLA.manager.objects[0].attributes[0].config   = THLA_CYCLIC;
THLA.manager.objects[0].attributes[0].publish  = false;
THLA.manager.objects[0].attributes[0].subscribe = true;
THLA.manager.objects[0].attributes[0].locally_owned = false;
THLA.manager.objects[0].attributes[0].rti_encoding = THLA_LITTLE_ENDIAN;

THLA.manager.objects[0].attributes[1].FOM_name = "Value";
THLA.manager.objects[0].attributes[1].trick_name = "subscriber.simplesine.state.x";
THLA.manager.objects[0].attributes[1].config   = THLA_INITIALIZE + THLA_CYCLIC;
THLA.manager.objects[0].attributes[1].publish  = false;
THLA.manager.objects[0].attributes[1].subscribe = true;
THLA.manager.objects[0].attributes[1].locally_owned = false;
THLA.manager.objects[0].attributes[1].rti_encoding = THLA_LITTLE_ENDIAN;

THLA.manager.objects[0].attributes[2].FOM_name = "dvdt";
THLA.manager.objects[0].attributes[2].trick_name = "subscriber.simplesine.state.x_dot";
THLA.manager.objects[0].attributes[2].config   = THLA_CYCLIC;
THLA.manager.objects[0].attributes[2].publish  = false;
THLA.manager.objects[0].attributes[2].subscribe = true;
THLA.manager.objects[0].attributes[2].locally_owned = false;
THLA.manager.objects[0].attributes[2].rti_encoding = THLA_LITTLE_ENDIAN;

THLA.manager.objects[0].attributes[3].FOM_name = "Phase";
THLA.manager.objects[0].attributes[3].trick_name = "subscriber.simplesine.params.phi";
THLA.manager.objects[0].attributes[3].config   = THLA_CYCLIC;
THLA.manager.objects[0].attributes[3].publish  = false;
THLA.manager.objects[0].attributes[3].subscribe = true;
THLA.manager.objects[0].attributes[3].locally_owned = false;
THLA.manager.objects[0].attributes[3].rti_encoding = THLA_LITTLE_ENDIAN;

THLA.manager.objects[0].attributes[4].FOM_name = "Frequency";
THLA.manager.objects[0].attributes[4].trick_name = "subscriber.simplesine.params.w";
THLA.manager.objects[0].attributes[4].config   = THLA_CYCLIC;
THLA.manager.objects[0].attributes[4].publish  = false;
THLA.manager.objects[0].attributes[4].subscribe = true;
THLA.manager.objects[0].attributes[4].locally_owned = false;
THLA.manager.objects[0].attributes[4].rti_encoding = THLA_LITTLE_ENDIAN;

THLA.manager.objects[0].attributes[5].FOM_name = "Amplitude";
THLA.manager.objects[0].attributes[5].trick_name = "subscriber.simplesine.params.A";
THLA.manager.objects[0].attributes[5].config   = THLA_CYCLIC;
THLA.manager.objects[0].attributes[5].publish  = false;
THLA.manager.objects[0].attributes[5].subscribe = true;
THLA.manager.objects[0].attributes[5].locally_owned = false;
THLA.manager.objects[0].attributes[5].rti_encoding = THLA_LITTLE_ENDIAN;
```

Listing 6.4: `SIM_simplesine_hla_sub` input file

6.5 Output

Together, these two simulations do in a distributed fashion what the single simulation did in Chapter 4. The publisher generates sine wave data and periodically sends updates to the subscriber. The subscriber receives the periodic updates and extrapolates the state until the next update arrives.

Figure 6.1 shows the sine wave as generated on the publisher. Figure 6.2 shows the sine wave as received by the subscriber with numerical integration disabled to emphasize the time of arrival of the data. Figure 6.3 shows the sine wave generated by the subscriber based on data received from the publisher with subscriber-side numerical integration between updates.

Close inspection of the publisher and subscriber plots reveals that they are slightly out of phase, hence the relatively large error in the third plot. This phase lag is due to HLA time management *lookahead*, which is 1.0sec in this case. Indeed, this effect can be seen clearly at $t = 1$, since up until that point the subscriber was integrating based solely on initial conditions (not based on any data received from the publisher), but at $t = 1$, the first data arrives from the publisher but it is approximately 1sec late, resulting in a discontinuity in the data. This lookahead-induced effect can be compensated using `TrickHLA` features discussed in Chapter 7.

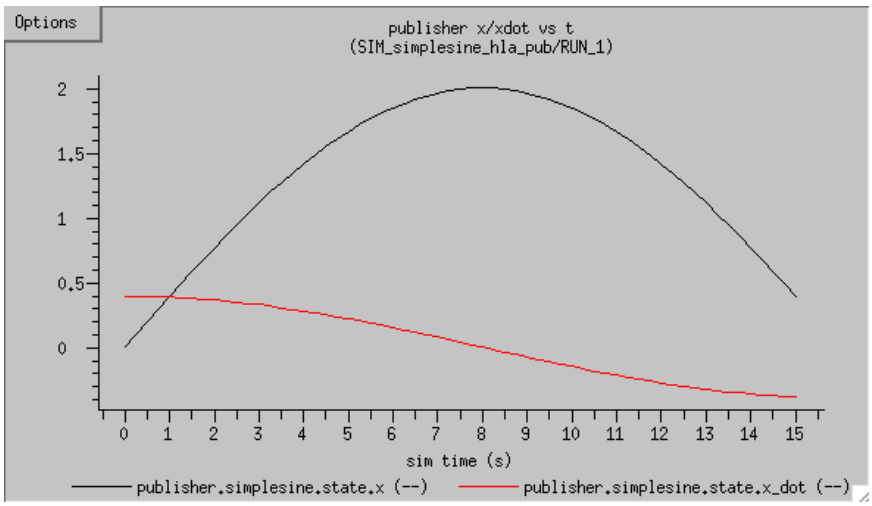

Figure 6.1: Output from `SIM_simplesine_hla_pub`

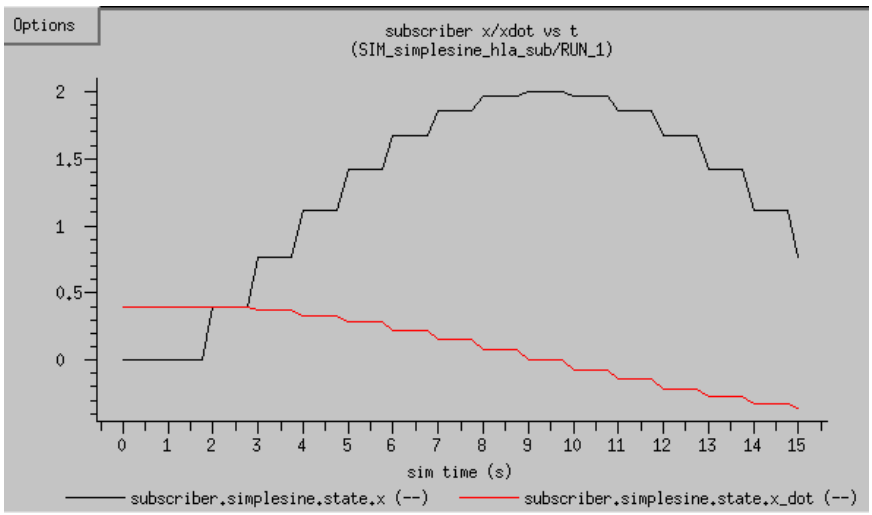

Figure 6.2: Output from `SIM_simplesine_hla_sub` (no integration)

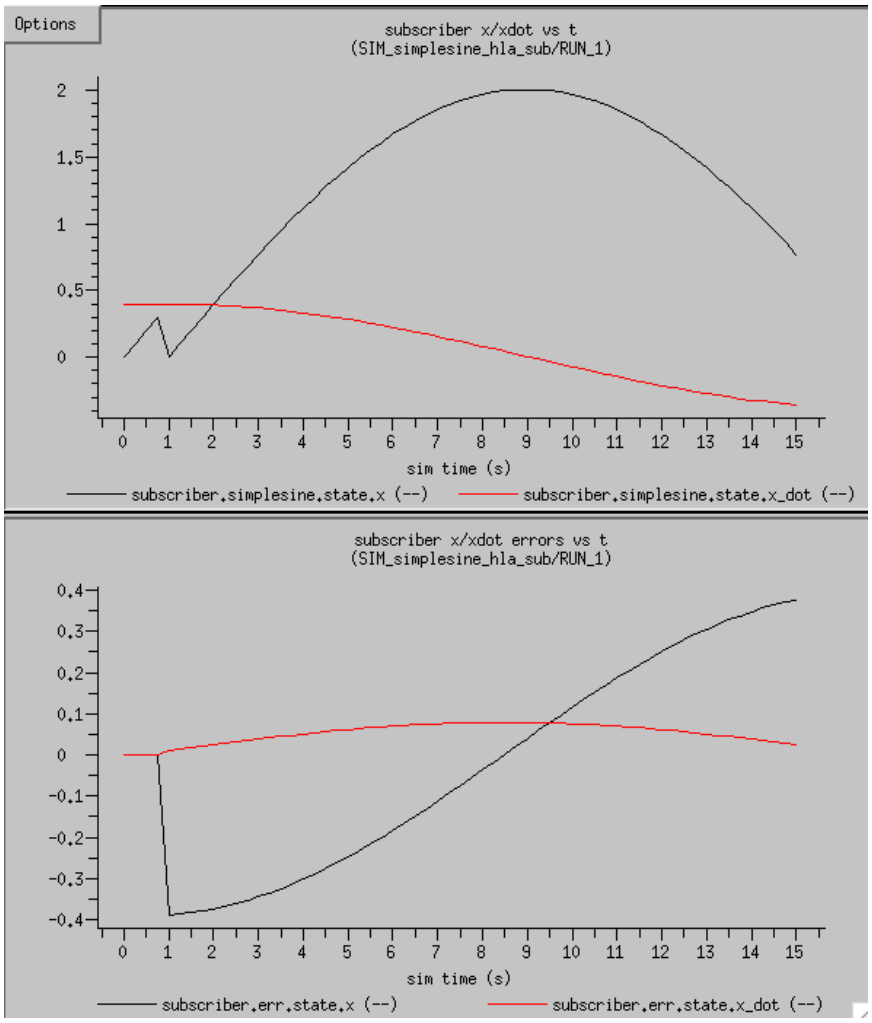

Figure 6.3: Output from `SIM_simplesine_hla_pub`

Chapter 7

Lag Compensation

In this chapter, we present simulations which compensate for time lags introduced due to HLA time management. The lags appear when published data are sent from one federate to its subscribers, and it becomes particularly noticeable when data ownership is transferred between federates.

There are two `TrickHLA` mechanisms which allow simulation developers to compensate for HLA-introduced lag: publisher-side and subscriber-side compensation.

7.1 What is a Lag Compensator?

7.1.1 The class `TrickHLALagCompensation`

`TrickHLA` defines a C++ class which may be subclassed by simulation developers in order to implement either kind of lag compensation. The class header is shown below.

```
1  class TrickHLALagCompensation
2  {
3     friend class InputProcessor;
4     friend void init_attrTrickHLALagCompensation();
5
6    public:
7     TrickHLALagCompensation() {};       // default constructor
8     virtual ~TrickHLALagCompensation() {}; // destructor
9
10    virtual void send_lag_compensation(); // RETURN: -- None.
11
12    virtual void receive_lag_compensation() // RETURN: -- None.
13 };
```

Listing 7.1: The `TrickHLALagCompensation` class

To use this class, simulation developers subclass it, overriding the relevant send/receive methods. The `TrickHLA` infrastructure automatically invokes those methods for publishers and subscribers at the appropriate time.

Developers only need to be concerned with the algorithm defined in these methods; the logistics of invoking them at the appropriate time is taken handled by `TrickHLA`. The following subsections illustrate how this is done in the `simplesine` model.

7.1.2 Lag compensation in `simplesine`

7.1.2.1 `simplesine_compensate()`

The `simplesine` model includes a `simplesine_compensate()` function which does most of the lag compensation work. It is based on the equations 3.2. Using those equations, the state at time $t + \Delta t$ can be written as

$$x(t + \Delta t) = A \sin(\omega(t + \Delta t) + \phi), \tag{7.1a}$$
$$\dot{x}(t + \Delta t) = A\omega \cos(\omega(t + \Delta t) + \phi). \tag{7.1b}$$

With a bit of trignometry and rearranging, the compensated state at time $t + \Delta t$ can be written in terms of the uncompensated state at time t as follows.

$$x(t + \Delta t) = x(t) \cos \omega\Delta t + \frac{\dot{x}(t)}{\omega} \sin \omega\Delta t \tag{7.2a}$$
$$\dot{x}(t + \Delta t) = \dot{x}(t) \cos \omega\Delta t - x(t)\omega \sin \omega\Delta t \tag{7.2b}$$

The compensate function is shown below. It takes an uncompensated state and Δt as inputs and calculates the corresonded compensated state at time $t + \Delta t$.

```
1  void simplesine_compensate(
2    simplesine_params_T* paramsP,
3    simplesine_state_T* uncompensated_stateP,
4    simplesine_state_T* compensated_stateP,
5    double dt )
6  {
7    const double w = paramsP->w;
8    const double wdt = w * dt;
9    const double sinwdt = sin( wdt );
10   const double coswdt = cos( wdt );

12   const double x    = uncompensated_stateP->x;
13   const double x_dot = uncompensated_stateP->x_dot;

15   // Calculate the compensated state.
16   const double x_compensated  = x     * coswdt +  x_dot * sinwdt / w ;
17   const double x_dot_compensated = x_dot * coswdt - x * sinwdt * w;

19   // Save the compenstated state.
20   compensated_stateP->x   = x_compensated;
21   compensated_stateP->x_dot = x_dot_compensated;
22 }
```

Listing 7.2: The `simplesine_compensate` function

7.1.2.2 `simplesine_LagCompensation`

The simplesine subsclass of `TrickHLALagCompensation` is shown below. It relies on the `simplesine_compensate()` function to do the real work, transforming the uncompensated state pointed to by `uncompensated_stateP` into a transformed state pointed to by `compensated_stateP`. These two pointers are initialized by the `initialize()` method, which is unique to this class (i.e., not part of the interface defined by the `TrickHLALagCompensation` class).

```
#include "simplesine.h"
#include "TrickHLA/include/TrickHLALagCompensation.hh"

class simplesine_LagCompensator : public TrickHLALagCompensation
{
  friend class InputProcessor;
  friend void init_attrsimplesine_LagCompensator();

  public:
   simplesine_LagCompensator();
   virtual ~simplesine_LagCompensator();

   int initialize( simplesine_T* sim_dataP, simplesine_T* lag_comp_dataP );
   virtual void send_lag_compensation();
   virtual void receive_lag_compensation();

  private:
    simplesine_T* uncompensated_stateP;
    simplesine_T* compensated_stateP;
};
```

Listing 7.3: The `simplesine_LagCompensation` class header

```
/******************************** TRICK HEADER *****************************
PURPOSE: (This class provides lag compensation.)
LIBRARY DEPENDENCY: ((simplesine_compensate.o))
*******************************************************************************/
// System include files.
#include <math.h>
#include <stdlib.h>
#include <iostream>
#include <string>

using namespace std;

#include "sim_services/include/exec_proto.h"
#include "trick_utils/math/include/trick_math.h"

#include "../include/simplesine_proto.h"
#include "../include/simplesine_LagCompensator.hh"

/* ------------------------------------------------------------------------
PURPOSE: (Default constructor for the sine wave lag compensation.)
------------------------------------------------------------------------*/
simplesine_LagCompensator::simplesine_LagCompensator() // RETURN: -- None.
{ }

/* ------------------------------------------------------------------------
PURPOSE: (Frees memory allocated.)
------------------------------------------------------------------------*/
simplesine_LagCompensator::~simplesine_LagCompensator() // RETURN: -- None.
{ }

/* ------------------------------------------------------------------------
PURPOSE: (Initializes the Sine Lag Compensation.)
------------------------------------------------------------------------*/
int simplesine_LagCompensator::initialize( // RETURN: -- Always returns zero.
     simplesine_T * uncompensated_stateP, // IN: -- Simulation data.
     simplesine_T * compensated_stateP ) // IN: -- Lag Compensation data.
{
   this->uncompensated_stateP = uncompensated_stateP;
   this->compensated_stateP = compensated_stateP;
   return( 0 );
}
```

```
43 /* ---------------------------------------------------------------------------
44 PURPOSE: (Send-side lag-compensation where we propagate the sine wave
45   state head by dt to predict the value at the next data cycle.)
46 ---------------------------------------------------------------------------*/
47 void simplesine_LagCompensator::send_lag_compensation() // RETURN: -- None.
48 {
49    double dt = get_fed_lookahead().getDoubleTime();
50
51    simplesine_compensate( &(uncompensated_stateP->params),
52                           &(uncompensated_stateP->state),
53                           &(compensated_stateP->state),
54                           dt );
55 }
56
57 /* ---------------------------------------------------------------------------
58 PURPOSE: (Receiveside lag-compensation where we propagate the sine wave
59   state ahead by dt to predict the value at the next data cycle.)
60 ---------------------------------------------------------------------------*/
61 void simplesine_LagCompensator::receive_lag_compensation() // RETURN: -- None.
62 {
63    double dt = get_fed_lookahead().getDoubleTime();
64
65    simplesine_compensate( &(uncompensated_stateP->params),
66                           &(uncompensated_stateP->state),
67                           &(compensated_stateP->state),
68                           dt );
69 }
```

Listing 7.4: The `simplesine_LagCompensation` class methods

7.2 Publisher-resident Compensation

This approach to lag compensation is handled by the publisher of data before it is sent out via HLA. Since the compensation is performed by the publisher, subscribers need not implement any lag-related logic. This section demonstrates how to implement kind of lag compensation.

7.2.1 `SIM_simplesine_hla_pub_lagComp`

Implementing publisher-side lag compensation in an existing (non-compensating) `TrickHLA` simulation is very easy. This simulation illustrates what is required. It is based on the `SIM_simplesine_hla_pub` simulation and differs only in that the `publisher` sim object has a few new elements.

The sim object is shown in Listing 7.5. The main differences between this simulation's `S_define` and that of the publisher discussed in Chapter 6 are

- the declaration of two `simplesine` variables, one for uncompensated data and the other for compensated,
- the declaration of a *lag compensator* variable, and
- the initialization job for the lag compensator

```
sim_object
{
  simplesine: simplesine_T uncompensated_simplesine (simplesine/data/simplesine.d);
  simplesine: simplesine_T compensated_simplesine (simplesine/data/simplesine.d);
  simplesine: simplesine_LagCompensator lag_compensator;

  (initialization) simplesine:
    simplesine_calc(
      simplesine_T* P = &publisher.uncompensated_simplesine,
      double t = sys.exec.out.time );

  (initialization) simplesine:
    publisher.lag_compensator.initialize(
      simplesine_T* uncompensatedP = &publisher.uncompensated_simplesine,
      simplesine_T* compensatedP = &publisher.compensated_simplesine );

  (PROPAGATE_TIMESTEP, scheduled) simplesine:
    simplesine_calc(
      simplesine_T* P = &publisher.uncompensated_simplesine,
      double t = sys.exec.out.time );
} publisher ;
```

Listing 7.5: `publisher` sim object for publisher-side lag compensation

7.2.2 Input file

The input file for the lag compensating publisher is virtually identical to the non-compensating publisher discussed previously with the following exceptions.

- The lag compensator defined in the `S_define` file is explicitly associated with the object being published and in particular is specified as a publisher-resident (sending) compensator by adding the following two lines to the input file:

  ```
  THLA.manager.objects[0].lag_comp      = &publisher.lag_compensator;
  THLA.manager.objects[0].lag_comp_type = THLA_LAG_COMP_SEND_SIDE;
  ```

- Since the `S_define` for the compensating publisher now includes two `simplesine` variables, one uncompensated and the other compensated, the lines in the input file which specify the Trick variable names to publish are changed accordingly from `publisher.simplesine` to `publisher.compensated_simplesine`.

7.2.3 Output

In this section, we show the results of running a subscriber, `SIM_simplesine_hla_sub` along with the lag-compensating publisher, `SIM_simplesine_hla_pub_lagComp`.

Unlike Figure 6.3 on page 35, where the subscriber is seen to be 1sec out of phase with the true state, when the subscriber received *lag compensated* data from the publisher, the subscriber's sine wave is in phase with the publisher as shown in Figure 7.1. The error plot on this graph closely resembles the error plot of Figure 4.3 on page 20, in which the publisher and subscriber are resident in the same simulation, showing that the publisher's lag compensation removes the HLA-induced lag.

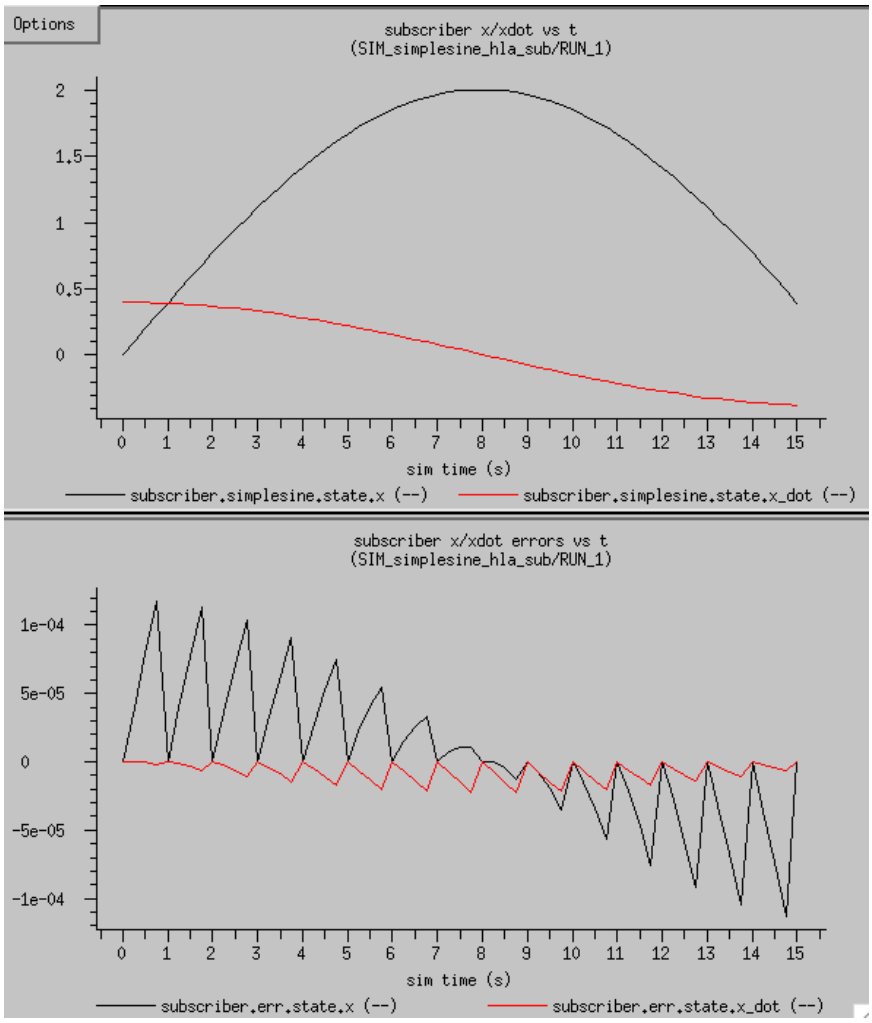

Figure 7.1: Output from `SIM_simplesine_hla_pub_lagComp`

7.3 Subscriber-resident Compensation

This approach to lag compensation is handled by subscribers receiving data from HLA. Each subscriber is responsible for implementing their own logic for handling the HLA-induced lag. This section demonstrates how to do this.

7.3.1 `SIM_simplesine_hla_sub_lagComp`

This simulation illustrates what is required to implement subscriber-side lag compensation in an existing (non-compensating) simulation. It is based on the `SIM_simplesine_hla_sub` simulation and differs only in that the `subscriber` sim object has a few new elements.

The sim object is shown in Listing 7.6. The main differences between this simulation's `S_define` and that of the subscriber discussed in Chapter 6 are

- the declaration of two `simplesine` variables, one for uncompensated data and the other for compensated,
- the declaration of a *lag compensator* variable, and
- the initialization job for the lag compensator

```
1  sim_object
2  {
3    sim_services/include: INTEGRATOR integ (simplesine/data/integ.d);
4    simplesine: simplesine_T uncompensated_simplesine (simplesine/data/simplesine.d);
5    simplesine: simplesine_T compensated_simplesine (simplesine/data/simplesine.d);
6    simplesine: simplesine_T compensated_simplesine_error (simplesine/data/simplesine.d);
7    simplesine: simplesine_LagCompensator lag_compensator;

9    (initialization) simplesine:
10     simplesine_calc(
11       simplesine_T* P = &subscriber.uncompensated_simplesine,
12       double t = sys.exec.out.time );
13   (initialization) simplesine:
14     simplesine_calc(
15       simplesine_T* P = &subscriber.compensated_simplesine,
16       double t = sys.exec.out.time );
17   (initialization) simplesine:
18     subscriber.lag_compensator.initialize(
19       simplesine_T* uncompensatedP = &subscriber.uncompensated_simplesine,
20       simplesine_T* compensatedP = &subscriber.compensated_simplesine );

22   (derivative) simplesine:
23     simplesine_deriv( simplesine_T* p = &subscriber.compensated_simplesine );
24   (integration) simplesine:
25     simplesine_integ(
26       INTEGRATOR* I = &subscriber.integ,
27       simplesine_T* p = &subscriber.compensated_simplesine );

29   (PROPAGATE_TIMESTEP, scheduled) simplesine:
30     simplesine_calcError(
31       double t = sys.exec.out.time,
32       simplesine_T* s = &subscriber.compensated_simplesine,
33       simplesine_state_T* err = &subscriber.compensated_simplesine_error.state );
34 } subscriber ;
```

Listing 7.6: `subscriber` sim object for subscriber-side lag compensation

7.3.2 Input file

The input file for the lag compensating subscriber is virtually identical to the non-compensating subscriber with the following exceptions.

- The lag compensator defined in the `S_define` file is explicitly associated with the object being subscribed to and in particular is specified as a subscriber-resident (receiving) compensator by adding the following two lines to the input file:

```
THLA.manager.objects[0].lag_comp      = &subscriber.lag_compensator;
THLA.manager.objects[0].lag_comp_type = THLA_LAG_COMP_RECEIVE_SIDE;
```

- Since the `S_define` for the compensating subscriber now includes uncompensated and the other compensated `simplesine` variables, the lines in the input file which specify the Trick variable names to publish are changed accordingly from `subscriber.simplesine` to `subscriber.uncompensated_simplesine`.

Notice that the simulation subscribes to data from the publisher which are saved in the *uncompensated* `simplesine` variable, since the publisher in this case does no lag compensation. The `TrickHLA` infrastructure takes care of executing the lag compensator, which calculates the compensated `simplesine` state by invoking the `simplesine_LagCompensator` defined in the `S_define` file.

7.3.3 Output

In this section, we show the results of running a publisher, `SIM_simplesine_hla_pub` along with the lag-compensating subscriber, `SIM_simplesine_hla_sub_lagComp`.

Again, unlike Figure 6.3 on page 35, where the subscriber is out of phase with the true state, in this case the subscriber's compensated sine wave is in phase with the publisher as shown in Figure 7.2.

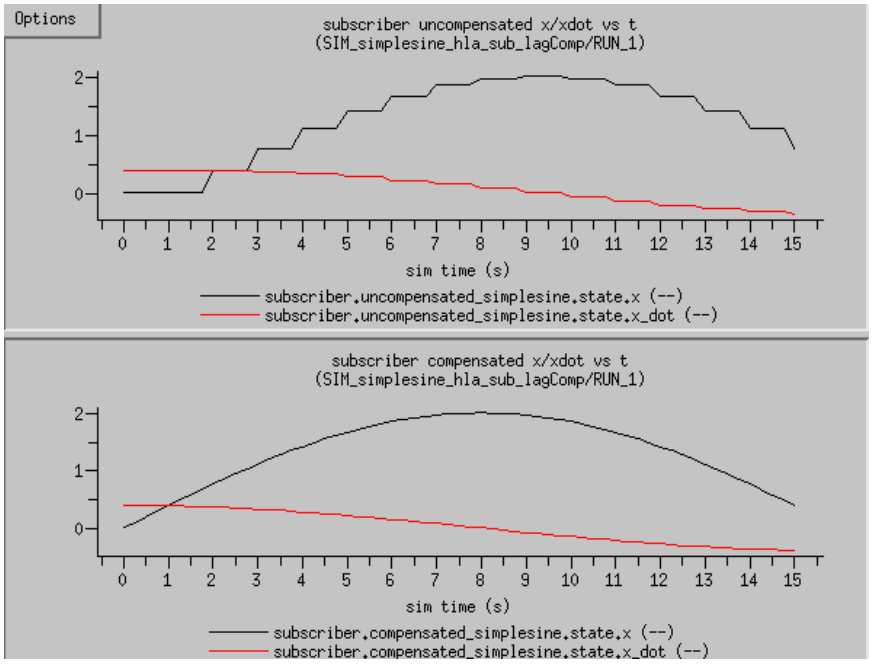

Figure 7.2: Output from `SIM_simplesine_hla_sub_lagComp`

Chapter 8

Sending and Receiving Interactions

This section illustrates how to use `TrickHLA` to send and receive HLA interactions. The example simulations are similar to the publisher and subscriber simulations discussed earlier, but in this case, the subscriber's sine parameters have been initialized with a zero-value amplitude. Thus, the subscriber's state is initially constant, $x(t) = 0$, $\dot{x}(t) = 0$. However, at a certain point in the simulation, the publisher sends its sine parameters (in particular, a non-zero amplitude) to the subscriber in an interaction. Once the subscriber receives these values and updates the appropriate Trick variables, $x(t)$ and $\dot{x}(t)$ assume their expected sine wave shapes.[1]

8.1 What is an interaction handler?

8.1.1 The class `TrickHLAInteractionHandler`

`TrickHLA` defines a C++ class which may be subclassed by simulation developers in order to send or receive interactions. The class header is shown below. It defines *send* and *receive* methods which may be called by simulation developers, avoiding the need to directly use the `TrickHLAInteraction` class, which is fairly complex.

There are actually two send methods. The one with no arguments, sends the interaction to the receiver, where it is delivered in the order interactions arrive off the network. The send method with a single timetag argument sends the interaction to receivers where it is delivered in time stamp order. The timetag argument must be a simulation timetag plus some lookahead interval. These send methods are sufficient unto themselves and do not need to be overridden in a subclass. Indeed they are not virtual methods.

The receive method is virtual. The `TrickHLA` infrastructure will automatically invoke a subclass's corresponding version of the method when interactions arrive from remote senders.

The protected `interaction` field may be ignored. The `TrickHLA` infrastructure automatically takes care of creating interactions according to the declarations encountered in the input file.

[1] No state data are exchanged in this interaction, only the `simplesine` parameters, A, ϕ and ω. The calculation of the state on the subscriber side is only used as a way to graphically illustrate the arrival of the interaction from the publisher.

```
1  class TrickHLAInteraction;
2  #include "TrickHLA/include/TrickHLAInteraction.hh"

4  class TrickHLAInteractionHandler
5  {
6     friend class InputProcessor;
7     friend void init_attrTrickHLAInteractionHandler();

9    public:
10    TrickHLAInteractionHandler();
11    virtual ~TrickHLAInteractionHandler();

13    virtual void initialize_callback( TrickHLAInteraction * inter );

15    bool send_interaction(); // Receive Order
16    bool send_interaction( double send_time ); // Timestamp Order

18    TrickHLADoubleInterval get_fed_lookahead();
19    TrickHLADoubleTime   get_granted_fed_time();

21    virtual void receive_interaction();

23   protected:
24    TrickHLAInteraction * interaction;
25 };
```

Listing 8.1: The `TrickHLAInteractionHandler` class

8.1.2 Interaction handling in `simplesine`

The class header for the `simplesine` interaction handler is shown below.

```
1  #include "TrickHLA/include/TrickHLAInteractionHandler.hh"

3  class simplesine_InteractionHandler : public TrickHLAInteractionHandler
4  {
5    friend class InputProcessor;
6    friend void init_attrsimplesine_InteractionHandler();

8    public:
9      simplesine_InteractionHandler();
10     virtual ~simplesine_InteractionHandler();

12     void send_sine_interaction( double send_time );
13     virtual void receive_interaction();

15   protected:
16     double lookahead_time;
17 };
```

Listing 8.2: `simplesine_InteractionHandler` header file

And the methods are shown below. The `send_sine_interaction()` method is just a wrapper around the parent class's `send_interaction()` method.

And the `receive_interaction()` method, just invokes the Trick output function, `send_hs()` to indicate that an interaction arrived; by the time the receive method is invoked, the data from the incoming interaction have already been stored in the Trick variable that is associated with the

interaction in the input file. (See below.) So there is nothing much to do in the implementation of the receive method.

```
1  /******************************** TRICK HEADER ****************************
2  PURPOSE: (Send/receive HLA interactions.)
3  ******************************************************************************/
4
5  #include <stdlib.h>
6  #include <string>
7  #include "../include/simplesine_InteractionHandler.hh"
8
9  using namespace std;
10
11 /* ----------------------------------------------------------------------------
12 PURPOSE: (Default constructor)
13 ----------------------------------------------------------------------------*/
14 simplesine_InteractionHandler::simplesine_InteractionHandler() // RETURN: -- None.
15 : lookahead_time(0.0)
16 { }
17
18
19 /* ----------------------------------------------------------------------------
20 PURPOSE: (Destructor.)
21 ----------------------------------------------------------------------------*/
22 simplesine_InteractionHandler::~simplesine_InteractionHandler() // RETURN: -- None.
23 { }
24
25
26 /* ----------------------------------------------------------------------------
27 PURPOSE: (Send this handler's HLA interaction. A pointer to the interaction
28   is stored in this class -- defined in the protected <interaction> field
29   in the parent class. The TrickHLA infrastructure takes care of setting
30   that pointer when it sees interactions declared in the Trick input file.)
31 ----------------------------------------------------------------------------*/
32 void simplesine_InteractionHandler::send_sine_interaction( // RETURN: -- None.
33    double send_time )      // IN: s HLA time to send the interaction.
34 {
35    const char* FOM_name = (const char*)this->interaction->get_FOM_name();
36    bool interaction_was_sent = false;
37    double timetag = send_time + lookahead_time;
38
39    bool was_sent = this->TrickHLAInteractionHandler::send_interaction( timetag );
40
41    if( was_sent ) {
42       const char* msg = string("sent␣interaction:␣").append(FOM_name).c_str();
43       send_hs( stdout, (char*)msg );
44    } else {
45       const char* msg = string("error␣sending␣interaction:␣").append(FOM_name).c_str();
46       send_hs( stderr, (char*)msg );
47    }
48 }
49
50
51 /* ----------------------------------------------------------------------------
52 PURPOSE: (Handle an incoming interaction.)
53 ----------------------------------------------------------------------------*/
54 void simplesine_InteractionHandler::receive_interaction() // RETURN: -- None.
55 {
56    const char* FOM_name = (const char*)this->interaction->get_FOM_name();
57    const char* msg = string("Received␣interaction:␣").append(FOM_name).c_str();
58
59    send_hs( stdout, (char*)msg );
60 }
```

Listing 8.3: `simplesine_InteractionHandler` methods

8.2 SIM_simplesine_hla_sendInt

This simulation is based on the publisher simulation, `SIM_simplesine_hla_pub` even though it does no real publishing. Instead, it just sends an interaction once during the simulation. The interaction handler *send* method is invoked at a specified time during the simulation, using the Trick `CALL` directive.[2] To do this, you must do two things to the `S_define` file.

- Define an instance of the interaction handler, and
- Specify a 0-frequency job for the *send* method.[3]

Thus, the `publisher` sim object for this simulation has two lines that do not appear in the plain publisher:

```
  sim_object {
    ...
    simplesine: simplesine_InteractionHandler interaction_handler;
    ...
    (0.0, scheduled) simplesine:
      publisher.interaction_handler.send_sine_interaction(
        In double time = sys.exec.out.time );
    ...
} publisher;
```

Listing 8.4: Sending interaction handler `S_define` changes

8.3 Sender input

The sender's input file is based on the plain publisher's input file. They differ in the following ways.

- The federate name is *sender* instead of *publisher*, and it waits for the *receiver* federate instead of *subscriber*,
- since this simulation does not actually publish anything, there are no objects and attributes defined,
- the interaction handler lookahead time is specified,
- the interaction and parameters to be sent are specified, and
- there is a `CALL` directive invoking the interaction handler method, `send_sine_interaction()`.

The complete input file is listed in Appendix B on page 104.

[2] More sophisticated simulations might call the method directly from simulation-specific code. We do not do that here.

[3] 0-frequency jobs are a common way to force Trick to generate code for the job even though it is not actually part of the periodically scheduled jobs. In this case, the send method is not invoked from a regularly scheduled job, but from a single invocation of it as specified in the input file. If we did not specify a 0-frequency job like this, Trick would not compile the code for the job, and we would get a runtime error.

8.4 SIM_simplesine_hla_receiveInt

This simulation is based on the plain subscriber simulation, `SIM_simplesine_hla_sub` even though it does no real subscribing. Instead, it just waits for an interaction to arrive. The `TrickHLA` infrastructure automatically handles incoming interactions and assigns the associated parameter values to Trick variables as specified in the input file. (See next section.) There are no explicit jobs to schedule in the `S_define` file, and there are no jobs to `CALL` from the input file, either.

To do this, all you need to do is declare an interaction handler in the `subscriber` sim object.[4]

The `S_define` for the receiver is virtually identical with that of the plain subscriber with the following exceptions.

- A interaction handler variable is defined:

```
simplesine: simplesine_InteractionHandler interaction_handler;
```

- The numerical integration of the the plain subscriber's state has been replaced with the analytical propagation, since this method is sensitive to changes in the parameters, whereas the numerical integration of the state does not sense changes in the value of the amplitude parameter.

this `S_define` file for this simulation is the same as the plain subscriber's.

8.5 Receiver input

The receiver's input file is based on the plain subscriber's input file. They differ in the following ways.

- The federate name is *receiver* instead of *subscriber*, and it waits for the *sender* federate instead of *publisher*,
- since this simulation does not actually subscribe to anything, there are no objects and attributes defined, and
- the interaction and parameters to be sent are specified.

The complete input file is listed in Appendix B on page 107.

In addition, for this simulation, the initial `simplesine` amplitude parameter is initialized to zero (in the `subscriber.d` initialization file included from the input file). Until an incoming interaction resets the parameters, the receiver's $x(t)$ and $\dot{x}(t)$ will be constant zero-valued functions until a non-zero amplitude parameter arrives in an interaction from from the sender.

[4] Unlike the case of sending interactions, receiving interactions requires no jobs be executed by Trick. Consequently, in this case (unlike the interaction sender), there is no need to declare a zero-frequency job in the `S_define` file.

8.6 Output

Output from the sender shown below reveals that it does indeed send an interaction to the federation at $t = 4$.

```
...
| |wormhole|1|0.00|2007/08/03,22:27:05| TrickHLAManager::initialization_complete()
       Simulation has started and is now running...
| |wormhole|1|0.00|2007/08/03,22:27:05| TrickHLAManager::receive_cyclic_data()
| |wormhole|1|0.00|2007/08/03,22:27:05| TrickHLAManager::send_cyclic_data()
| |wormhole|-1|0.25|2007/08/03,22:27:05| TrickHLAFedAmb::timeAdvanceGrant()
Federate "sender" Time granted to: 1
| |wormhole|1|1.00|2007/08/03,22:27:06| TrickHLAManager::receive_cyclic_data()
| |wormhole|1|1.00|2007/08/03,22:27:06| TrickHLAManager::send_cyclic_data()
| |wormhole|-1|1.25|2007/08/03,22:27:06| TrickHLAFedAmb::timeAdvanceGrant()
Federate "sender" Time granted to: 2
| |wormhole|1|2.00|2007/08/03,22:27:07| TrickHLAManager::receive_cyclic_data()
| |wormhole|1|2.00|2007/08/03,22:27:07| TrickHLAManager::send_cyclic_data()
| |wormhole|-1|2.25|2007/08/03,22:27:07| TrickHLAFedAmb::timeAdvanceGrant()
Federate "sender" Time granted to: 3
| |wormhole|1|3.00|2007/08/03,22:27:08| TrickHLAManager::receive_cyclic_data()
| |wormhole|1|3.00|2007/08/03,22:27:08| TrickHLAManager::send_cyclic_data()
| |wormhole|-1|3.25|2007/08/03,22:27:08| TrickHLAFedAmb::timeAdvanceGrant()
Federate "sender" Time granted to: 4
| |wormhole|1|4.00|2007/08/03,22:27:09| TrickHLAInteraction::send() Timestamp Order:
     Interaction 'SimplesineParameters' sent for time 4.999 seconds.
| |wormhole|1|4.00|2007/08/03,22:27:09| sent interaction: SimplesineParameters
| |wormhole|1|4.00|2007/08/03,22:27:09| TrickHLAManager::receive_cyclic_data()
| |wormhole|1|4.00|2007/08/03,22:27:09| TrickHLAManager::send_cyclic_data()
| |wormhole|-1|4.25|2007/08/03,22:27:09| TrickHLAFedAmb::timeAdvanceGrant()
Federate "sender" Time granted to: 5
...
```

Listing 8.5: *sender* output showing interaction at $t = 4$

Similarly, output from the receiver shown below reveals that the interaction did indeed arrive.

```
...
| |wormhole|1|0.00|2007/08/03,22:27:05| TrickHLAManager::initialization_complete()
       Simulation has started and is now running...
| |wormhole|1|0.00|2007/08/03,22:27:05| TrickHLAManager::receive_cyclic_data()
| |wormhole|1|0.00|2007/08/03,22:27:05| TrickHLAManager::send_cyclic_data()
| |wormhole|-1|0.25|2007/08/03,22:27:05| TrickHLAFedAmb::timeAdvanceGrant()
Federate "receiver" Time granted to: 1
| |wormhole|1|1.00|2007/08/03,22:27:06| TrickHLAManager::receive_cyclic_data()
| |wormhole|1|1.00|2007/08/03,22:27:06| TrickHLAManager::send_cyclic_data()
| |wormhole|-1|1.25|2007/08/03,22:27:06| TrickHLAFedAmb::timeAdvanceGrant()
Federate "receiver" Time granted to: 2
| |wormhole|1|2.00|2007/08/03,22:27:07| TrickHLAManager::receive_cyclic_data()
| |wormhole|1|2.00|2007/08/03,22:27:07| TrickHLAManager::send_cyclic_data()
| |wormhole|-1|2.25|2007/08/03,22:27:07| TrickHLAFedAmb::timeAdvanceGrant()
Federate "receiver" Time granted to: 3
| |wormhole|1|3.00|2007/08/03,22:27:08| TrickHLAManager::receive_cyclic_data()
| |wormhole|1|3.00|2007/08/03,22:27:08| TrickHLAManager::send_cyclic_data()
| |wormhole|-1|3.25|2007/08/03,22:27:08| TrickHLAFedAmb::timeAdvanceGrant()
Federate "receiver" Time granted to: 4
| |wormhole|1|4.00|2007/08/03,22:27:09| TrickHLAManager::receive_cyclic_data()
| |wormhole|1|4.00|2007/08/03,22:27:09| TrickHLAManager::send_cyclic_data()
| |wormhole|-1|4.25|2007/08/03,22:27:09| TrickHLAManager::receive_TSO_interaction() ID:77, HLA time:4.999
| |wormhole|-1|4.25|2007/08/03,22:27:09| TrickHLAFedAmb::timeAdvanceGrant()
Federate "receiver" Time granted to: 5
...
```

Listing 8.6: *receiver* output showing interaction at $t = 4$

Note that the interaction has a timestamp of 4.999, which is $t_{sim} + t_{lookahead}$. (The lookahead is set to 1.000sec in the sender's input file.)

Finally, the plot below shows the analytically propagated subscriber state. In particular, it reveals that upon arrival of the parameter values at $t = 4.999$, the functions $x(t)$ and $\dot{x}(t)$ pick up their expected sine wave forms, confirming the arrival of the interaction.

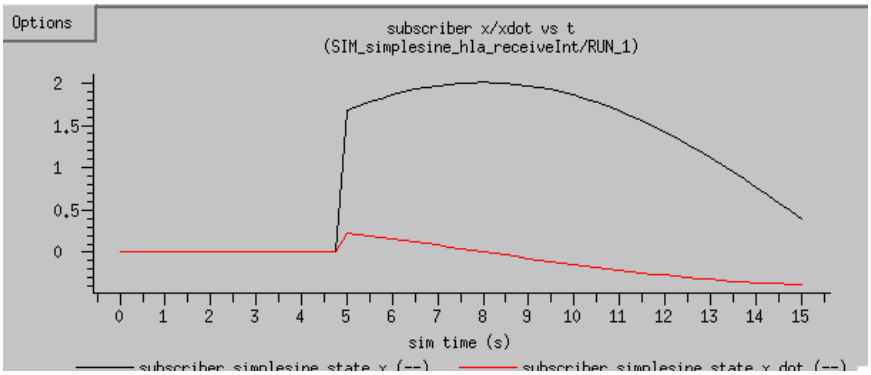

Figure 8.1: $x(t)$ and $\dot{x}(t)$ from `SIM_simplesine_hla_receiveInt`

Chapter 9

Ownership Transfer

There are two `TrickHLA` mechanisms for initiating ownership transfer: the current owner may *divest* itself of ownership contingent on the existence of some other federate that is willing to assume it, or a non-owner may *acquire* ownership from the current owner contingent on that federate being willing. The `TrickHLA` terminology for these two approaches to ownership transfer is *push* and *pull*, respectively. This chapter illustrates how to write simulations that push and/or pull ownership.

The simulations we present are

- A publisher that initially owns the `simplesine` state and *pushes* ownership away at a specified time in the run then at some subsequent time *pulls* ownership back, and
- Another publisher that does not initially own the `simplesine` state but will assume ownership when the other federate divests it and will surrender it when the original federate tries to pull ownership back.

`TrickHLA` automatically ensures that publishers (federates who explicitly indicate their *ability* to update the associated data) are eligible to assume ownership when the current owner divests itself of ownership by invoking the `TrickHLA` *push* method. The non-owning publisher does not need to take any explicit action for this to happen.[1] Similarly, when a non-owning publisher invokes the `TrickHLA` *pull* method, transfer of ownership from the current owner to the requesting federate is automatically handled. The current owner needs not explicitly handle the transfer request.[2]

9.1 What is an ownership handler?

`TrickHLA` includes a utility class that may be used directly (i.e., no subclassing necessary) to push and pull ownership. The class header is shown below.

```
1 #include <string>
```

[1] Thus, in the HLA nomenclature, the distinction between a publisher that *owns* the object/attributes and a publisher that *does not own* them is important. For former may generate updates for them, and the latter may not. Thus, ownership tranfer leads to a change in which publisher is actually generating data updates.

[2] Indeed, `TrickHLA` does not currently provide any mechanism for the current owner to decline a pull request.

```
 3  #include "TrickHLA/include/TrickHLAStandardsSupport.hh"
 4  class TrickHLAObject;
 5  class TrickHLAAttribute;
 6  #include "TrickHLA/include/TrickHLAObject.hh"
 7  #include "TrickHLA/include/TrickHLAAttribute.hh"
 8  #include "TrickHLA/include/TrickHLATypesNoICG.hh"
 9  #include "TrickHLA/include/TrickHLAOwnershipHandlerNoICG.hh"
10  #include "TrickHLA/include/TrickHLAOwnershipItem.hh"
11  #include "TrickHLA/include/TrickHLADoubleInterval.hh"
12  #include "TrickHLA/include/TrickHLADoubleTime.hh"

14  #include RTI1516_HEADER

16  using namespace std;
17  using namespace RTI1516_NAMESPACE;

19  class TrickHLAOwnershipHandler
20  {
21    friend class InputProcessor;
22    friend void init_attrTrickHLAOwnershipHandler();

24    friend class TrickHLAObject;

26    public:
27     TrickHLAOwnershipHandler();
28     virtual ~TrickHLAOwnershipHandler();

30     void checkpoint_requests();
31     void clear_checkpoint();
32     void restore_requests();

34     virtual void initialize_callback( TrickHLAObject * obj );

36     string get_object_name();
37     string get_object_FOM_name();

39     int get_attribute_count();
40     VectorOfStrings get_attribute_FOM_names() const;

42     bool is_locally_owned( const char * attribute_FOM_name );
43     bool is_remotely_owned( const char * attribute_FOM_name );

45     bool is_published( const char * attribute_FOM_name );
46     bool is_subscribed( const char * attribute_FOM_name );

48     void pull_ownership();
49     void pull_ownership( double time );
50     void pull_ownership( const char * attribute_FOM_name );
51     void pull_ownership( const char * attribute_FOM_name, double time );

53     void push_ownership();
54     void push_ownership( double time );
55     void push_ownership( const char * attribute_FOM_name );
56     void push_ownership( const char * attribute_FOM_name, double time );

58     TrickHLADoubleInterval get_fed_lookahead();
59     TrickHLADoubleTime   get_granted_fed_time();

61    private:
62     TrickHLAAttribute * get_attribute( const char * attribute_FOM_name );
63     TrickHLAObject * object; // ** Reference to the TrickHLA Object.
64     AttributeOwnershipMap pull_requests; // ** Map of pull ownership user requests.
65     AttributeOwnershipMap push_requests; // ** Map of push ownership user requests.
66     int                   pull_items_cnt; // -- # of pull items
67     TrickHLAOwnershipItem * pull_items; // -- array of pulled attributes
```

```
68    int                   push_items_cnt; // -- # of push items
69    TrickHLAOwnershipItem * push_items; // -- array of pushed attributes
70 };
```

Listing 9.1: `TrickHLAOwnershipHandler` class header

The `pull_ownership()` and `push_ownership()` methods are of particular interest. The methods without arguments push or pull all the attributes of the associated object immediately. The methods with only a time argument push or pull all the attributes of the specific object at the specified future time. And the methods with an explicit attribute name argument allow pushing and pulling of single attributes within an object without affecting the others.

9.2 SIM_simplesine_hla_own

Our illustration of how to use the `TrickHLAOwnershipHandler` class involves two different instances of a single simulation.

The first instance is the *active* publisher that initially owns the relevant object and attributes and explicitly calls the `push_ownership()` and `pull_ownership()` methods. The second instance is a *passive* publisher that does not initially own the object and attributes, nor does it explicitly request any ownership transfers. Instead, the passive instance of the simulation gains and surrenders ownership as the result of the remote push/pull requests by the active instance.

Both instances not only publish the `simplesine` data, but they also subscribe, allowing them to receive updates from the other instance when the object and attributes are remotely owned.

The two instances share the same `S_define` file, which is derived from the plain publisher. The only differences between the `S_define` file for the `SIM_simplesine_hla_own` simulation and that of the plain publisher are

- an ownership handler variable is defined by adding the following line,

  ```
  TrickHLA: TrickHLAOwnershipHandler ownership_handler;
  ```

- and zero-frequency jobs are declared for `push_ownership()` and `pull_ownership()` as follows

  ```
  (0.0, scheduled) TrickHLA: publisher.ownership_handler.push_ownership();
  (0.0, scheduled) TrickHLA: publisher.ownership_handler.pull_ownership();
  ```

  so that the methods may be invoked from the input file using the `CALL` directive.

9.3 Active input file

The input file for the active ownership transfer simulation is based on the input file for the plain publisher. The differences are:

- A new file, `LogData/THLA_objects.d` is included. This file contains inputs that ensure that the ownership flag `THLA.manager.objects[0].attributes[0].locally_owned` is logged. This allows us to plot the ownership transfer as it takes place.

- The federate name is *active* instead of *publisher*.

- The federate waits for a second federate named *passive* instead of *subscriber*.

- The ownership handler defined in the `S_define` file is associated with the object as follows.

```
THLA.manager.objects[0].ownership = &publisher.ownership_handler;
```

- The `.subscribe` flag of the object's attributes are all set to true so that the simulation will receive updated values when the other federate owns the object.

- The simulation *pushes* ownership away at $t = 5$ using the `CALL` directive as follows.

```
read = 5.0;
CALL publisher.publisher.ownership_handler.push_ownership();
```

- And finally the simulation *pulls* ownership back at $t = 10$ as follows.

```
read = 10.0;
CALL publisher.publisher.ownership_handler.pull_ownership();
```

9.4 Passive input file

The input file for the passive ownership transfer simulation is virtually identical to the active input file with the following exceptions.

- The federate name is *passive*.

- The attribute `.locally_owned` flags are initially false, since at simulation start the passive simulation does not own them.

- There are no `CALL` directives.

That last point deserves emphasizing. This simulation accepts ownership when the *active* federate pushes ownership away from itself; no explicit action is required on the part of the passive simulation in order to accept ownerhip. Similarly, this simulation willingly surrenders ownership when the *active* federate pulls it back; again, this happens without any explicit action on the part of the passive simulation.

9.5 Output

Inspection of the (abbreviated) output shown below from the active simulation reveals that it does indeed divest ownership and then reacquire it later.

```
...
| |wormhole|1|3.00|2007/08/05,21:56:18| TrickHLAManager::receive_cyclic_data()
| |wormhole|1|3.00|2007/08/05,21:56:18| TrickHLAManager::send_cyclic_data()
| |wormhole|-1|3.25|2007/08/05,21:56:18| TrickHLAFedAmb::timeAdvanceGrant()
Federate "active" Time granted to: 4
| |wormhole|1|4.00|2007/08/05,21:56:19| TrickHLAManager::receive_cyclic_data()
| |wormhole|1|4.00|2007/08/05,21:56:19| TrickHLAManager::send_cyclic_data()
| |wormhole|-1|4.25|2007/08/05,21:56:19| TrickHLAFedAmb::timeAdvanceGrant()
Federate "active" Time granted to: 5
| |wormhole|1|5.00|2007/08/05,21:56:20| TrickHLAManager::receive_cyclic_data()
| |wormhole|1|5.00|2007/08/05,21:56:20| TrickHLAManager::send_cyclic_data()
| |wormhole|1|5.00|2007/08/05,21:56:20| TrickHLAObject::push_ownership()
...
| |wormhole|1|6.00|2007/08/05,21:56:21| TrickHLAObject::release_ownership()
   DIVESTED Ownership of attribute 'SimplesineStateAndParameters'->'Time'
   of object 'simplesineStateAndParameters'.
| |wormhole|1|6.00|2007/08/05,21:56:21| TrickHLAObject::release_ownership()
   DIVESTED Ownership of attribute 'SimplesineStateAndParameters'->'Value'
   of object 'simplesineStateAndParameters'.
| |wormhole|1|6.00|2007/08/05,21:56:21| TrickHLAObject::release_ownership()
   DIVESTED Ownership of attribute 'SimplesineStateAndParameters'->'dvdt'
   of object 'simplesineStateAndParameters'.
| |wormhole|1|6.00|2007/08/05,21:56:21| TrickHLAObject::release_ownership()
   DIVESTED Ownership of attribute 'SimplesineStateAndParameters'->'Phase'
   of object 'simplesineStateAndParameters'.
| |wormhole|1|6.00|2007/08/05,21:56:21| TrickHLAObject::release_ownership()
   DIVESTED Ownership of attribute 'SimplesineStateAndParameters'->'Frequency'
   of object 'simplesineStateAndParameters'.
| |wormhole|1|6.00|2007/08/05,21:56:21| TrickHLAObject::release_ownership()
   DIVESTED Ownership of attribute 'SimplesineStateAndParameters'->'Amplitude'
   of object 'simplesineStateAndParameters'.
...
| |wormhole|1|9.00|2007/08/05,21:56:24| TrickHLAManager::receive_cyclic_data()
| |wormhole|1|9.00|2007/08/05,21:56:24| TrickHLAManager::send_cyclic_data()
| |wormhole|-1|9.25|2007/08/05,21:56:24| TrickHLAFedAmb::timeAdvanceGrant()
Federate "active" Time granted to: 10
| |wormhole|1|10.00|2007/08/05,21:56:25| TrickHLAManager::receive_cyclic_data()
| |wormhole|1|10.00|2007/08/05,21:56:25| TrickHLAManager::send_cyclic_data()
| |wormhole|1|10.00|2007/08/05,21:56:25| TrickHLAObject::pull_ownership()
...
   ACQUIRED ownership of attribute 'SimplesineStateAndParameters'->'Time'
   of object 'simplesineStateAndParameters'.
| |wormhole|-1|11.25|2007/08/05,21:56:26| TrickHLAFedAmb::attributeOwnershipAcquisitionNotification()
   ACQUIRED ownership of attribute 'SimplesineStateAndParameters'->'Value'
   of object 'simplesineStateAndParameters'.
| |wormhole|-1|11.25|2007/08/05,21:56:26| TrickHLAFedAmb::attributeOwnershipAcquisitionNotification()
   ACQUIRED ownership of attribute 'SimplesineStateAndParameters'->'dvdt'
   of object 'simplesineStateAndParameters'.
| |wormhole|-1|11.25|2007/08/05,21:56:26| TrickHLAFedAmb::attributeOwnershipAcquisitionNotification()
   ACQUIRED ownership of attribute 'SimplesineStateAndParameters'->'Phase'
   of object 'simplesineStateAndParameters'.
| |wormhole|-1|11.25|2007/08/05,21:56:26| TrickHLAFedAmb::attributeOwnershipAcquisitionNotification()
   ACQUIRED ownership of attribute 'SimplesineStateAndParameters'->'Frequency'
   of object 'simplesineStateAndParameters'.
| |wormhole|-1|11.25|2007/08/05,21:56:26| TrickHLAFedAmb::attributeOwnershipAcquisitionNotification()
   ACQUIRED ownership of attribute 'SimplesineStateAndParameters'->'Amplitude'
   of object 'simplesineStateAndParameters'.
...
```

Listing 9.2: Output stream from the active ownerhsip transfer simulation

On the other side, inspection of the (abbreviated) output shown below from the passive simulation reveals that it does indeed acquire ownership and then divest it later.

```
...
| |wormhole|1|4.00|2007/08/05,21:56:19| TrickHLAManager::receive_cyclic_data()
| |wormhole|1|4.00|2007/08/05,21:56:19| TrickHLAManager::send_cyclic_data()
| |wormhole|-1|4.25|2007/08/05,21:56:19| TrickHLAFedAmb::timeAdvanceGrant()
Federate "passive" Time granted to: 5
| |wormhole|1|5.00|2007/08/05,21:56:20| TrickHLAManager::receive_cyclic_data()
| |wormhole|1|5.00|2007/08/05,21:56:20| TrickHLAManager::send_cyclic_data()
| |wormhole|-1|5.25|2007/08/05,21:56:20| TrickHLAFedAmb::timeAdvanceGrant()
Federate "passive" Time granted to: 6
| |wormhole|-1|5.25|2007/08/05,21:56:20| TrickHLAFedAmb::requestAttributeOwnershipAssumption()
  push request received, tag='simplesineStateAndParameters'
| |wormhole|-1|5.25|2007/08/05,21:56:20| TrickHLAFedAmb::requestAttributeOwnershipAssumption()
...
| |wormhole|-1|6.25|2007/08/05,21:56:21| TrickHLAFedAmb::attributeOwnershipAcquisitionNotification()
   ACQUIRED ownership of attribute 'SimplesineStateAndParameters'->'Time'
   of object 'simplesineStateAndParameters'.
| |wormhole|-1|6.25|2007/08/05,21:56:21| TrickHLAFedAmb::attributeOwnershipAcquisitionNotification()
   ACQUIRED ownership of attribute 'SimplesineStateAndParameters'->'Value'
   of object 'simplesineStateAndParameters'.
| |wormhole|-1|6.25|2007/08/05,21:56:21| TrickHLAFedAmb::attributeOwnershipAcquisitionNotification()
   ACQUIRED ownership of attribute 'SimplesineStateAndParameters'->'dvdt'
   of object 'simplesineStateAndParameters'.
| |wormhole|-1|6.25|2007/08/05,21:56:21| TrickHLAFedAmb::attributeOwnershipAcquisitionNotification()
   ACQUIRED ownership of attribute 'SimplesineStateAndParameters'->'Phase'
   of object 'simplesineStateAndParameters'.
| |wormhole|-1|6.25|2007/08/05,21:56:21| TrickHLAFedAmb::attributeOwnershipAcquisitionNotification()
   ACQUIRED ownership of attribute 'SimplesineStateAndParameters'->'Frequency'
   of object 'simplesineStateAndParameters'.
| |wormhole|-1|6.25|2007/08/05,21:56:21| TrickHLAFedAmb::attributeOwnershipAcquisitionNotification()
   ACQUIRED ownership of attribute 'SimplesineStateAndParameters'->'Amplitude'
   of object 'simplesineStateAndParameters'.
| |wormhole|-1|6.25|2007/08/05,21:56:21| TrickHLAFedAmb::timeAdvanceGrant()
Federate "passive" Time granted to: 7
...
| |wormhole|-1|10.25|2007/08/05,21:56:25| TrickHLAFedAmb::requestAttributeOwnershipRelease()
  pull request received, tag='simplesineStateAndParameters'
| |wormhole|-1|10.25|2007/08/05,21:56:25| TrickHLAFedAmb::requestAttributeOwnershipRelease()
...
   DIVESTED Ownership for attribute 'SimplesineStateAndParameters'->'Time'
   of object 'simplesineStateAndParameters'.
| |wormhole|1|11.00|2007/08/05,21:56:26| TrickHLAObject::grant_pull_request()
   DIVESTED Ownership for attribute 'SimplesineStateAndParameters'->'Value'
   of object 'simplesineStateAndParameters'.
| |wormhole|1|11.00|2007/08/05,21:56:26| TrickHLAObject::grant_pull_request()
   DIVESTED Ownership for attribute 'SimplesineStateAndParameters'->'dvdt'
   of object 'simplesineStateAndParameters'.
| |wormhole|1|11.00|2007/08/05,21:56:26| TrickHLAObject::grant_pull_request()
   DIVESTED Ownership for attribute 'SimplesineStateAndParameters'->'Phase'
   of object 'simplesineStateAndParameters'.
| |wormhole|1|11.00|2007/08/05,21:56:26| TrickHLAObject::grant_pull_request()
   DIVESTED Ownership for attribute 'SimplesineStateAndParameters'->'Frequency'
   of object 'simplesineStateAndParameters'.
| |wormhole|1|11.00|2007/08/05,21:56:26| TrickHLAObject::grant_pull_request()
   DIVESTED Ownership for attribute 'SimplesineStateAndParameters'->'Amplitude'
   of object 'simplesineStateAndParameters'.
...
```

Listing 9.3: Output stream from the passive ownerhsip transfer simulation

This is also illustrated in the two following plots of the `.locally_owned` flag for the first attribute of the object.

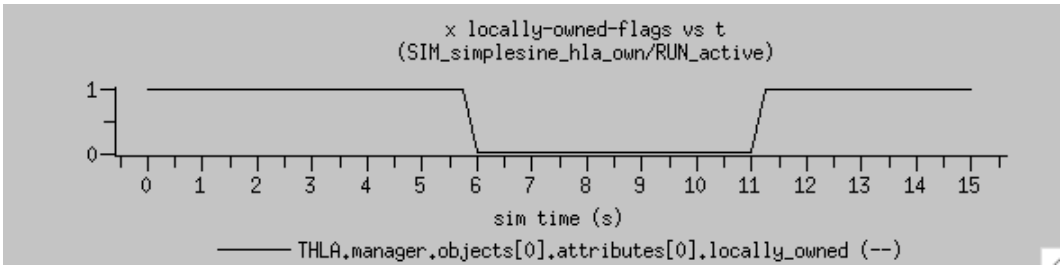

Figure 9.1: Output from the active `SIM_simplesine_hla_own` simulation

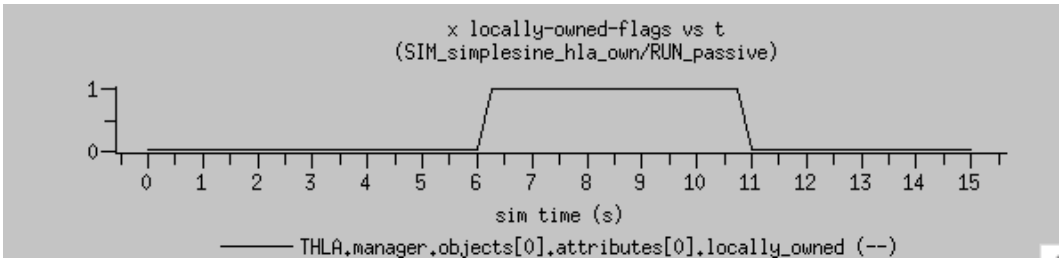

Figure 9.2: Output from the passive `SIM_simplesine_hla_own` simulation

Chapter 10

Data Encoding and Packing

`TrickHLA` provides a mechanism for simulation developers to modify updated class attribute data before sending it out to other federates or upon receiving new data from publishers. This capability might be necessary, for example, if the agreed-upon units for an attribute are different from those used internally by the simulation, in which case a developer could write *pack* and *unpack* logic to change units as data come into and leaves the simulation. This chapter illustrates how to do that.

10.1 What is a *packing* class?

`TrickHLA` provides two *hooks* that allow simulation developers to modify[1] data as it is sent out via HLA and as it arrives from HLA. This is implemented in the `pack()` and `unpack()` methods of the `TrickHLAPacking` class. The `pack()` method is automatically invoked by `TrickHLA` before data are sent via HLA. The `unpack()` method is automatically invoked by `TrickHLA` after data are received from HLA. These are virtual methods and must be overriden in a subclass in order to add application-specific packing and unpacking to the simulation.

10.1.1 `TrickHLAPacking`

The header file for the `TrickHLAPacking` class is shown below.

```
1  class TrickHLAObject;
2  #include "TrickHLA/include/TrickHLAObject.hh"

4  class TrickHLAPacking
5  {
6     friend class InputProcessor;
7     friend void init_attrTrickHLAPacking();

9    public:
10    TrickHLAPacking() {};
11    virtual ~TrickHLAPacking() {};
```

[1] What *modify* means is application specific. It might be encoding/decoding. It might be packing/unpacking. Or it might be changing units back and forth from FOM-agreed units (e.g., degrees) and application-specific units (e.g., radians).

```
13    TrickHLADebugHandler debug_handler; // -- Prints out multiple debug levels

15    TrickHLAAttribute * get_attribute( const char * attr_FOM_name );

17    virtual void initialize_callback( TrickHLAObject * obj );

19    //-----------------------------------------------------------------
20    // These are virtual functions and must be defined by a full class.
21    //-----------------------------------------------------------------

23    virtual void pack();

25    virtual void unpack();

27   protected:
28    TrickHLAObject * object; // ** Reference to the TrickHLA Object.
29 };
```

Listing 10.1: `TrickHLAPacking` class header

10.1.2 `simplesine_Packing`

In order to illustrate the use of the `TrickHLAPacking` class, the `simplesine` model has a packing class. The class header is shown below.

```
1  #include "TrickHLA/include/TrickHLAPacking.hh"
2  #include "simplesine/include/simplesine.h"

4  class simplesine_Packing : public TrickHLAPacking
5  {
6    friend class InputProcessor;
7    friend void init_attrsimplesine_Packing();

9    public:
10     simplesine_Packing();
11     virtual ~simplesine_Packing();

13     virtual void init(
14       simplesine_T* originalP,
15       simplesine_T* packedP,
16       simplesine_T* unpackedP );

18     virtual void pack();
19     virtual void unpack();

21   private:
22     bool is_initialized;
23     simplesine_T* originalP;
24     simplesine_T* packedP;
25     simplesine_T* unpackedP;
26 };
```

Listing 10.2: `simplesine_Packing` class header

This class is designed to work as follows. the `init()` method *must* be called in order to initialize the class (i.e., invoked as a Trick initialization job). The initialization method specifies three `simplesine` objects:

- `originalP` – The `simplesine` data used as input to the `pack()` and `unpack()` methods.

- `packedP` – The `simplesine` data used as output from the `pack()` method.
- `unpackedP` – The `simplesine` data used as output from the `unpack()` method.

The implementation of the methods is shown below. In this case, both methods just copy the data without doing any modification at all, which of course has no value other than illustrating (in the following simulations) how to use the `TrickHLAPacking` class.

```
1  /******************************** TRICK HEADER *****************************
2  PURPOSE: (implementation of packing/unpacking methods)
3  LIBRARY DEPENDENCY: ((simplesine_copy.o))
4  ***************************************************************************/
5  // System include files.
6  #include <math.h>
7  #include <stdlib.h>
8  #include <iostream>
9  #include <string>

11 // Trick include files.
12 #include "sim_services/include/exec_proto.h"

14 // TrickHLA model include files.
15 #include "TrickHLA/include/TrickHLAAttribute.hh"

17 // Model include files.
18 #include "../include/simplesine_Packing.hh"
19 #include "../include/simplesine_proto.h"

21 using namespace std;

23 /*-------------------------------------------------------------------------
24 PURPOSE: (default constructor for the simplesine packing/unpacking class.)
25 -------------------------------------------------------------------------*/
26 simplesine_Packing::simplesine_Packing() // RETURN: -- None.
27   : is_initialized(false),
28     originalP(NULL),
29     packedP(NULL),
30     unpackedP(NULL)
31 { }

33 /*-------------------------------------------------------------------------
34 PURPOSE: (destructor for the simplesine packing/unpacking class.)
35 -------------------------------------------------------------------------*/
36 simplesine_Packing::~simplesine_Packing() // RETURN: -- None.
37 { }

39 /*-------------------------------------------------------------------------
40 PURPOSE: (initialization method)
41 -------------------------------------------------------------------------*/
42 void simplesine_Packing::init( // RETURN: -- None.
43   simplesine_T* originalP,  // INOUT: -- where the data is coming from
44   simplesine_T* packedP,    // INOUT: -- where to pack it into
45   simplesine_T* unpackedP)  // INOUT: -- where to unpack it into
46 {
47   this->originalP = originalP;
48   this->packedP = packedP;
49   this->unpackedP = unpackedP;
50   this->is_initialized = true;
51 }

53 /*-------------------------------------------------------------------------
54 PURPOSE: (data packing method. This is called before data is sent to the RTI.)
55 -------------------------------------------------------------------------*/
56 void simplesine_Packing::pack() // RETURN: -- None.
```

```
57 {
58   if( ! this->is_initialized ) {
59     exec_terminate(
60       "simplesine_Packing.cpp",
61       "simplesine_Packing::pack() called on non-initialized object" );
62   }

64   send_hs( stdout, "pack(): packing data from %p into %p", originalP, packedP );
65   simplesine_copy( originalP, packedP );
66 }

68 /*-----------------------------------------------------------------------
69 PURPOSE: (data unpacking method. This is called after data is received
70   from the RTI.)
71 -----------------------------------------------------------------------*/
72 void simplesine_Packing::unpack() // RETURN: -- None.
73 {
74   if( ! this->is_initialized ) {
75     exec_terminate(
76       "simplesine_Packing.cpp",
77       "simplesine_Packing::unpack() called on non-initialized object" );
78   }

80   send_hs( stdout, "unpack(): unpacking data from %p into %p", originalP, unpackedP );
81   simplesine_copy( originalP, unpackedP );
82 }
```

Listing 10.3: `simplesine_Packing` class methods

10.2 SIM_simplesine_hla_pack

This `SIM_simplesine_hla_pack` simulation illustrates how to *pack* data from a publishing simulation. The simulation is based on the plain publisher, `SIM_simplesine_hla_pub`, from Chapter 6.

10.2.1 S_define file

This differences between the `S_define` for this simulation and that of the plain publisher are as follows.

- A new `simplesine` variable, `simplesine_packed`, is defined in the `publisher` sim object.
- The definition of an instance, `packer`, of the `simplesine_Packing` class.
- The invocation of `simplesine_Packing::init()` as a Trick initialization job, specifying `publisher.simplesine` as the input to the `pack()` method and `publisher.simplesine_packed` as the output.

10.2.2 Input file

The only difference between the input file for this simulation and that of the plain publisher is the following new line

```
THLA.manager.objects[0].packing = &publisher.packer;
```

which associates `publisher.packer` as the packing class for the object being published.

10.3 SIM_simplesine_hla_unpack

This `SIM_simplesine_hla_unpack` simulation illustrates how to *unpack* data from a subscribing simulation. The simulation is based on the plain subscriber, `SIM_simplesine_hla_sub`, from Chapter 6.

10.3.1 S_define file

This differences between the `S_define` for this simulation and that of the plain subscriber are as follows.

- A new `simplesine` variable, `simplesine_unpacked`, is defined in the `subscriber` sim object.
- The definition of an instance, `unpacker`, of the `simplesine_Packing` class.
- The invocation of `simplesine_Packing::init()` as a Trick initialization job, specifying `subscriber.simplesine` as the input to the `unpack()` method and `subscriber.simplesine_unpacked` as the output.

10.3.2 Input file

The only difference between the input file for this simulation and that of the plain subscriber is the following new line

```
THLA.manager.objects[0].packing = &subscriber.unpacker;
```

which associates `subscriber.unpacker` as the packing class for the object to which the simulation subscribes.

10.4 Output

The output of running the pack and unpack simulations shows issued by the `simplesine` `pack()` and `unpack()` methods.

```
...
| |wormhole|1|0.00|2007/08/05,23:39:14| TrickHLAManager::initialization_complete()
      Simulation has started and is now running...
| |wormhole|1|0.00|2007/08/05,23:39:14| TrickHLAManager::receive_cyclic_data()
| |wormhole|1|0.00|2007/08/05,23:39:14| TrickHLAManager::send_cyclic_data()
| |wormhole|1|0.00|2007/08/05,23:39:14| pack(): packing data from 0x99aaee4 into 0x99aaf1c
| |wormhole|-1|0.25|2007/08/05,23:39:14| TrickHLAFedAmb::timeAdvanceGrant()
```

```
Federate "publisher" Time granted to: 1
| |wormhole|1|1.00|2007/08/05,23:39:15| TrickHLAManager::receive_cyclic_data()
| |wormhole|1|1.00|2007/08/05,23:39:15| TrickHLAManager::send_cyclic_data()
| |wormhole|1|1.00|2007/08/05,23:39:15| pack(): packing data from 0x99aaee4 into 0x99aaf1c
| |wormhole|-1|1.25|2007/08/05,23:39:15| TrickHLAFedAmb::timeAdvanceGrant()
Federate "publisher" Time granted to: 2
| |wormhole|1|2.00|2007/08/05,23:39:16| TrickHLAManager::receive_cyclic_data()
| |wormhole|1|2.00|2007/08/05,23:39:16| TrickHLAManager::send_cyclic_data()
| |wormhole|1|2.00|2007/08/05,23:39:16| pack(): packing data from 0x99aaee4 into 0x99aaf1c
| |wormhole|-1|2.25|2007/08/05,23:39:16| TrickHLAFedAmb::timeAdvanceGrant()
Federate "publisher" Time granted to: 3
| |wormhole|1|3.00|2007/08/05,23:39:17| TrickHLAManager::receive_cyclic_data()
| |wormhole|1|3.00|2007/08/05,23:39:17| TrickHLAManager::send_cyclic_data()
| |wormhole|1|3.00|2007/08/05,23:39:17| pack(): packing data from 0x99aaee4 into 0x99aaf1c
| |wormhole|-1|3.25|2007/08/05,23:39:17| TrickHLAFedAmb::timeAdvanceGrant()
Federate "publisher" Time granted to: 4
...
```

Listing 10.4: Output from the packer simulation

```
...
| |wormhole|1|0.00|2007/08/05,23:39:14| TrickHLAManager::initialization_complete()
       Simulation has started and is now running...
| |wormhole|1|0.00|2007/08/05,23:39:14| TrickHLAManager::receive_cyclic_data()
| |wormhole|1|0.00|2007/08/05,23:39:14| TrickHLAManager::send_cyclic_data()
| |wormhole|1|0.00|2007/08/05,23:39:14| pack(): packing data from 0x99aaee4 into 0x99aaf1c
| |wormhole|-1|0.25|2007/08/05,23:39:14| TrickHLAFedAmb::timeAdvanceGrant()
Federate "publisher" Time granted to: 1
| |wormhole|1|1.00|2007/08/05,23:39:15| TrickHLAManager::receive_cyclic_data()
| |wormhole|1|1.00|2007/08/05,23:39:15| TrickHLAManager::send_cyclic_data()
| |wormhole|1|1.00|2007/08/05,23:39:15| pack(): packing data from 0x99aaee4 into 0x99aaf1c
| |wormhole|-1|1.25|2007/08/05,23:39:15| TrickHLAFedAmb::timeAdvanceGrant()
Federate "publisher" Time granted to: 2
| |wormhole|1|2.00|2007/08/05,23:39:16| TrickHLAManager::receive_cyclic_data()
| |wormhole|1|2.00|2007/08/05,23:39:16| TrickHLAManager::send_cyclic_data()
| |wormhole|1|2.00|2007/08/05,23:39:16| pack(): packing data from 0x99aaee4 into 0x99aaf1c
| |wormhole|-1|2.25|2007/08/05,23:39:16| TrickHLAFedAmb::timeAdvanceGrant()
Federate "publisher" Time granted to: 3
| |wormhole|1|3.00|2007/08/05,23:39:17| TrickHLAManager::receive_cyclic_data()
| |wormhole|1|3.00|2007/08/05,23:39:17| TrickHLAManager::send_cyclic_data()
| |wormhole|1|3.00|2007/08/05,23:39:17| pack(): packing data from 0x99aaee4 into 0x99aaf1c
| |wormhole|-1|3.25|2007/08/05,23:39:17| TrickHLAFedAmb::timeAdvanceGrant()
Federate "publisher" Time granted to: 4
...
```

Listing 10.5: Output from the unpacker simulation

Chapter 11

Initialization

As was discussed in Section 5, there is more to initializing an HLA federate using `TrickHLA` than just joining the federation. In addition to joining, `TrickHLA` **requires** that you follow the *multiphase initialization process*. This chapter illustrates how to do that.

11.1 What is multiphase initialization?

The initialization of the components in a distributed simulation can be more complex than simply letting each components initialize itself. Initialization dependencies between components might require that the components *partially* initialize themselves. The mechanism provided by `TrickHLA` for this called *multiphase initialization*. It works as follows.

The developers of a distributed simulation must agree beforehand on the number of *initialization phases* to be executed by each simulation during startup. Furthermore, each phase has a unique name. `TrickHLA` then provides a means for each simulation developer to schedule jobs during each phase of the initialization. In particular, data calculated during one phase may be shared with other federates for use during their subsequent initialization phases.

11.2 `DSESSimConfig`

The `TrickHLA` mechanism for doing this is part of the `THLA_INIT` sim object that was introduced in Chapter 5. In that and all the subsequent examples so far in this guide, the `THLA_INIT` object consisted only of a single `DSESSimConfig` variable. This variable is a fundamental part of the `TrickHLA` infrastructure, and indeed, `TrickHLA` will not function without the variable.

However, in order to participate in the multiphase initialization process, each simulation must also define the logic to be executed during each phase of the initialization. This is done by

- Setting the `THLA_INIT` variables according to how many phases there are, and
- adding phase-specific jobs to the `THLA_INIT` sim object.

11.2.1 THLA_INIT inputs for multiphase initialization

The TrickHLA implemenation of multiphase initialization is based on HLA *synchronization points*. Federates may use synchronization points as *barriers* where all must gather before any are allowed to continue. TrickHLA uses them in this fashion to make sure that all participating simulations have completed phase-i processing before any of the simulations proceed to phase-$i+1$.

Upon agreement among all the participants in the federation as to how many initialization phases are necessary and what each is named, the input file needs only include an entry for the federate object which specifies the name of each phase. For example, if a simulation requires two phases, named *Phase1* and *Phase2*, then the input file would include a line of the form

```
THLA.federate.multiphase_init_sync_points = "Phase1, Phase2";
```

Similarly, for three phases named A, B and C, the input file would include

```
THLA.federate.multiphase_init_sync_points = "A, B, C";
```

11.2.2 THLA_INIT jobs for multiphase initialization

The tasks to be carried out in each phase of initialization are

- Execute some application-specific initialization code.
- Share the results of that initialization code between the federates. (This involves sending locally-calculated intialization data out to other federates and receviing remotely-calculated initialization data from other federates.)
- Execute some application-specific post-initialization code.

Accordingly, each phase consists of Trick jobs which make calls to the following functions.

- `THLA.manager.send_init_data()`
- `THLA.manager.receive_init_data()`
- `THLA.manager.wait_for_init_sync_point()`

The arguments to the `send_init_data()` and `receive_init_data()` methods are the names of Trick variables defined in the `S_define` file. If there are many data to send, there will be correspondingly many jobs invoking `send_init_data()` and similarly for receiving data. The argument to the `wait_for_init_sync_point()` method is the name of the corresponding phase as defined in the sim config object. Invocation of the `wait_for_init_sync_point()` method tells TrickHLA that the simulation has completed the corresponding initialization phase. TrickHLA ensures that no simulations will proceed beyond this point until all the participating simulations have reached it.[1]

[1] This is implemented with HLA *synchronization points*, whence the name of the method.

This is illustrated below for two-phase initialization. Multiphase initialization is similar, except that there are more send/receive/wait-for sequences, one for each phase defined in the sim config object.

```
1   //-------------------------------------------------------------------------
2   // NOTE: Initialization phase numbers must be greater than P10 so that the
3   // initialization jobs run after the P10 THLA.manager.initialize() job.
4   //-------------------------------------------------------------------------

6   //
7   // PHASE 1
8   //
9   P100 (initialization) TrickHLA:
10     THLA.manager.send_init_data( In const char * obj_instance_name = "...name-of-trick-variable..." );
11  P100 (initialization) TrickHLA:
12     THLA.manager.receive_init_data( In const char * obj_instance_name = "...name-of-trick-variable..." );
13  // ...Add optional application-specific post-phase-1 processing here.
14  P100 (initialization) TrickHLA:
15     THLA.manager.wait_for_init_sync_point( In const char * syc_point_label = "...name-of-phase-1..." );

17  //
18  // PHASE 2
19  //
20  P200 (initialization) TrickHLA:
21     THLA.manager.send_init_data( In const char * obj_instance_name = "...name-of-trick-variable..." );
22  P200 (initialization) TrickHLA:
23     THLA.manager.receive_init_data( In const char * obj_instance_name = "...name-of-trick-variable..." );
24  // ...Add optional application-specific post-phase-1 processing here.
25  P200 (initialization) TrickHLA:
26     THLA.manager.wait_for_init_sync_point( In const char * syc_point_label = "...name-of-phase-2..." );
```

Listing 11.1: Multiphase initialization jobs

11.2.3 THLA_INIT jobs for one-phase initialization

If true multiphase initialization is overkill for your simulation, it is possible to define THLA_INIT jobs that just carry out traditional single-step initalization in a slightly simpler way than going through the steps described above.

The tasks to be carried out in single-step initialization are

- Execute some application-specific initialization code.
- Share the results of that initialization code between the federates.
- Execute some application-specific post-initialization code.

To do this, the following jobs can be inserted into the THLA_INIT sim object.

```
2   // Application-specific initialization code should have been
3   // executed already in some previous Trick initialization job

5   // Tell TrickHLA to "share" initialization data.
6   P100 (initialization) TrickHLA: THLA.manager.send_init_data();
7   P100 (initialization) TrickHLA: THLA.manager.receive_init_data();
```

```
9   // ...Add application-specific post-initialization code here if necessary.

11  // Tell TrickHLA that we're finished with the initialization process.
12  P100 (initialization) TrickHLA: THLA.manager.clear_init_sync_points();
```

Listing 11.2: One-phase initialization jobs

Chapter 12

Timeline

TrickHLA provides a mechanism for the user to specify the scenario timeline for the simulation. TrickHLA needs access to the scenario timeline in order to coordinate freezing (pausing) the federation. The scenario timeline is the only timeline that can be counted on to be synchronized between all the federates.

12.1 What is the *TrickHLATimeline* class?

TrickHLA provides a TrickHLATimeline class with a get_time() method that is used for getting the current simulation scenario time. This is a virtual method and must be overridden by a derived class in order to add application-specific functionality to the simulation. If a secnario timeline is not specified by the user then TrickHLA will use the Trick simulation time as the default scenario timeline, which is only valid if all Federates are using Trick and start with the same simulation time.

12.1.1 TrickHLATimeline

The header file for the TrickHLATimeline class is shown below.

```
1  class TrickHLATimeline
2  {
3     friend class InputProcessor;
4     friend void init_attrTrickHLATimeline();
5
6    public:
7     TrickHLATimeline();
8     virtual ~TrickHLATimeline();
9    private:
10    TrickHLATimeline(const TrickHLATimeline & rhs);
11    TrickHLATimeline & operator=(const TrickHLATimeline & rhs);
12
13   public:
14    virtual double get_time(); // Returns a time in seconds, typically
15                               // Terrestrial Time (TT) for the Scenario Timeline.
16 };
```

Listing 12.1: TrickHLATimeline class header

12.1.2 TrickHLASimTimeline

In order to illustrate the use of the `TrickHLATimeline` class, we subclass it, as shown below.

```
1  #include "TrickHLA/include/TrickHLATimeline.hh"
2
3  class TrickHLASimTimeline : public TrickHLATimeline
4  {
5     friend class InputProcessor;
6     friend void init_attrTrickHLASimTimeline();
7
8    public:
9     TrickHLASimTimeline();       // default constructor
10    virtual ~TrickHLASimTimeline(); // destructor
11
12    virtual double get_time(); // RETURN: s Current simulation time in seconds to represent the scenario time.
13 };
```

Listing 12.2: `TrickHLASimTimeline` class header

We give the `get_time()` method something to do, as shown below.

```
1  /******************************** TRICK HEADER ****************************
2  PURPOSE: (TrickHLASimTimeline : This class represents the simulation timeline.)
3  LIBRARY DEPENDENCY: ((TrickHLASimTimeline.o))
4  *************************************************************************/
5  // Trick include files.
6  #if TRICK_VER >= 10
7  #  include "sim_services/Executive/include/Executive.hh"
8  #  include "sim_services/Executive/include/exec_proto.h"
9  #else
10    // Trick 07
11 #  include "sim_services/include/executive.h"
12 #  include "sim_services/include/exec_proto.h"
13 #endif
14
15 // TrickHLA include files.
16 #include "TrickHLA/include/TrickHLASimTimeline.hh"
17
18 /******************************** TRICK HEADER ****************************
19 PURPOSE: (TrickHLASimTimeline::TrickHLASimTimeline : Default constructor.)
20 *************************************************************************/
21 TrickHLASimTimeline::TrickHLASimTimeline() // RETURN: -- None.
22 { }
23
24 /******************************** TRICK HEADER ****************************
25 PURPOSE: (TrickHLASimTimeline::~TrickHLASimTimeline : Destructor.)
26 *************************************************************************/
27 TrickHLASimTimeline::~TrickHLASimTimeline() // RETURN: -- None.
28 { }
29
30 /******************************** TRICK HEADER ****************************
31 PURPOSE: (TrickHLASimTimeline::get_time() : Get the current simulation time.)
32 LIBRARY DEPENDENCY: ((TrickHLATimeline.o)(TrickHLASimTimeline.o))
33 *************************************************************************/
34 double TrickHLASimTimeline::get_time() // RETURN: -- Current simulation time in seconds to represent the scenario time.
35 {
36 #if TRICK_VER >= 10
37    return exec_get_sim_time();
38 #else
39    return exec_get_exec()->out.time;
40 #endif
41 }
```

Listing 12.3: `TrickHLASimTimeline` code

In this example, all the `get_time()` method does is just return the Trick simulation time.

12.2 `S_define` file

The `TrickHLASimTimeline` class is introduced into the simulation via the `S_define` file. There, you would need to add a new `TrickHLASimTimeline` object into one simulation object and in this example we add it to the THLA_INIT simulation object like the following:

```
TrickHLA: TrickHLASimTimeline  sim_timeline;
```

`TrickHLA` will call the get_time() function when it needs to get the current scenario time.

12.3 `input` file

You need to register the `TrickHLATimeline` object with the THLA federate by adding the following lines.

```
THLA.federate.scenario_timeline = &THLA_INIT.sim_timeline
```

The simulation scenario timeline is specified by the THLA_INIT.sim_timeline implementation.

Chapter 13

Object Deleted Notification

TrickHLA provides a mechanism to run user specified code when an object is deleted from the federation. This capability allows for user specified operation to be performed when an object is deleted.

13.1 What is an *ObjectDeleted* class?

TrickHLA provides a TrickHLAObjectDeleted class with a deleted() method that is used for notification of a deleted object. This is a virtual method and must be overridden by a derived class in order to add application-specific functionality to the simulation.

13.1.1 TrickHLAObjectDeleted

The header file for the TrickHLAObjectDeleted class is shown below.

```
1  class TrickHLAObject;
2  #include "TrickHLA/include/TrickHLAObject.hh"
3
4  class TrickHLAObjectDeleted
5  {
6     friend class InputProcessor;
7     friend void init_attrTrickHLAObjectDeleted();
8
9    public:
10    TrickHLAObjectDeleted() {};
11    virtual ~TrickHLAObjectDeleted() {};
12
13    virtual void deleted (  // RETURN: -- None.
14       TrickHLAObject * ) {}; // IN: -- Deleted object data.
15 };
```

Listing 13.1: TrickHLAObjectDeleted class header

13.1.2 simplesine_objectDeleted

In order to illustrate the use of the TrickHLAObjectDeleted class, we subclass it, as shown below.

```
#include "TrickHLA/include/TrickHLAObjectDeleted.hh"

class simplesine_objectDeleted : public TrickHLAObjectDeleted
{
   friend class InputProcessor;
   friend void init_attrsimplesine_objectDeleted();

  public:
   simplesine_objectDeleted();       // default constructor
   virtual ~simplesine_objectDeleted(); // destructor

   void deleted(          // RETURN: -- None.
        TrickHLAObject * ); // IN:  -- Deleted object.
};
```

Listing 13.2: `simplesine_objectDeleted` class header

We give the `deleted()` method something to do, as shown below.

```
/********************************* TRICK HEADER *******************************
PURPOSE: (simplesine_objectDeleted : Callback class the user writes to do
         something once the object has been deleted from the RTI.)
LIBRARY DEPENDENCY: ((simplesine_objectDeleted.o))
******************************************************************************/
// System include files.
#include <sstream>

// Trick include files.
#include "sim_services/include/exec_proto.h"

// TrickHLA model include files.

// Model include files.
#include "../include/simplesine_objectDeleted.hh"

using namespace std;

/********************************* TRICK HEADER *******************************
PURPOSE: (simplesine_objectDeleted::simplesine_objectDeleted : Default
         constructor.)
******************************************************************************/
simplesine_objectDeleted::simplesine_objectDeleted() // RETURN: -- None.
   : TrickHLAObjectDeleted()
{ }

/********************************* TRICK HEADER *******************************
PURPOSE: (simplesine_objectDeleted::~simplesine_objectDeleted : Destructor.)
******************************************************************************/
simplesine_objectDeleted::~simplesine_objectDeleted() // RETURN: -- None.
{ }

/********************************* TRICK HEADER *******************************
PURPOSE: (simplesine_objectDeleted::deleted : Callback routine implementation
         to report that this object has been deleted from the federation.)
LIBRARY DEPENDENCY: ((TrickHLAObject.o))
******************************************************************************/
void simplesine_objectDeleted::deleted( // RETURN: -- None.
   TrickHLAObject * obj)            // IN: -- Deleted object.
{
   ostringstream msg;
   msg << "object␣'" << obj->get_name()
       << "'␣resigned␣from␣the␣federation." << endl;
   send_hs( stdout, (char *) msg.str().c_str() );
}
```

Listing 13.3: `simplesine_objectDeleted` code

As you can see, all the `deleted()` method does is just echoes a message to the simulation window.

13.2 `S_define` file

The `simplesine_objectDeleted` class is introduced into the simulation via the `S_define` file. There, you would need to add a new `simplesine_objectDeleted` object into each `sim_object` to which you wish to add a callback, like the following:

```
simplesine: simplesine_objectDeleted      obj_deleted_callback;
```

Additionally, you need to add a scheduled job into `TrickHLA` simulation object, like the following:

```
(PROPAGATE_TIMESTEP, scheduled) TrickHLA: THLA.manager.process_deleted_objects();
```

thereby scheduling the `TrickHLA`'s manager to identify any newly deleted objects every data cycle. When an object has been deleted from the federation, the manager will trigger, only once, all registered callback methods `[deleted()]` (see the next section).

13.3 `input` file

You need to register the `simplesine_objectDeleted` object to the THLA manager by adding the following lines.

```
THLA.manager.objects[0].deleted = &subscriber.obj_deleted_callback;
. . .
THLA.manager.objects[1].deleted = &publisher.obj_deleted_callback;
```

These lines associate obj_deleted_callback as the callback code for the subscriber and publisher `sim_objects`, respectively.

13.4 Output

The following output sample shows the callback code in action.

```
1 ...
2 | |rat|1|0.00|2008/06/06,12:27:10| TrickHLAFedAmb::initialize()
3 | |rat|1|0.00|2008/06/06,12:27:10| TrickHLAFederate::initialize()
4 | |rat|1|0.00|2008/06/06,12:27:10| TrickHLAFederate::restart_initialization()
5 | |rat|1|0.00|2008/06/06,12:27:10| TrickHLAManager::IMSim_initialization_version_1()
6 | |rat|1|0.00|2008/06/06,12:27:10| TrickHLAManager::setup_all_ref_attributes()
```

```
| |rat|1|0.00|2008/06/06,12:27:10| TrickHLAManager::setup_object_ref_attributes()
...
| |rat|1|15.00|2008/06/06,12:27:26| object 'P-side-Federate.Test' resigned from the federation.
...
SIMULATION TERMINATED IN
  PROCESS: 1
  JOB/ROUTINE: 21/sim_services/mains/master.c
DIAGNOSTIC:
Simulation reached input termination time.

LAST JOB CALLED: THLA.THLA.federate.time_advance_request()
             TOTAL OVERRUNS:          0
PERCENTAGE REALTIME OVERRUNS:     0.000%


      SIMULATION START TIME:     0.000
       SIMULATION STOP TIME:    15.000
    SIMULATION ELAPSED TIME:    15.000
        ACTUAL ELAPSED TIME:    15.000
       ACTUAL CPU TIME USED:     4.847
   SIMULATION / ACTUAL TIME:     1.000
      SIMULATION / CPU TIME:     3.095
  ACTUAL INITIALIZATION TIME:    0.000
     INITIALIZATION CPU TIME:    0.674
*** DYNAMIC MEMORY USAGE ***
    CURRENT ALLOCATION SIZE: 1411651
      NUM OF CURRENT ALLOCS:   888
        MAX ALLOCATION SIZE: 1413600
          MAX NUM OF ALLOCS:   894
       TOTAL ALLOCATION SIZE: 1726197
         TOTAL NUM OF ALLOCS:  5991
```

Listing 13.4: output displayed on the console

Chapter 14

Upgrading Your Federate To Initiate a Federation Save

Before a Trick federate can save itself, some upgrades to the Trick model have to occur.

14.1 Trick `input` file update

The `input` file must specify a simulation initialization scheme of `THLA_MULTIPHASE_INIT_V2`, which represents the *IMSim Multiphase Initialization Design with Late Joiners, Rejoiners and Federation Save & Restore* [6] scheme.

```
/* Use a simulation initialization scheme that supports save and restore. */
THLA.manager.sim_initialization_scheme = THLA_MULTIPHASE_INIT_V2;
```

The user can specify where each checkpoint-ing and restore-ing Trick federate is to store and reload the checkpoint files in `TrickHLAFederate`'s `HLA_save_directory` variable in each input file. If the directory is not specified in the `input` file, it will be assigned by `TrickHLA` to the simulation's `RUN` directory.

This is done because the checkpoint file name will be the same for all Trick federates with no uniqueness identifiers added, i.e. something to distinguish one file from another in the same directory, so it is recommended that the `HLA_save_directory` specify the Trick simulation's RUN directory.

If the user wishes to have the checkpoint files saved in another location, all they have to do is specify the absolute path to the new location in the `THLA.federate.HLA_save_directory` variable in the input file.

14.2 `S_define` file updates

The updates needed in the `S_define` file necessary for `TrickHLA`'s save and restore capability are provided in the `TrickHLA sim_object` in the `THLA.sm` file, located in the `S_modules` subdirectory.

The specific lines of code are described in this section.

14.2.1 Data Declarations

All federates must be in `freeze` mode at the same logical time in order to perform a federation save. The same is true for a federation restore (although you can also perform a federation restore at startup). This is accomplished by sending a special "freeze" interaction to all federates. A freeze interaction handler is declared in the `DATA STRUCTURE DECLARATIONS` section:

```
TrickHLA: TrickHLAFreezeInteractionHandler freeze_ih;
```

When performing a federation save via the `start_federation_save()` call (see section 14.2.3 `Programmatic Save and Restore` below), the time (optionally) and filename used for the save must be specified, either in a trick model or in the input file. So a checkpoint_time and checkpoint_label variable are also declared:

```
double checkpoint_time;
char   checkpoint_label[256];
```

14.2.2 Interactive Save and Restore

The user may choose to freeze the federation by issuing a Trick freeze command, usually by clicking the `Freeze` button on the Trick Simulation Control Panel, or by an input file freeze command. The following job will send a freeze interaction when a Trick freeze is commanded:

```
/*
 * -- Coordinate federates going to freeze mode
 */
P65534 (THLA_DATA_CYCLE_TIME, THLA_CHECK_PAUSE_JOB_OFFSET, logging) TrickHLA: THLA.federate
```

Another job will assure that each federate freezes at the same logical time when the freeze interaction is received:

```
/*
 * -- Check to see if an interaction informed us that we are to
 *    FREEZE the sim before entering the next logical frame.
 */
P65534 (THLA_DATA_CYCLE_TIME, logging) TrickHLA: THLA.federate.check_freeze_time();
```

Once the federation is frozen by this means, the user can issue a Trick `Dump Chkpnt` or `Load Chkpnt` command via the Trick Simulation Control Panel to initiate the federation save or restore, respectively.

The following two jobs get the federation running again when the Trick Simulation Control Panel `Start` button is clicked:

```
/*
 * -- Coordinate federates going to run mode
 */
(freeze)   TrickHLA: THLA.federate.check_freeze();
(unfreeze) TrickHLA: THLA.federate.exit_freeze();
```

14.2.3 Programmatic Save and Restore

If the user wishes to trigger a federation save via a trick model or the input file, calling the `start_federation_save()` routine will send the freeze interaction and initiate the federation save when the freeze occurs (see section 15 `Federation Save` below).

```
/*
 * -- Initiate a federation save announcement if told to do so...
 */
P1 (0.0, environment) TrickHLA: THLA.manager.start_federation_save(
   In const char * file_name = THLA.checkpoint_label );

P1 (0.0, environment) TrickHLA: THLA.manager.start_federation_save_at_sim_time(
   In double freeze_sim_time = THLA.checkpoint_time,
   In const char * file_name = THLA.checkpoint_label );

P1 (0.0, environment) TrickHLA: THLA.manager.start_federation_save_at_scenario_time(
   In double freeze_scenario_time = THLA.checkpoint_time,
   In const char * file_name = THLA.checkpoint_label );
```

If a time value is not specified, or if the time specified has already past, the current federation time will be used to determine when the next opportunity is to perform the federation save.

The only `TrickHLA` provided means of performing a restore programmatically is by setting the manager's `restore_federation` flag (see section 16 `Federation Restore` below), in which case the restore occurs at startup. A time cannot be specified for performing a restore.

14.2.4 Save and Restore Jobs

When a federation save or restore has been initiated, the current `TrickHLA` implementation depends on asynchronous callbacks from the `RTI` to guide the federation from one stage to another until the federation save or restore is complete. `TrickHLA` accomplishes this using a checkpoint class job and freeze class job (for save), and a pre_load_checkpoint class job and freeze job (for restore).

```
/*
 * -- Perform the federate save (checkpoint) or restore in FREEZE mode...
 */
P1 (checkpoint)          TrickHLA: THLA.federate.setup_checkpoint();
(freeze)                 TrickHLA: THLA.federate.perform_checkpoint();
```

```
P1 (pre_load_checkpoint) TrickHLA: THLA.federate.setup_restore();
(freeze)                 TrickHLA: THLA.federate.perform_restore();
```

Chapter 15

Federation Save

A federation save must occur in `freeze` mode, initiated via a freeze interaction (see *IMSim Multi-phase Initialization Design with Late Joiners, Rejoiners and Federation Save & Restore* [6] chapter 14, for details.) Once a federate receives this interaction, or reaches the interaction frame if this is the federate that sent the freeze interaction, it will go into `freeze` mode at the bottom of the next execution frame.

If this is the federate that sent the interaction, `TrickHLA` registers a `FEDSAVE_v2` synchronization point which all federates must acknowledge, and the federation must synchronize on, before proceeding with the federation save process. This is to avoid a race condition in which the federate that sent the interaction would request the federation save and any federate not in the same state (i.e. still executing the frame and not in `freeze` mode) will emit an exception when going into the time advance state (via calling `wait_for_time_advance_grant()` routine) because the `RTI` is already in federation save mode. This would be a fatal error since the non-frozen federate's execution would timeout waiting for a time advance grant.

Each federate, once in `freeze` mode, shall achieve the `FEDSAVE_v2` synchronization point with the `RTI` and wait until signaled by the `RTI` to begin its save.

15.1 Interactive Save

Perhaps the most straightforward way to perform a federation save is via the Trick Simulation Control Panel. Simply click the `Freeze` button on a federate's simulation control panel, and a freeze interaction is sent so that all federates will freeze at the same time (usually one or two lookahead_time frames after the freeze click). Then click the `Dump Chkpnt` button to trigger the federation save. A window will pop up where you can enter the user file name for the checkpoint file to be dumped. Each federate will dump a file of the form:

```
<federation_name>_<user_file_name>
```

in the directory specified by `THLA.federate.HLA_save_directory`, which defaults to the `RUN` directory. Note that `TrickHLA` automatically prepends the federation name to the given user file name.

There is also another file created named `<federation_name>_<user_file_name>.running_feds` which is for `TrickHLA`'s internal use. All federates will dump these two files in their respective directories.

The `RTI` itself will also save its own relevant data in a separate directory, which should be transparent to the user, but is configurable (see the `RTI` documentation for more information).

Simply click the `Start` button on the simulation control panel when you are ready to continue execution.

IMPORTANT: Use the same federate's Trick simulation control panel when clicking `Freeze` and `Start`. Each federate may have its own control panel, but you must use the same control panel that you clicked `Freeze` on to then click `Start` on. If you use a different control panel to click the `Start` button, the simulation will most likely not be able to continue.

15.2 Programmatic Save

The `TrickHLAManager::start_federation_save()` routine can be used to initiate the federation save from any Trick federate. There are three flavors of this routine. The first version of the routine will send a freeze interaction to pause the simulation at the bottom of the next frame, which is the earliest time a coordinated federation save can occur:

```
void start_federation_save( const char * file_name ) ;
```

The second version of the routine will send a freeze interaction to pause the simulation at the bottom of the frame at the user-supplied simulation time, which could be a time later than the next frame:

```
void start_federation_save_at_sim_time( double freeze_sim_time,
                                        const char * file_name ) ;
```

The third version of the routine will send a freeze interaction to pause the simulation at the bottom of the frame at the user-supplied scenario time, which could be a time later than the next frame:

```
void start_federation_save_at_scenario_time( double freeze_scenario_time,
                                             const char * file_name ) ;
```

The following are examples of triggering the federation save at simulation time 8.0 via the Trick input file. Note that the freeze (and therefore the save) will occur a couple of lookahead_time frames after 8.0:

```
read=8.0
THLA.checkpoint_label = "checkpoint.8.000";
CALL THLA.THLA.manager.start_federation_save( THLA.checkpoint_label );
```

```
   -- or --

THLA.checkpoint_label = "checkpoint.8.000";
THLA.checkpoint_time = 8.0;
CALL THLA.THLA.manager.start_federation_save_at_sim_time(
                                  THLA.checkpoint_time, THLA.checkpoint_label );

   -- or --

BEGIN_EVENT checkpoint_1 {
  CONDITION: sys.exec.out.time == 8.0 ;
ACTION {
   THLA.checkpoint_label = "checkpoint.8.000";
   CALL THLA.THLA.manager.start_federation_save( THLA.checkpoint_label );
   }
};
```

The location and name of the files created are the same as described in the above section 15.1 `Interactive Save`.

NOTE: You do not have to specify the federation name prepended to the checkpoint file name. `TrickHLA` will automatically prepend the federation name to the file name you supply if need be. So in the above example, if the federation name is `MyFederation`, then the file that will be saved is `MyFederation_checkpoint.8.000` in the `RUN` directory.

By default, after the save has taken place each federate simulation will remain in `freeze` mode until commanded by the user to go to run (e.g. via the Trick simulation control panel). If this is not the desired behavior, then set the federate `unfreeze_after_save` flag in each federate's Trick input file:

```
THLA.federate.unfreeze_after_save = 1 ;
```

Setting this flag for all federates will cause the federation to resume execution immediately after the save has completed.

Chapter 16

Federation Restore

A federation restore must occur in `freeze` mode (see section 16.1 `Interactive Restore` below) or it must occur at simulation startup (see section 16.2 `Programmatic Restore` below).

16.1 Interactive Restore

Perhaps the most straightforward way to perform a federation restore is via the Trick Simulation Control Panel. Simply click the `Freeze` button on a federate's simulation control panel, and a freeze interaction is sent so that all federates will freeze at the same time (usually one or two lookahead_time frames after the freeze click). Then click the `Load Chkpnt` button to trigger the federation restore. A window will pop up where you can select an existing checkpoint file to be loaded.

See section 15 `Federaton Save` above for how to dump a checkpoint file. Valid checkpoint file names to load will be of the form:

```
<federation_name>_<user_file_name>
```

Each federate will load its own checkpoint file using the chosen file name. Simply click the `Start` button on the simulation control panel when you are ready to continue execution.

IMPORTANT: Use the same federate's Trick simulation control panel when clicking `Freeze` and `Start`. Each federate may have its own control panel, but you must use the same control panel that you clicked `Freeze` on to then click `Start` on. If you use a different control panel to click the `Start` button, the simulation will most likely not be able to continue.

16.2 Programmatic Restore

In the current `TrickHLA` implementation, a programmatic federation restore can be initiated ONLY from the first federate which creates the federation (the `master`), and the restore will occur at simulation startup. Since the only way to trigger a federation restore in this manner is via the

input file at the startup of the federation, you must provide the name of the file to restore from in each input file and set the `restore_federation` flag to true.

Note that only the `master` federate needs to have the `restore_federation` flag set in the input file, but setting it in every federate's input file is a way to ensure that the federation is restored at startup no matter which federate is the `master`.

The only other thing to set is the `THLA.federate.HLA_save_directory`, which defaults to the `RUN` directory if left unset.

The following is an example of triggering the federation restore at startup time via the Trick input file:

```
THLA.manager.restore_federation = 1;
THLA.manager.restore_file_name = "checkpoint.15.000";
```

NOTE: You do not have to specify the federation name (prepended to the checkpoint file name) in the `restore_file_name`. `TrickHLA` will automatically prepend the federation name to the file name you supply if need be. So in the above example, if the federation name is `MyFederation`, then the file that will be loaded is `MyFederation_checkpoint.15.000` from the `RUN` directory.

Chapter 17

Conditional sending of cyclic data

This section illustrates how to use `TrickHLA` to conditionally send the simulation's cyclic attributes. The example `SIM_sine_conditional_data` shows how the user can write a simulation to determine when to send the attributes, for example: when an attribute changes.

17.1 How do you send cyclic data conditionally?

17.1.1 The `TrickHLAConditional` class

`TrickHLA` defines a C++ class must be subclassed by simulation developers in order to identify the conditions when to send cyclic data. Unless this class is subclassed, the default return is true so that the data is always sent across the wire on each cycle.

The class header is shown below. There is only a lone virtual method in this class, called `should_send()`, which is tied to its supplied `TrickHLAAttribute`. The user must subclass the `TrickHLAConditional` class in order to override this method, supplying code to the overridden `should_send()` method to determine when to send the attribute across the wire. The user is free to determine when the correct condition has (or conditions have) been met so that the attribute is sent across the wire.

The user is responible for updating the simulation's `S_define` file to declare a `TrickHLAConditional` class for each attribute which they wish to send conditionally as well as tieing each attribute and its corresponding conditional object in the input file.

```
1  class TrickHLAAttribute;
2  #include "TrickHLA/include/TrickHLAAttribute.hh"

4  class TrickHLAConditional
5  {
6       friend class InputProcessor;
7       friend void init_attrTrickHLAConditional();

9    public:
10      TrickHLAConditional(); // default constructor
11      virtual ~TrickHLAConditional(); // destructor

13      virtual bool should_send( TrickHLAAttribute * attr );
14 };
```

Listing 17.1: The `TrickHLAConditional` class

17.1.2 Subclassing TrickHLAConditional in the `SIM_sine_conditional_cyclic` example

The class header for the `SineConditional` class is shown below.

```
1  #include "TrickHLA/include/TrickHLAConditional.hh"
2
3  #include "SineData.hh"
4
5  class SineConditional : public TrickHLAConditional
6  {
7       friend class InputProcessor;
8       friend void init_attrSineConditional();
9
10   public:
11      SineConditional(); // default constructor
12      virtual ~SineConditional(); // destructor
13
14      void initialize( SineData *, const char * );
15
16      virtual bool should_send( TrickHLAAttribute * attr );
17
18   private:
19      int       convert_FOM_name_to_pos( const char * );
20
21      SineData * sim_data;    // -- pointer to the data to reflect in this cycle
22      SineData  prev_sim_data; // -- copy of the data we previously reflected
23
24      int       attr_pos;     // -- attribute position in SineData
25 };
```

Listing 17.2: `SineConditional` header file

And the code is shown below. The `initialize()` method specifies the simulation data collection and the attribute name whose value is checked before it is sent over the wire.

And the `should_send()` method compares the value of the attribute from the previous and the current calls of the routine, returning true if the value has changed (see the code below).

```
1  /******************************** TRICK HEADER ****************************
2  PURPOSE: (Define the conditions when to send cyclic data to other federates.)
3  *************************************************************************/
4  #include <stdlib.h>
5  #include <iostream>
6  #include <string>
7
8  #include "sim_services/include/exec_proto.h"
9
10 #include "../include/SineConditional.hh"
11
12 using namespace std;
13
14 /******************************** TRICK HEADER ****************************
15 PURPOSE: (SineConditional::SineHLAConditional : Default constructor.)
16 *************************************************************************/
17 SineConditional::SineConditional() // RETURN: -- None.
```

```
18    : TrickHLAConditional(),
19      sim_data(NULL),
20      attr_pos(-1)
21  { }

23  /******************************** TRICK HEADER ****************************
24  PURPOSE: (SineConditional::~SineConditional : Frees memory allocated for the
25           class.)
26  ******************************************************************************/
27  SineConditional::~SineConditional() // RETURN: -- None.
28  { }

30  /******************************** TRICK HEADER ****************************
31  PURPOSE: (SineConditional::initialize : Initializes the sim_data to the
32           supplied.)
33  ******************************************************************************/
34  void SineConditional::initialize( // RETURN: -- None.
35     SineData * data,              // IN: -- external simulation data.
36     const char * attr_FOM_name ) // IN: -- FOM name of the attribute to track when changed
37  {
38     sim_data = data;

40     attr_pos = convert_FOM_name_to_pos( attr_FOM_name );

42     // make a copy of the incoming data so that we have something to compare to
43     // when it comes time to compare (especially SineData's 'name', which is a
44     // char *).
45     prev_sim_data = *data;
46  }

48  /******************************** TRICK HEADER ****************************
49  PURPOSE: (SineConditional::should_send() : Determines if the attribute has
50           changed and returns the truth of that determination.)
51  ******************************************************************************/
52  bool SineConditional::should_send( // RETURN: -- None.
53     TrickHLAAttribute* attr )    // IN: ** Attribute to send
54  {
55     bool rc = false; // if there is no data or wrong attribute, send nothing!

57     // if there is simulation data to compare to, if the attribute FOM name has
58     // been specified and if the specified attribute position matches the
59     // supplied attribute's position, check the value of the current simulation
60     // variable versus the previous value. return true if there was a change.
61     if ( ( sim_data != NULL ) && ( attr_pos != -1 ) &&
62          ( convert_FOM_name_to_pos( attr->get_FOM_name() ) == attr_pos ) ) {

64        switch ( attr_pos ) {
65           case 0: // "Time"
66              if ( sim_data->get_time() != prev_sim_data.get_time() ) {
67                 rc = true;
68              }
69           break;
70           case 1: // "Value"
71              if ( sim_data->get_value() != prev_sim_data.get_value() ) {
72                 rc = true;
73              }
74           break;
75           case 2: // "dvdt"
76              if ( sim_data->get_derivative() != prev_sim_data.get_derivative() ) {
77                 rc = true;
78              }
79           break;
80           case 3: // "Phase"
81              if ( sim_data->get_phase() != prev_sim_data.get_phase() ) {
82                 rc = true;
83              }
```

```
84          break;
85          case 4: // "Frequency"
86             if ( sim_data->get_frequency() != prev_sim_data.get_frequency() ) {
87                rc = true;
88             }
89          break;
90          case 5: // "Amplitude"
91             if ( sim_data->get_amplitude() != prev_sim_data.get_amplitude() ) {
92                rc = true;
93             }
94          break;
95          case 6: // "Tolerance"
96             if ( sim_data->get_tolerance() != prev_sim_data.get_tolerance() ) {
97                rc = true;
98             }
99          break;
100         case 7: // "Name"
101            if ( strcmp( sim_data->get_name(), prev_sim_data.get_name() ) != 0 ) {
102               rc = true;
103            }
104         break;
105      };

107      prev_sim_data = *sim_data; // make a copy of the current data
108   } else {
109      send_hs( stderr, "SineConditional::should_send() => ERROR: either you \
110 forgot to call the initialize() routine to specify the attribute FOM name from \
111 the sim_data you wish to track or you provided the wrong TrickHLAAttribute to \
112 an already-initialized SineConditional!" );
113   }
114   return rc;
115 }

117 /******************************** TRICK HEADER *****************************
118 PURPOSE: (SineConditional::convert_FOM_name_to_pos() : Determines the supplied
119     name's position in the SineData structure. If a match does not exist or an
120     empty string was supplied, -1 is returned.)
121 ***************************************************************************/
122 int SineConditional::convert_FOM_name_to_pos( // RETURN: -- position in SineData
123    const char * attr_FOM_name )           // IN: -- FOM name of the attribute
124 {
125    string attr_name = attr_FOM_name;

127    if ( ! attr_name.empty() ) {
128       // speed up the code by NOT using string compares, which are very costly!
129       // instead, compare the first character of the attribute. since there is
130       // only one overlapping first character, this should be a very fast
131       // algorithm...
132       char first = attr_name[0];
133       char second = attr_name[1];

135       switch ( first ) {
136          case 'A':
137             return 5; // "Amplitude"
138          break;
139          case 'F':
140             return 4; // "Frequency"
141          break;
142          case 'N':
143             return 7; // "Name"
144          break;
145          case 'P':
146             return 3; // "Phase"
147          break;
148          case 'T':
149            if ( second == 'i' ) {
```

```
150             return 0; // "Time"
151           } else {
152             return 6; // "Tolerance"
153           }
154         break;
155         case 'V':
156           return 1; // "Value"
157         break;
158         case 'd':
159           return 2; // "dvdt"
160         break;
161       }
162     }

164     return -1;
165 }
```

Listing 17.3: `SineConditional` code

17.2 SIM_sine_conditional_cyclic

This simulation is based on the `SIM_sine` simulation, upgraded to send each one of the SineData's variables conditionally.

To accomplish this, you must do these two things to the `S_define` file for each of the two `sim_objects` in the `SIM_sine_conditional_cyclic` simulation.

- Define an instance of `SineConditional` for each attribute you wish to make conditional, and
- Specify a initialization class job to wire a data element of the SineData structure to the corresponding instance of SineConditional.

```
sim_object {
  ...
 sine: SineConditional     conditional_time;
 sine: SineConditional     conditional_value;
 sine: SineConditional     conditional_derivative;
 sine: SineConditional     conditional_phase;
 sine: SineConditional     conditional_frequency;
 sine: SineConditional     conditional_amplitude;
 sine: SineConditional     conditional_tolerance;
 sine: SineConditional     conditional_name;
  ...
 /* ------------------------ */
 /* -- INITIALIZATION JOBS -- */
 /* ------------------------ */
  ...
 P50 (initialization) sine: A.conditional_time.initialize(
    In SineData  * sim_data   = &A.sim_data,
    In const char * name      = "Time" );
 P50 (initialization) sine: A.conditional_value.initialize(
    In SineData  * sim_data   = &A.sim_data,
    In const char * name      = "Value" );
 P50 (initialization) sine: A.conditional_derivative.initialize(
    In SineData  * sim_data   = &A.sim_data,
    In const char * name      = "dvdt" );
 P50 (initialization) sine: A.conditional_phase.initialize(
```

```
      In SineData  * sim_data   = &A.sim_data,
      In const char * name     = "Phase" );
   P50 (initialization) sine: A.conditional_frequency.initialize(
      In SineData  * sim_data   = &A.sim_data,
      In const char * name     = "Frequency" );
   P50 (initialization) sine: A.conditional_amplitude.initialize(
      In SineData  * sim_data   = &A.sim_data,
      In const char * name     = "Amplitude" );
   P50 (initialization) sine: A.conditional_tolerance.initialize(
      In SineData  * sim_data   = &A.sim_data,
      In const char * name     = "Tolerance" );
   P50 (initialization) sine: A.conditional_name.initialize(
      In SineData  * sim_data   = &A.sim_data,
      In const char * name     = "Name" );
    ...
} A; // don't forget to update sim_object 'P'
```

Listing 17.4: Conditional `S_define` changes

17.3 Input

The input file is based on the original `SIM_sine`'s input file. It differs in that each `sim_object`'s attribute's `conditional` object is filled in for each attribute which is to be sent conditionally with the corresponding `sim_object`'s conditional object updated in listing 17.4, as detailed in the `THLA.manager.objects[0].attributes[`x`].conditional` lines in the listing below.

```
    ...
// Show or hide the TrickHLA debug messages.
THLA.manager.debug_handler.debug_level = THLA_LEVEL7_TRACE; // prints attribute names that are sent across the wire...
    ...
// The Federate has two objects, it publishes one and subscribes to another.
THLA.manager.obj_count = 2;
THLA.manager.objects = alloc(THLA.manager.obj_count);

// Configure the object this federate owns and will publish.
THLA.manager.objects[0].FOM_name        = "Test";
THLA.manager.objects[0].name            = "A-side-Federate.Test";
THLA.manager.objects[0].create_HLA_instance = true;
THLA.manager.objects[0].packing         = &A.packing;
THLA.manager.objects[0].ownership       = &A.ownership_handler;
THLA.manager.objects[0].deleted         = &A.obj_deleted_callback;
THLA.manager.objects[0].attr_count      = 8;
THLA.manager.objects[0].attributes      = alloc(THLA.manager.objects[0].attr_count);

THLA.manager.objects[0].attributes[0].FOM_name   = "Time";
THLA.manager.objects[0].attributes[0].trick_name = "A.sim_data.time";
THLA.manager.objects[0].attributes[0].config     = THLA_CYCLIC;
THLA.manager.objects[0].attributes[0].publish    = true;
THLA.manager.objects[0].attributes[0].locally_owned = true;
THLA.manager.objects[0].attributes[0].preferred_order = THLA_PREFERRED_ORDER;
THLA.manager.objects[0].attributes[0].rti_encoding = THLA_LITTLE_ENDIAN;
THLA.manager.objects[0].attributes[0].conditional = &A.conditional_time;

THLA.manager.objects[0].attributes[1].FOM_name   = "Value";
THLA.manager.objects[0].attributes[1].trick_name = "A.sim_data.value";
THLA.manager.objects[0].attributes[1].config     = THLA_INITIALIZE + THLA_CYCLIC;
THLA.manager.objects[0].attributes[1].publish    = true;
THLA.manager.objects[0].attributes[1].locally_owned = true;
THLA.manager.objects[0].attributes[1].preferred_order = THLA_PREFERRED_ORDER;
THLA.manager.objects[0].attributes[1].rti_encoding = THLA_LITTLE_ENDIAN;
```

```
THLA.manager.objects[0].attributes[1].conditional = &A.conditional_value;

THLA.manager.objects[0].attributes[2].FOM_name   = "dvdt";
THLA.manager.objects[0].attributes[2].trick_name = "A.sim_data.dvdt";
THLA.manager.objects[0].attributes[2].config     = THLA_CYCLIC;
THLA.manager.objects[0].attributes[2].publish    = true;
THLA.manager.objects[0].attributes[2].locally_owned = true;
THLA.manager.objects[0].attributes[2].preferred_order = THLA_PREFERRED_ORDER;
THLA.manager.objects[0].attributes[2].rti_encoding = THLA_LITTLE_ENDIAN;
THLA.manager.objects[0].attributes[2].conditional = &A.conditional_derivative;

THLA.manager.objects[0].attributes[3].FOM_name   = "Phase";
THLA.manager.objects[0].attributes[3].trick_name = "A.packing.phase_deg"; // using packed data instead of "A.sim_data.phase";
THLA.manager.objects[0].attributes[3].config     = THLA_CYCLIC;
THLA.manager.objects[0].attributes[3].publish    = true;
THLA.manager.objects[0].attributes[3].locally_owned = true;
THLA.manager.objects[0].attributes[3].preferred_order = THLA_PREFERRED_ORDER;
THLA.manager.objects[0].attributes[3].rti_encoding = THLA_LITTLE_ENDIAN;
THLA.manager.objects[0].attributes[3].conditional = &A.conditional_phase;

THLA.manager.objects[0].attributes[4].FOM_name   = "Frequency";
THLA.manager.objects[0].attributes[4].trick_name = "A.sim_data.freq";
THLA.manager.objects[0].attributes[4].config     = THLA_CYCLIC;
THLA.manager.objects[0].attributes[4].publish    = true;
THLA.manager.objects[0].attributes[4].locally_owned = true;
THLA.manager.objects[0].attributes[4].preferred_order = THLA_PREFERRED_ORDER;
THLA.manager.objects[0].attributes[4].rti_encoding = THLA_LITTLE_ENDIAN;
THLA.manager.objects[0].attributes[4].conditional = &A.conditional_frequency;

THLA.manager.objects[0].attributes[5].FOM_name   = "Amplitude";
THLA.manager.objects[0].attributes[5].trick_name = "A.sim_data.amp";
THLA.manager.objects[0].attributes[5].config     = THLA_CYCLIC;
THLA.manager.objects[0].attributes[5].publish    = true;
THLA.manager.objects[0].attributes[5].locally_owned = true;
THLA.manager.objects[0].attributes[5].preferred_order = THLA_PREFERRED_ORDER;
THLA.manager.objects[0].attributes[5].rti_encoding = THLA_LITTLE_ENDIAN;
THLA.manager.objects[0].attributes[5].conditional = &A.conditional_amplitude;

THLA.manager.objects[0].attributes[6].FOM_name   = "Tolerance";
THLA.manager.objects[0].attributes[6].trick_name = "A.sim_data.tol";
THLA.manager.objects[0].attributes[6].config     = THLA_CYCLIC;
THLA.manager.objects[0].attributes[6].publish    = true;
THLA.manager.objects[0].attributes[6].locally_owned = true;
THLA.manager.objects[0].attributes[6].preferred_order = THLA_PREFERRED_ORDER;
THLA.manager.objects[0].attributes[6].rti_encoding = THLA_LITTLE_ENDIAN;
THLA.manager.objects[0].attributes[6].conditional = &A.conditional_tolerance;

THLA.manager.objects[0].attributes[7].FOM_name   = "Name";
THLA.manager.objects[0].attributes[7].trick_name = "A.sim_data.name";
THLA.manager.objects[0].attributes[7].config     = THLA_INITIALIZE + THLA_CYCLIC;
THLA.manager.objects[0].attributes[7].publish    = true;
THLA.manager.objects[0].attributes[7].locally_owned = true;
THLA.manager.objects[0].attributes[7].preferred_order = THLA_PREFERRED_ORDER;
THLA.manager.objects[0].attributes[7].rti_encoding = THLA_UNICODE_STRING;
THLA.manager.objects[0].attributes[7].conditional = &A.conditional_name;
   ...
```

Listing 17.5: Conditional `input` file changes

17.4 Output

Output from the simulation shown below reveals which attributes are sent only because they changed, rather than all of the eight attributes.

```
   ...
| |wormhole|1|0.00|2010/02/03,17:44:44| TrickHLAManager::initialization_complete()
       Simulation has started and is now running...
| |wormhole|1|0.00|2010/02/03,17:44:44| TrickHLAFederate::wait_for_time_advance_grant() waiting for time advance grant (TAG)
| |wormhole|1|0.00|2010/02/03,17:44:44| TrickHLAFederate::wait_for_time_advance_grant() Time granted to 0 seconds.
| |wormhole|1|0.00|2010/02/03,17:44:44| TrickHLAManager::receive_cyclic_data()
| |wormhole|1|0.00|2010/02/03,17:44:44| TrickHLAManager::send_cyclic_data()
| |wormhole|1|0.00|2010/02/03,17:44:44| TrickHLAManager::send_requested_data()
| |wormhole|1|0.00|2010/02/03,17:44:44| TrickHLAFederate::time_advance_request()
requesting time advance grant (TAG) to 0.250000
| |wormhole|1|0.25|2010/02/03,17:44:45| TrickHLAFederate::wait_for_time_advance_grant() waiting for time advance grant (TAG)
| |wormhole|1|0.25|2010/02/03,17:44:45| TrickHLAFederate::wait_for_time_advance_grant() Time granted to 0.25 seconds.
| |wormhole|1|0.25|2010/02/03,17:44:45| TrickHLAManager::receive_cyclic_data()
| |wormhole|1|0.25|2010/02/03,17:44:45| TrickHLAManager::send_cyclic_data()
| |wormhole|1|0.25|2010/02/03,17:44:45| TrickHLAObject::create_attribute_set() -- adding 'Time' to attribute map
| |wormhole|1|0.25|2010/02/03,17:44:45| TrickHLAObject::create_attribute_set() -- adding 'Value' to attribute map
| |wormhole|1|0.25|2010/02/03,17:44:45| TrickHLAObject::create_attribute_set() -- adding 'dvdt' to attribute map
| |wormhole|1|0.25|2010/02/03,17:44:45| TrickHLAManager::send_requested_data()
| |wormhole|1|0.25|2010/02/03,17:44:45| TrickHLAFederate::time_advance_request()
requesting time advance grant (TAG) to 0.500000
| |wormhole|-1|0.30|2010/02/03,17:44:45| TrickHLAAttribute::extract_data() -- decoding 'Time' from attribute map
| |wormhole|-1|0.30|2010/02/03,17:44:45| TrickHLAAttribute::extract_data() -- decoding 'Value' from attribute map
| |wormhole|-1|0.30|2010/02/03,17:44:45| TrickHLAAttribute::extract_data() -- decoding 'dvdt' from attribute map
| |wormhole|1|0.50|2010/02/03,17:44:45| TrickHLAFederate::wait_for_time_advance_grant() waiting for time advance grant (TAG)
| |wormhole|1|0.50|2010/02/03,17:44:45| TrickHLAFederate::wait_for_time_advance_grant() Time granted to 0.5 seconds.
| |wormhole|1|0.50|2010/02/03,17:44:45| TrickHLAManager::receive_cyclic_data()
| |wormhole|1|0.50|2010/02/03,17:44:45| TrickHLAManager::send_cyclic_data()
| |wormhole|1|0.50|2010/02/03,17:44:45| TrickHLAObject::create_attribute_set() -- adding 'Time' to attribute map
| |wormhole|1|0.50|2010/02/03,17:44:45| TrickHLAObject::create_attribute_set() -- adding 'Value' to attribute map
| |wormhole|1|0.50|2010/02/03,17:44:45| TrickHLAObject::create_attribute_set() -- adding 'dvdt' to attribute map
| |wormhole|1|0.50|2010/02/03,17:44:45| TrickHLAManager::send_requested_data()
| |wormhole|1|0.50|2010/02/03,17:44:45| TrickHLAFederate::time_advance_request()
requesting time advance grant (TAG) to 0.750000
| |wormhole|-1|0.55|2010/02/03,17:44:45| TrickHLAAttribute::extract_data() -- decoding 'Time' from attribute map
| |wormhole|-1|0.55|2010/02/03,17:44:45| TrickHLAAttribute::extract_data() -- decoding 'Value' from attribute map
| |wormhole|-1|0.55|2010/02/03,17:44:45| TrickHLAAttribute::extract_data() -- decoding 'dvdt' from attribute map
| |wormhole|1|0.75|2010/02/03,17:44:45| TrickHLAFederate::wait_for_time_advance_grant() waiting for time advance grant (TAG)
| |wormhole|1|0.75|2010/02/03,17:44:45| TrickHLAFederate::wait_for_time_advance_grant() Time granted to 0.75 seconds.
| |wormhole|1|0.75|2010/02/03,17:44:45| TrickHLAManager::receive_cyclic_data()
| |wormhole|1|0.75|2010/02/03,17:44:45| TrickHLAManager::send_cyclic_data()
| |wormhole|1|0.75|2010/02/03,17:44:45| TrickHLAObject::create_attribute_set() -- adding 'Time' to attribute map
| |wormhole|1|0.75|2010/02/03,17:44:45| TrickHLAObject::create_attribute_set() -- adding 'Value' to attribute map
| |wormhole|1|0.75|2010/02/03,17:44:45| TrickHLAObject::create_attribute_set() -- adding 'dvdt' to attribute map
| |wormhole|1|0.75|2010/02/03,17:44:45| TrickHLAManager::send_requested_data()
| |wormhole|1|0.75|2010/02/03,17:44:45| TrickHLAFederate::time_advance_request()
requesting time advance grant (TAG) to 1.000000
| |wormhole|-1|0.80|2010/02/03,17:44:45| TrickHLAAttribute::extract_data() -- decoding 'Time' from attribute map
| |wormhole|-1|0.80|2010/02/03,17:44:45| TrickHLAAttribute::extract_data() -- decoding 'Value' from attribute map
| |wormhole|-1|0.80|2010/02/03,17:44:45| TrickHLAAttribute::extract_data() -- decoding 'dvdt' from attribute map
| |wormhole|1|1.00|2010/02/03,17:44:45| TrickHLAFederate::wait_for_time_advance_grant() waiting for time advance grant (TAG)
| |wormhole|1|1.00|2010/02/03,17:44:45| TrickHLAFederate::wait_for_time_advance_grant() Time granted to 1 seconds.
| |wormhole|1|1.00|2010/02/03,17:44:45| TrickHLAManager::receive_cyclic_data()
| |wormhole|1|1.00|2010/02/03,17:44:45| TrickHLAManager::send_cyclic_data()
| |wormhole|1|1.00|2010/02/03,17:44:45| TrickHLAObject::create_attribute_set() -- adding 'Time' to attribute map
| |wormhole|1|1.00|2010/02/03,17:44:45| TrickHLAObject::create_attribute_set() -- adding 'Value' to attribute map
| |wormhole|1|1.00|2010/02/03,17:44:45| TrickHLAObject::create_attribute_set() -- adding 'dvdt' to attribute map
| |wormhole|1|1.00|2010/02/03,17:44:45| TrickHLAManager::send_requested_data()
| |wormhole|1|1.00|2010/02/03,17:44:45| TrickHLAFederate::time_advance_request()
requesting time advance grant (TAG) to 1.250000
```

```
| |wormhole|-1|1.05|2010/02/03,17:44:45| TrickHLAAttribute::extract_data() -- decoding 'Time' from attribute map
| |wormhole|-1|1.05|2010/02/03,17:44:45| TrickHLAAttribute::extract_data() -- decoding 'Value' from attribute map
| |wormhole|-1|1.05|2010/02/03,17:44:45| TrickHLAAttribute::extract_data() -- decoding 'dvdt' from attribute map
| |wormhole|1|1.25|2010/02/03,17:44:46| TrickHLAFederate::wait_for_time_advance_grant() waiting for time advance grant (TAG)
| |wormhole|1|1.25|2010/02/03,17:44:46| TrickHLAFederate::wait_for_time_advance_grant() Time granted to 1.25 seconds.
| |wormhole|1|1.25|2010/02/03,17:44:46| TrickHLAManager::receive_cyclic_data()
| |wormhole|1|1.25|2010/02/03,17:44:46| TrickHLAManager::send_cyclic_data()
| |wormhole|1|1.25|2010/02/03,17:44:46| TrickHLAObject::create_attribute_set() -- adding 'Time' to attribute map
| |wormhole|1|1.25|2010/02/03,17:44:46| TrickHLAObject::create_attribute_set() -- adding 'Value' to attribute map
| |wormhole|1|1.25|2010/02/03,17:44:46| TrickHLAObject::create_attribute_set() -- adding 'dvdt' to attribute map
| |wormhole|1|1.25|2010/02/03,17:44:46| TrickHLAManager::send_requested_data()
| |wormhole|1|1.25|2010/02/03,17:44:46| TrickHLAFederate::time_advance_request()
requesting time advance grant (TAG) to 1.500000
| |wormhole|-1|1.30|2010/02/03,17:44:46| TrickHLAAttribute::extract_data() -- decoding 'Time' from attribute map
| |wormhole|-1|1.30|2010/02/03,17:44:46| TrickHLAAttribute::extract_data() -- decoding 'Value' from attribute map
| |wormhole|-1|1.30|2010/02/03,17:44:46| TrickHLAAttribute::extract_data() -- decoding 'dvdt' from attribute map
| |wormhole|1|1.50|2010/02/03,17:44:46| TrickHLAFederate::wait_for_time_advance_grant() waiting for time advance grant (TAG)
| |wormhole|1|1.50|2010/02/03,17:44:46| TrickHLAFederate::wait_for_time_advance_grant() Time granted to 1.5 seconds.
| |wormhole|1|1.50|2010/02/03,17:44:46| TrickHLAManager::receive_cyclic_data()
| |wormhole|1|1.50|2010/02/03,17:44:46| TrickHLAManager::send_cyclic_data()
| |wormhole|1|1.50|2010/02/03,17:44:46| TrickHLAObject::create_attribute_set() -- adding 'Time' to attribute map
| |wormhole|1|1.50|2010/02/03,17:44:46| TrickHLAObject::create_attribute_set() -- adding 'Value' to attribute map
| |wormhole|1|1.50|2010/02/03,17:44:46| TrickHLAObject::create_attribute_set() -- adding 'dvdt' to attribute map
| |wormhole|1|1.50|2010/02/03,17:44:46| TrickHLAManager::send_requested_data()
| |wormhole|1|1.50|2010/02/03,17:44:46| TrickHLAFederate::time_advance_request()
requesting time advance grant (TAG) to 1.750000
| |wormhole|-1|1.55|2010/02/03,17:44:46| TrickHLAAttribute::extract_data() -- decoding 'Time' from attribute map
| |wormhole|-1|1.55|2010/02/03,17:44:46| TrickHLAAttribute::extract_data() -- decoding 'Value' from attribute map
| |wormhole|-1|1.55|2010/02/03,17:44:46| TrickHLAAttribute::extract_data() -- decoding 'dvdt' from attribute map
| |wormhole|1|1.75|2010/02/03,17:44:46| TrickHLAFederate::wait_for_time_advance_grant() waiting for time advance grant (TAG)
| |wormhole|1|1.75|2010/02/03,17:44:46| TrickHLAFederate::wait_for_time_advance_grant() Time granted to 1.75 seconds.
| |wormhole|1|1.75|2010/02/03,17:44:46| TrickHLAManager::receive_cyclic_data()
| |wormhole|1|1.75|2010/02/03,17:44:46| TrickHLAManager::send_cyclic_data()
| |wormhole|1|1.75|2010/02/03,17:44:46| TrickHLAObject::create_attribute_set() -- adding 'Time' to attribute map
| |wormhole|1|1.75|2010/02/03,17:44:46| TrickHLAObject::create_attribute_set() -- adding 'Value' to attribute map
| |wormhole|1|1.75|2010/02/03,17:44:46| TrickHLAObject::create_attribute_set() -- adding 'dvdt' to attribute map
| |wormhole|1|1.75|2010/02/03,17:44:46| TrickHLAManager::send_requested_data()
| |wormhole|1|1.75|2010/02/03,17:44:46| TrickHLAFederate::time_advance_request()
requesting time advance grant (TAG) to 2.000000
| |wormhole|-1|1.80|2010/02/03,17:44:46| TrickHLAAttribute::extract_data() -- decoding 'Time' from attribute map
| |wormhole|-1|1.80|2010/02/03,17:44:46| TrickHLAAttribute::extract_data() -- decoding 'Value' from attribute map
| |wormhole|-1|1.80|2010/02/03,17:44:46| TrickHLAAttribute::extract_data() -- decoding 'dvdt' from attribute map
| |wormhole|1|2.00|2010/02/03,17:44:46| TrickHLAFederate::wait_for_time_advance_grant() waiting for time advance grant (TAG)
| |wormhole|1|2.00|2010/02/03,17:44:46| TrickHLAFederate::wait_for_time_advance_grant() Time granted to 2 seconds.
| |wormhole|1|2.00|2010/02/03,17:44:46| TrickHLAManager::receive_cyclic_data()
| |wormhole|1|2.00|2010/02/03,17:44:46| TrickHLAManager::send_cyclic_data()
| |wormhole|1|2.00|2010/02/03,17:44:46| TrickHLAObject::create_attribute_set() -- adding 'Time' to attribute map
| |wormhole|1|2.00|2010/02/03,17:44:46| TrickHLAObject::create_attribute_set() -- adding 'Value' to attribute map
| |wormhole|1|2.00|2010/02/03,17:44:46| TrickHLAObject::create_attribute_set() -- adding 'dvdt' to attribute map
| |wormhole|1|2.00|2010/02/03,17:44:46| TrickHLAManager::send_requested_data()
| |wormhole|1|2.00|2010/02/03,17:44:46| TrickHLAFederate::time_advance_request()
requesting time advance grant (TAG) to 2.250000
| |wormhole|-1|2.05|2010/02/03,17:44:46| TrickHLAAttribute::extract_data() -- decoding 'Time' from attribute map
| |wormhole|-1|2.05|2010/02/03,17:44:46| TrickHLAAttribute::extract_data() -- decoding 'Value' from attribute map
| |wormhole|-1|2.05|2010/02/03,17:44:46| TrickHLAAttribute::extract_data() -- decoding 'dvdt' from attribute map
| |wormhole|1|2.25|2010/02/03,17:44:47| TrickHLAFederate::wait_for_time_advance_grant() waiting for time advance grant (TAG)
| |wormhole|1|2.25|2010/02/03,17:44:47| TrickHLAFederate::wait_for_time_advance_grant() Time granted to 2.25 seconds.
| |wormhole|1|2.25|2010/02/03,17:44:47| TrickHLAManager::receive_cyclic_data()
| |wormhole|1|2.25|2010/02/03,17:44:47| TrickHLAManager::send_cyclic_data()
| |wormhole|1|2.25|2010/02/03,17:44:47| TrickHLAObject::create_attribute_set() -- adding 'Time' to attribute map
| |wormhole|1|2.25|2010/02/03,17:44:47| TrickHLAObject::create_attribute_set() -- adding 'Value' to attribute map
| |wormhole|1|2.25|2010/02/03,17:44:47| TrickHLAObject::create_attribute_set() -- adding 'dvdt' to attribute map
| |wormhole|1|2.25|2010/02/03,17:44:47| TrickHLAManager::send_requested_data()
| |wormhole|1|2.25|2010/02/03,17:44:47| TrickHLAFederate::time_advance_request()
requesting time advance grant (TAG) to 2.500000
| |wormhole|-1|2.30|2010/02/03,17:44:47| TrickHLAAttribute::extract_data() -- decoding 'Time' from attribute map
```

```
| |wormhole|-1|2.30|2010/02/03,17:44:47| TrickHLAAttribute::extract_data() -- decoding 'Value' from attribute map
| |wormhole|-1|2.30|2010/02/03,17:44:47| TrickHLAAttribute::extract_data() -- decoding 'dvdt' from attribute map
| |wormhole|1|2.50|2010/02/03,17:44:47| TrickHLAFederate::wait_for_time_advance_grant() waiting for time advance grant (TAG)
| |wormhole|1|2.50|2010/02/03,17:44:47| TrickHLAFederate::wait_for_time_advance_grant() Time granted to 2.5 seconds.
| |wormhole|1|2.50|2010/02/03,17:44:47| TrickHLAManager::receive_cyclic_data()
| |wormhole|1|2.50|2010/02/03,17:44:47| TrickHLAManager::send_cyclic_data()
| |wormhole|1|2.50|2010/02/03,17:44:47| TrickHLAObject::create_attribute_set() -- adding 'Time' to attribute map
| |wormhole|1|2.50|2010/02/03,17:44:47| TrickHLAObject::create_attribute_set() -- adding 'Value' to attribute map
| |wormhole|1|2.50|2010/02/03,17:44:47| TrickHLAObject::create_attribute_set() -- adding 'dvdt' to attribute map
| |wormhole|1|2.50|2010/02/03,17:44:47| TrickHLAManager::send_requested_data()
| |wormhole|1|2.50|2010/02/03,17:44:47| TrickHLAFederate::time_advance_request()
requesting time advance grant (TAG) to 2.750000
| |wormhole|-1|2.55|2010/02/03,17:44:47| TrickHLAAttribute::extract_data() -- decoding 'Time' from attribute map
| |wormhole|-1|2.55|2010/02/03,17:44:47| TrickHLAAttribute::extract_data() -- decoding 'Value' from attribute map
| |wormhole|-1|2.55|2010/02/03,17:44:47| TrickHLAAttribute::extract_data() -- decoding 'dvdt' from attribute map
| |wormhole|1|2.75|2010/02/03,17:44:47| TrickHLAFederate::wait_for_time_advance_grant() waiting for time advance grant (TAG)
| |wormhole|1|2.75|2010/02/03,17:44:47| TrickHLAFederate::wait_for_time_advance_grant() Time granted to 2.75 seconds.
| |wormhole|1|2.75|2010/02/03,17:44:47| TrickHLAManager::receive_cyclic_data()
| |wormhole|1|2.75|2010/02/03,17:44:47| TrickHLAManager::send_cyclic_data()
| |wormhole|1|2.75|2010/02/03,17:44:47| TrickHLAObject::create_attribute_set() -- adding 'Time' to attribute map
| |wormhole|1|2.75|2010/02/03,17:44:47| TrickHLAObject::create_attribute_set() -- adding 'Value' to attribute map
| |wormhole|1|2.75|2010/02/03,17:44:47| TrickHLAObject::create_attribute_set() -- adding 'dvdt' to attribute map
| |wormhole|1|2.75|2010/02/03,17:44:47| TrickHLAManager::send_requested_data()
| |wormhole|1|2.75|2010/02/03,17:44:47| TrickHLAFederate::time_advance_request()
requesting time advance grant (TAG) to 3.000000
| |wormhole|-1|2.80|2010/02/03,17:44:47| TrickHLAAttribute::extract_data() -- decoding 'Time' from attribute map
| |wormhole|-1|2.80|2010/02/03,17:44:47| TrickHLAAttribute::extract_data() -- decoding 'Value' from attribute map
| |wormhole|-1|2.80|2010/02/03,17:44:47| TrickHLAAttribute::extract_data() -- decoding 'dvdt' from attribute map
| |wormhole|1|3.00|2010/02/03,17:44:47| TrickHLAFederate::wait_for_time_advance_grant() waiting for time advance grant (TAG)
| |wormhole|1|3.00|2010/02/03,17:44:47| TrickHLAFederate::wait_for_time_advance_grant() Time granted to 3 seconds.
| |wormhole|1|3.00|2010/02/03,17:44:47| TrickHLAManager::receive_cyclic_data()
| |wormhole|1|3.00|2010/02/03,17:44:47| TrickHLAManager::send_cyclic_data()
| |wormhole|1|3.00|2010/02/03,17:44:47| TrickHLAObject::create_attribute_set() -- adding 'Time' to attribute map
| |wormhole|1|3.00|2010/02/03,17:44:47| TrickHLAObject::create_attribute_set() -- adding 'Value' to attribute map
| |wormhole|1|3.00|2010/02/03,17:44:47| TrickHLAObject::create_attribute_set() -- adding 'dvdt' to attribute map
| |wormhole|1|3.00|2010/02/03,17:44:47| TrickHLAManager::send_requested_data()
...
| |wormhole|1|13.75|2010/02/03,17:44:58| TrickHLAFederate::time_advance_request()
requesting time advance grant (TAG) to 14.000000
| |wormhole|1|14.00|2010/02/03,17:44:58| TrickHLAFederate::wait_for_time_advance_grant() waiting for time advance grant (TAG)
| |wormhole|1|14.00|2010/02/03,17:44:58| TrickHLAFederate::wait_for_time_advance_grant() Time granted to 14 seconds.
| |wormhole|1|14.00|2010/02/03,17:44:58| TrickHLAManager::receive_cyclic_data()
| |wormhole|1|14.00|2010/02/03,17:44:58| TrickHLAManager::send_cyclic_data()
| |wormhole|1|14.00|2010/02/03,17:44:58| TrickHLAObject::create_attribute_set() -- adding 'Time' to attribute map
| |wormhole|1|14.00|2010/02/03,17:44:58| TrickHLAObject::create_attribute_set() -- adding 'Value' to attribute map
| |wormhole|1|14.00|2010/02/03,17:44:58| TrickHLAObject::create_attribute_set() -- adding 'dvdt' to attribute map
| |wormhole|1|14.00|2010/02/03,17:44:58| TrickHLAManager::send_requested_data()
| |wormhole|1|14.00|2010/02/03,17:44:58| TrickHLAFederate::time_advance_request()
requesting time advance grant (TAG) to 14.250000
| |wormhole|-1|14.05|2010/02/03,17:44:58| TrickHLAAttribute::extract_data() -- decoding 'Time' from attribute map
| |wormhole|-1|14.05|2010/02/03,17:44:58| TrickHLAAttribute::extract_data() -- decoding 'Value' from attribute map
| |wormhole|-1|14.05|2010/02/03,17:44:58| TrickHLAAttribute::extract_data() -- decoding 'dvdt' from attribute map
| |wormhole|1|14.25|2010/02/03,17:44:59| TrickHLAFederate::wait_for_time_advance_grant() waiting for time advance grant (TAG)
| |wormhole|1|14.25|2010/02/03,17:44:59| TrickHLAFederate::wait_for_time_advance_grant() Time granted to 14.25 seconds.
| |wormhole|1|14.25|2010/02/03,17:44:59| TrickHLAManager::receive_cyclic_data()
| |wormhole|1|14.25|2010/02/03,17:44:59| TrickHLAManager::send_cyclic_data()
| |wormhole|1|14.25|2010/02/03,17:44:59| TrickHLAObject::create_attribute_set() -- adding 'Time' to attribute map
| |wormhole|1|14.25|2010/02/03,17:44:59| TrickHLAObject::create_attribute_set() -- adding 'Value' to attribute map
| |wormhole|1|14.25|2010/02/03,17:44:59| TrickHLAObject::create_attribute_set() -- adding 'dvdt' to attribute map
| |wormhole|1|14.25|2010/02/03,17:44:59| TrickHLAManager::send_requested_data()
| |wormhole|1|14.25|2010/02/03,17:44:59| TrickHLAFederate::time_advance_request()
requesting time advance grant (TAG) to 14.500000
| |wormhole|-1|14.30|2010/02/03,17:44:59| TrickHLAAttribute::extract_data() -- decoding 'Time' from attribute map
| |wormhole|-1|14.30|2010/02/03,17:44:59| TrickHLAAttribute::extract_data() -- decoding 'Value' from attribute map
| |wormhole|-1|14.30|2010/02/03,17:44:59| TrickHLAAttribute::extract_data() -- decoding 'dvdt' from attribute map
| |wormhole|1|14.50|2010/02/03,17:44:59| TrickHLAFederate::wait_for_time_advance_grant() waiting for time advance grant (TAG)
```

```
| |wormhole|1|14.50|2010/02/03,17:44:59| TrickHLAFederate::wait_for_time_advance_grant() Time granted to 14.5 seconds.
| |wormhole|1|14.50|2010/02/03,17:44:59| TrickHLAManager::receive_cyclic_data()
| |wormhole|1|14.50|2010/02/03,17:44:59| TrickHLAManager::send_cyclic_data()
| |wormhole|1|14.50|2010/02/03,17:44:59| TrickHLAObject::create_attribute_set() -- adding 'Time' to attribute map
| |wormhole|1|14.50|2010/02/03,17:44:59| TrickHLAObject::create_attribute_set() -- adding 'Value' to attribute map
| |wormhole|1|14.50|2010/02/03,17:44:59| TrickHLAObject::create_attribute_set() -- adding 'dvdt' to attribute map
| |wormhole|1|14.50|2010/02/03,17:44:59| TrickHLAManager::send_requested_data()
| |wormhole|1|14.50|2010/02/03,17:44:59| TrickHLAFederate::time_advance_request()
requesting time advance grant (TAG) to 14.750000
| |wormhole|-1|14.55|2010/02/03,17:44:59| TrickHLAAttribute::extract_data() -- decoding 'Time' from attribute map
| |wormhole|-1|14.55|2010/02/03,17:44:59| TrickHLAAttribute::extract_data() -- decoding 'Value' from attribute map
| |wormhole|-1|14.55|2010/02/03,17:44:59| TrickHLAAttribute::extract_data() -- decoding 'dvdt' from attribute map
| |wormhole|1|14.75|2010/02/03,17:44:59| TrickHLAFederate::wait_for_time_advance_grant() waiting for time advance grant (TAG)
| |wormhole|1|14.75|2010/02/03,17:44:59| TrickHLAFederate::wait_for_time_advance_grant() Time granted to 14.75 seconds.
| |wormhole|1|14.75|2010/02/03,17:44:59| TrickHLAManager::receive_cyclic_data()
| |wormhole|1|14.75|2010/02/03,17:44:59| TrickHLAManager::send_cyclic_data()
| |wormhole|1|14.75|2010/02/03,17:44:59| TrickHLAObject::create_attribute_set() -- adding 'Time' to attribute map
| |wormhole|1|14.75|2010/02/03,17:44:59| TrickHLAObject::create_attribute_set() -- adding 'Value' to attribute map
| |wormhole|1|14.75|2010/02/03,17:44:59| TrickHLAObject::create_attribute_set() -- adding 'dvdt' to attribute map
| |wormhole|1|14.75|2010/02/03,17:44:59| TrickHLAManager::send_requested_data()
| |wormhole|1|14.75|2010/02/03,17:44:59| TrickHLAFederate::time_advance_request()
requesting time advance grant (TAG) to 15.000000
| |wormhole|-1|14.80|2010/02/03,17:44:59| TrickHLAAttribute::extract_data() -- decoding 'Time' from attribute map
| |wormhole|-1|14.80|2010/02/03,17:44:59| TrickHLAAttribute::extract_data() -- decoding 'Value' from attribute map
| |wormhole|-1|14.80|2010/02/03,17:44:59| TrickHLAAttribute::extract_data() -- decoding 'dvdt' from attribute map
| |wormhole|-1|14.80|2010/02/03,17:44:59| TrickHLAFedAmb::removeObjectInstance() Instance ID:104
| |wormhole|-1|14.80|2010/02/03,17:44:59| TrickHLAManager::mark_object_as_deleted_from_federation()
Object 'P-side-Federate.Test' Instance-ID:104
| |wormhole|-1|14.80|2010/02/03,17:44:59| TrickHLAObject::mark_all_attributes_as_nonlocal()
  Object:'P-side-Federate.Test' FOM-Name:'Test' Instance-ID:104
   1/8 FOM-Attribute:'Time' Trick-Name:'P.sim_data.time' locally_owned: No
   2/8 FOM-Attribute:'Value' Trick-Name:'P.sim_data.value' locally_owned: No
   3/8 FOM-Attribute:'dvdt' Trick-Name:'P.sim_data.dvdt' locally_owned: No
   4/8 FOM-Attribute:'Phase' Trick-Name:'P.packing.phase_deg' locally_owned: No
   5/8 FOM-Attribute:'Frequency' Trick-Name:'P.sim_data.freq' locally_owned: No
   6/8 FOM-Attribute:'Amplitude' Trick-Name:'P.sim_data.amp' locally_owned: No
   7/8 FOM-Attribute:'Tolerance' Trick-Name:'P.sim_data.tol' locally_owned: No
   8/8 FOM-Attribute:'Name' Trick-Name:'P.sim_data.name' locally_owned: No
| |wormhole|-1|14.80|2010/02/03,17:44:59| TrickHLAFedAmb::removeObjectInstance() Instance ID:103
| |wormhole|-1|14.80|2010/02/03,17:44:59| TrickHLAManager::mark_object_as_deleted_from_federation()
Instance-ID:103 is not for a data object.
| |wormhole|1|15.00|2010/02/03,17:44:59| TrickHLAFederate::wait_for_time_advance_grant() waiting for time advance grant (TAG)
| |wormhole|1|15.00|2010/02/03,17:44:59| TrickHLAFederate::wait_for_time_advance_grant() Time granted to 15 seconds.
| |wormhole|1|15.00|2010/02/03,17:44:59| TrickHLAObject::process_deleted_object()
Object 'P-side-Federate.Test' Instance-ID:104.
| |wormhole|1|15.00|2010/02/03,17:44:59| SineObjectDeleted::deleted()
Object 'P-side-Federate.Test' deleted from the federation.
| |wormhole|1|15.00|2010/02/03,17:44:59| TrickHLAManager::receive_cyclic_data()
| |wormhole|1|15.00|2010/02/03,17:44:59| TrickHLAManager::send_cyclic_data()
| |wormhole|1|15.00|2010/02/03,17:44:59| TrickHLAObject::create_attribute_set() -- adding 'Time' to attribute map
| |wormhole|1|15.00|2010/02/03,17:44:59| TrickHLAObject::create_attribute_set() -- adding 'Value' to attribute map
| |wormhole|1|15.00|2010/02/03,17:44:59| TrickHLAObject::create_attribute_set() -- adding 'dvdt' to attribute map
| |wormhole|1|15.00|2010/02/03,17:44:59| TrickHLAManager::send_requested_data()
| |wormhole|1|15.00|2010/02/03,17:44:59| TrickHLAFederate::shutdown()
| |wormhole|1|15.00|2010/02/03,17:44:59| TrickHLAFederate::shutdown_time_constrained() Disabling HLA Time Constrained.
| |wormhole|1|15.00|2010/02/03,17:44:59| TrickHLAFederate::shutdown_time_regulating() Disabling HLA Time Regulation.
| |wormhole|1|15.00|2010/02/03,17:44:59|
 TRIVIAL: Trick Federation "SineWaveSim": RESIGNING FROM FEDERATION
| |wormhole|1|15.00|2010/02/03,17:44:59|
 ADVISORY: Trick Federation "SineWaveSim": RESIGNED FROM FEDERATION
| |wormhole|1|15.00|2010/02/03,17:44:59|
 TRIVIAL: TrickHLAFederate::destroy() Federation 'SineWaveSim': ATTEMPTING TO DESTROY FEDERATION
Federation destroyed
| |wormhole|1|15.00|2010/02/03,17:44:59|
 ADVISORY: TrickHLAFederate::destroy() Federation 'SineWaveSim': DESTROYED FEDERATION
| |wormhole|1|15.00|2010/02/03,17:44:59|
```

```
SIMULATION TERMINATED IN
  PROCESS: 1
  JOB/ROUTINE: 23/sim_services/mains/master.c
DIAGNOSTIC:
Simulation reached input termination time.

LAST JOB CALLED: THLA.THLA.federate.check_freeze_time()
             TOTAL OVERRUNS:          0
PERCENTAGE REALTIME OVERRUNS:     0.000%


      SIMULATION START TIME:     0.000
       SIMULATION STOP TIME:    15.000
    SIMULATION ELAPSED TIME:    15.000
        ACTUAL ELAPSED TIME:    15.000
       ACTUAL CPU TIME USED:     5.632
   SIMULATION / ACTUAL TIME:     1.000
      SIMULATION / CPU TIME:     2.663
  ACTUAL INITIALIZATION TIME:    0.000
     INITIALIZATION CPU TIME:    0.486
*** DYNAMIC MEMORY USAGE ***
     CURRENT ALLOCATION SIZE: 1437886
       NUM OF CURRENT ALLOCS:   972
         MAX ALLOCATION SIZE: 1437886
           MAX NUM OF ALLOCS:   972
       TOTAL ALLOCATION SIZE: 1765261
         TOTAL NUM OF ALLOCS:  7106
```

Listing 17.6: output showing conditionally sent cyclic data

Bibliography

[1] Edwin Z. Crues. *TrickHLA Inspection, Verification, and Validation*. National Aeronautics and Space Administration, Johnson Space Center, Software, Robotics & Simulation Division, Simulation and Graphics Branch, 2101 NASA Parkway, Houston, Texas, 77058, June 2020.

[2] Edwin Z. Crues. *Trick High Level Architecture (TrickHLA)*. National Aeronautics and Space Administration, Johnson Space Center, Software, Robotics & Simulation Division, Simulation and Graphics Branch, 2101 NASA Parkway, Houston, Texas, 77058, June 2020.

[3] Edwin Z. Crues. *TrickHLA Product Requirements*. National Aeronautics and Space Administration, Johnson Space Center, Software, Robotics & Simulation Division, Simulation and Graphics Branch, 2101 NASA Parkway, Houston, Texas, 77058, June 2020.

[4] Edwin Z. Crues. *TrickHLA Product Specification*. National Aeronautics and Space Administration, Johnson Space Center, Software, Robotics & Simulation Division, Simulation and Graphics Branch, 2101 NASA Parkway, Houston, Texas, 77058, June 2020.

[5] Dan E. Dexter. *Distributed Space Exploration Simulation Multiphase Initialization Design*. National Aeronautics and Space Administration, Johnson Space Center, Software, Robotics & Simulation Division, Simulation and Graphics Branch, 2101 NASA Parkway, Houston, Texas, 77058, December 2007.

[6] Dan E. Dexter. *Integrated Mission Simulation Multiphase Initialization Design*. National Aeronautics and Space Administration, Johnson Space Center, Software, Robotics & Simulation Division, Simulation and Graphics Branch, 2101 NASA Parkway, Houston, Texas, 77058, July 2009.

[7] NASA. *NASA Software Engineering Requirements*. Technical Report NPR-7150.2C, National Aeronautics and Space Administration, NASA Headquarters, Washington, D.C., August 2019.

[8] National Aeronautics and Space Administration, Johnson Space Center, Software, Robotics & Simulation Division, Simulation and Graphics Branch (ER7). Trick simulation environment: Documentation. Trick Wiki Site: https://nasa.github.io/trick/documentation/Documentation-Home, May 2020. Accessed 18 May 2020.

[9] National Aeronautics and Space Administration, Johnson Space Center, Software, Robotics & Simulation Division, Simulation and Graphics Branch (ER7). Trick simulation environment: Installation guide. Trick Wiki Site: https://nasa.github.io/trick/documentation/install_guide/Install-Guide, May 2020 (accessed May 18, 2020). Accessed 18 May 2020.

[10] National Aeronautics and Space Administration, Johnson Space Center, Software, Robotics & Simulation Division, Simulation and Graphics Branch (ER7). Trick simulation environment: Tutorial. Trick Wiki Site: https://nasa.github.io/trick/tutorial/Tutorial, May 2020 (accessed May 18, 2020). Accessed 18 May 2020.

[11] Simulation Interoperability Standards Organization/ Standards Activities Committee (SISO/SAC). *IEEE Standard for Modeling and Simulation (M&S) High Level Architecture (HLA) - Framework and Rules*. Technical Report IEEE-1516-2010, The Institute of Electrical and Electronics Engineers, 2 Park Avenue, New York, NY 10016-5997, August 2010.

Appendix A

`simplesine` Files

This appendix contains the data and source code files for the `simplesine` model.

A.1 `simplesine.h`

```
1  /*************************************************************************
2  PURPOSE: (A simple sinewave model.)
3  *************************************************************************/
4  #ifndef _SIMPLESINE_H_
5  #define _SIMPLESINE_H_
6
7  /*-------------------------------------------------------------------------
8  PURPOSE: (A structure to hold the sinewave parameters.)
9  -------------------------------------------------------------------------*/
10 typedef struct {
11   double A;    // *io -- the amplitude of the wave
12   double w;    // *io -- the frequency
13   double phi;  // *io -- the phase
14 } simplesine_params_T;
15
16 /*-------------------------------------------------------------------------
17 PURPOSE: (A structure to hold the sinewave state and space for derivatives
18 calulated for numerical integration.)
19 -------------------------------------------------------------------------*/
20 typedef struct {
21   double x;    // *o -- the current value of the sinewave, x(t)
22   double x_dot; // *o -- the current value of the sinewave derivative
23
24   double deriv_x;   // *o -- the derivative of x
25   double deriv_x_dot; // *o -- the derivative of x_dot
26 } simplesine_state_T;
27
28 /*-------------------------------------------------------------------------
29 PURPOSE: (A structure to hold the sinewave state.)
30 -------------------------------------------------------------------------*/
31 typedef struct {
32   simplesine_params_T params; // *i -- the parameters
33   simplesine_state_T state; // *i -- the state
34 } simplesine_T;
35
36 #endif
```

Listing A.1: `simplesine.h`

A.2 `simplesine_proto.h`

```
1  /*****************************************************************
2  ICG: (No)
3  PURPOSE: (A simple sinewave model.)
4  *****************************************************************/
5
6  #ifndef _SIMPLESINE_PROTO_H_
7  #define _SIMPLESINE_PROTO_H_
8
9  #ifdef __cplusplus
10 extern "C" {
11 #endif
12
13 #include "simplesine.h"
14 #include "sim_services/include/integrator.h"
15
16 int simplesine_calc( // RETURN: -- 0 on success
17   simplesine_T* p, // INOUT: -- the simplesine data structure
18   double t );      // IN:    s  the current time
19
20 int simplesine_deriv( // RETURN: -- 0 on success
21   simplesine_T* s ); // INOUT: -- pointer to the model data structure
22
23 int simplesine_integ( // RETURN: -- intermediate iteration count (0 at end)
24   INTEGRATOR* I,    // INOUT: -- integrator data structure
25   simplesine_T* s ); // INOUT: -- data structure for the harmonic oscillator
26
27 int simplesine_copyState( // RETURN: -- 0 on success
28   simplesine_state_T* fromP, // IN:  -- where is the data coming from?
29   simplesine_state_T* toP ); // OUT: -- where is the data going to?
30
31 int simplesine_copyParams( // RETURN: -- 0 on success
32   simplesine_params_T* fromP, // IN: -- where is the data coming from?
33   simplesine_params_T* toP ); // OUT: -- where is the data going to?
34
35 int simplesine_copy(  // RETURN: -- 0 on success
36   simplesine_T* fromP, // IN:  -- where is the data coming from?
37   simplesine_T* toP ); // OUT: -- where is the data going to?
38
39 int simplesine_calcError( // RETURN: -- 0 on success
40   double t,               // IN:    -- the current time
41   simplesine_T* s,        // IN:    -- the simplesine structure
42   simplesine_state_T* diff );// OUT: -- where to put the state error
43
44 void simplesine_compensate(
45   simplesine_params_T* paramsP,
46   simplesine_state_T* uncompensated_state,
47   simplesine_state_T* compensated_state,
48   double dt );
49
50 #ifdef __cplusplus
51 }
52 #endif
53 #endif
```

Listing A.2: `simplesine_proto.h`

A.3 simplesine_InteractionHandler.hh

```
1  /******************************** TRICK HEADER ****************************
2  PURPOSE: (A class to send/receive HLA interactions.)
3  LIBRARY DEPENDENCY: ((simplesine_InteractionHandler.o))
4  ***************************************************************************/
5
6  #ifndef _SIMPLESINE_INTERACTION_HANDLER_HH_
7  #define _SIMPLESINE_INTERACTION_HANDLER_HH_
8
9  #include "TrickHLA/include/TrickHLAInteractionHandler.hh"
10
11 class simplesine_InteractionHandler : public TrickHLAInteractionHandler
12   friend class InputProcessor; // necessary for Trick
13   friend void init_attrsimplesine_InteractionHandler(); // necessary for Trick
14
15   public:
16     simplesine_InteractionHandler();
17     virtual ~simplesine_InteractionHandler();
18
19     void send_sine_interaction( double send_time );
20     virtual void receive_interaction();
21
22   protected:
23     double lookahead_time;
24 };
25 #endif // _SIMPLESINE_INTERACTION_HANDLER_HH_: Do NOT put anything after this line!
```

Listing A.3: simplesine_InteractionHandler.hh

A.4 simplesine_LagCompensator.hh

```
1  /******************************** TRICK HEADER ****************************
2  PURPOSE: (Send and receive side lag compensation.)
3  LIBRARY DEPENDENCY: ((simplesine_LagCompensator.o))
4  ***************************************************************************/
5
6  #ifndef _SIMPLESINE_LAG_COMPENSATOR_HH_
7  #define _SIMPLESINE_LAG_COMPENSATOR_HH_
8
9  // Model include files.
10 #include "simplesine.h"
11
12 // Trick HLA include files.
13 #include "TrickHLA/include/TrickHLALagCompensation.hh"
14
15 class simplesine_LagCompensator : public TrickHLALagCompensation
16 {
17   friend class InputProcessor; // necessary for Trick
18   friend void init_attrsimplesine_LagCompensator(); // necessary for Trick
19
20   public:
21    // Public constructors and destructors.
22    simplesine_LagCompensator();       // Default constructor.
23    virtual ~simplesine_LagCompensator(); // Destructor.
24
25    int initialize( simplesine_T* sim_dataP, simplesine_T* lag_comp_dataP );
26
27    // From the TrickHLALagCompensation class.
28    virtual void send_lag_compensation();
```

```
30    // From the TrickHLALagCompensation class.
31    virtual void receive_lag_compensation();

33   private:
34     simplesine_T* uncompensated_stateP;
35     simplesine_T* compensated_stateP;
36 };
37 #endif // _SIMPLESINE_LAG_COMPENSATOR_HH_: Do NOT put anything after this line!
```

Listing A.4: simplesine_LagCompensator.hh

A.5 simplesine_Packing.hh

```
1  /******************************** TRICK HEADER *****************************
2  PURPOSE: (packing class)
3  LIBRARY DEPENDENCY: ((simplesine_Packing.o))
4  ***************************************************************************/
5  #ifndef _SIMPLESINE_PACKING_HH_
6  #define _SIMPLESINE_PACKING_HH_

8  #include "TrickHLA/include/TrickHLAPacking.hh"
9  #include "simplesine/include/simplesine.h"

11 class simplesine_Packing : public TrickHLAPacking
12 {
13   friend class InputProcessor; // necessary for Trick
14   friend void init_attrsimplesine_Packing(); // necessary for Trick

16   public:
17     simplesine_Packing();        // Default constructor.
18     virtual ~simplesine_Packing(); // Destructor.

20     virtual void init(
21       simplesine_T* originalP,
22       simplesine_T* packedP,
23       simplesine_T* unpackedP );

25     virtual void pack();
26     virtual void unpack();

28   private:
29     bool is_initialized;
30     simplesine_T* originalP;
31     simplesine_T* packedP;
32     simplesine_T* unpackedP;
33 };
34 #endif // _SIMPLESINE_PACKING_HH_: Do NOT put anything after this line!
```

Listing A.5: simplesine_Packing.hh

Appendix B

Interaction send/receive input files

This appendix contains the input files for the interaction sending and receiving simulations, `SIM_simplesine_hla_sendInt` and `SIM_simplesine_hla_receiveInt`.

B.1 Complete sender input file

```
#include "S_properties"
#include "S_default.dat"
#include "Log_data/states.d"
#include "Modified_data/realtime.d"
#include "Modified_data/publisher.d"

stop =32.5;

//
// Basic RTI/federation connection info
//

// Configure the CRC for the Pitch RTI.
THLA.federate.local_settings = "crcHost␣=␣localhost\n␣crcPort␣=␣8989";

THLA.federate.name         = "sender";
THLA.federate.FOM_modules  = "FOM.xml";
THLA.federate.federation_name = "simplesine";

THLA.federate.lookahead_time = THLA_DATA_CYCLE_TIME;
THLA.federate.time_regulating = true;
THLA.federate.time_constrained = true;
THLA.federate.multiphase_init_sync_points = "Phase1,␣Phase2";

THLA.federate.enable_known_feds  = true;
THLA.federate.known_feds_count   = 2;
THLA.federate.known_feds         = alloc(THLA.federate.known_feds_count);
THLA.federate.known_feds[0].name = "sender";
THLA.federate.known_feds[0].required = true;
THLA.federate.known_feds[1].name = "receiver";
THLA.federate.known_feds[1].required = true;

// TrickHLA debug messages.
THLA.manager.debug_handler.debug_level = THLA_LEVEL2_TRACE;
```

```
// DSES simulation configuration.
THLA_INIT.dses_config.owner          = "sender";
THLA_INIT.dses_config.run_duration   = 15.0;
THLA_INIT.dses_config.num_federates  = 1;
THLA_INIT.dses_config.required_federates = "sender";
THLA_INIT.dses_config.start_year     = 2007;
THLA_INIT.dses_config.start_seconds  = 0;
THLA_INIT.dses_config.scenario       = "Nominal";
THLA_INIT.dses_config.mode           = "Unknown";

// Simulation Configuration for DSES Multi-phase Initialization.
THLA.manager.sim_config.FOM_name     = "SimulationConfiguration";
THLA.manager.sim_config.name         = "SimConfig";
THLA.manager.sim_config.packing      = &THLA_INIT.dses_config;
THLA.manager.sim_config.attr_count   = 8;
THLA.manager.sim_config.attributes   = alloc(THLA.manager.sim_config.attr_count);

THLA.manager.sim_config.attributes[0].FOM_name = "owner";
THLA.manager.sim_config.attributes[0].trick_name = "THLA_INIT.dses_config.owner";
THLA.manager.sim_config.attributes[0].publish = true;
THLA.manager.sim_config.attributes[0].subscribe = true;
THLA.manager.sim_config.attributes[0].rti_encoding = THLA_UNICODE_STRING;

THLA.manager.sim_config.attributes[1].FOM_name = "run_duration";
THLA.manager.sim_config.attributes[1].trick_name = "THLA_INIT.dses_config.run_duration_microsec";
THLA.manager.sim_config.attributes[1].publish = true;
THLA.manager.sim_config.attributes[1].subscribe = true;
THLA.manager.sim_config.attributes[1].rti_encoding = THLA_LITTLE_ENDIAN;

THLA.manager.sim_config.attributes[2].FOM_name = "number_of_federates";
THLA.manager.sim_config.attributes[2].trick_name = "THLA_INIT.dses_config.num_federates";
THLA.manager.sim_config.attributes[2].publish = true;
THLA.manager.sim_config.attributes[2].subscribe = true;
THLA.manager.sim_config.attributes[2].rti_encoding = THLA_LITTLE_ENDIAN;

THLA.manager.sim_config.attributes[3].FOM_name = "required_federates";
THLA.manager.sim_config.attributes[3].trick_name = "THLA_INIT.dses_config.required_federates";
THLA.manager.sim_config.attributes[3].publish = true;
THLA.manager.sim_config.attributes[3].subscribe = true;
THLA.manager.sim_config.attributes[3].rti_encoding = THLA_UNICODE_STRING;

THLA.manager.sim_config.attributes[4].FOM_name = "start_year";
THLA.manager.sim_config.attributes[4].trick_name = "THLA_INIT.dses_config.start_year";
THLA.manager.sim_config.attributes[4].publish = true;
THLA.manager.sim_config.attributes[4].subscribe = true;
THLA.manager.sim_config.attributes[4].rti_encoding = THLA_LITTLE_ENDIAN;

THLA.manager.sim_config.attributes[5].FOM_name = "start_seconds";
THLA.manager.sim_config.attributes[5].trick_name = "THLA_INIT.dses_config.start_seconds";
THLA.manager.sim_config.attributes[5].publish = true;
THLA.manager.sim_config.attributes[5].subscribe = true;
THLA.manager.sim_config.attributes[5].rti_encoding = THLA_LITTLE_ENDIAN;

THLA.manager.sim_config.attributes[6].FOM_name = "scenario";
THLA.manager.sim_config.attributes[6].trick_name = "THLA_INIT.dses_config.scenario";
THLA.manager.sim_config.attributes[6].publish = true;
THLA.manager.sim_config.attributes[6].subscribe = true;
THLA.manager.sim_config.attributes[6].rti_encoding = THLA_UNICODE_STRING;

THLA.manager.sim_config.attributes[7].FOM_name = "mode";
THLA.manager.sim_config.attributes[7].trick_name = "THLA_INIT.dses_config.mode";
THLA.manager.sim_config.attributes[7].publish = true;
THLA.manager.sim_config.attributes[7].subscribe = true;
THLA.manager.sim_config.attributes[7].rti_encoding = THLA_UNICODE_STRING;

//
```

```
103 // Object class/attribute info.
104 //
105 THLA.manager.obj_count = 0;



108 //
109 // Interaction info
110 //

112 // Set the lookahead_time of the simplesine interaction handler to be equal to
113 // the already-specifed HLA federate lookahead_time.
114 //
115 publisher.interaction_handler.lookahead_time = THLA.federate.lookahead_time;

117 THLA.manager.inter_count = 1;
118 THLA.manager.interactions = alloc(THLA.manager.inter_count);

120 THLA.manager.interactions[0].FOM_name = "SimplesineParameters";
121 THLA.manager.interactions[0].publish = true;
122 THLA.manager.interactions[0].subscribe = false;
123 THLA.manager.interactions[0].handler = &publisher.interaction_handler;
124 THLA.manager.interactions[0].param_count = 3;
125 THLA.manager.interactions[0].parameters = alloc(THLA.manager.interactions[0].param_count);

127 THLA.manager.interactions[0].parameters[0].FOM_name = "A";
128 THLA.manager.interactions[0].parameters[0].trick_name = "publisher.simplesine.params.A";
129 THLA.manager.interactions[0].parameters[0].rti_encoding = THLA_LITTLE_ENDIAN;

131 THLA.manager.interactions[0].parameters[1].FOM_name = "w";
132 THLA.manager.interactions[0].parameters[1].trick_name = "publisher.simplesine.params.w";
133 THLA.manager.interactions[0].parameters[1].rti_encoding = THLA_LITTLE_ENDIAN;

135 THLA.manager.interactions[0].parameters[2].FOM_name = "phi";
136 THLA.manager.interactions[0].parameters[2].trick_name = "publisher.simplesine.params.phi";
137 THLA.manager.interactions[0].parameters[2].rti_encoding = THLA_LITTLE_ENDIAN;



141 read = 4.0;
142 CALL publisher.publisher.interaction_handler.send_sine_interaction(sys.exec);
```

Listing B.1: `SIM_simplesine_hla_sendInt` input file

B.2 Complete receiver input file

```
#include "S_properties"
#include "S_default.dat"
#include "Log_data/states.d"
#include "Modified_data/realtime.d"
#include "Modified_data/subscriber.d"

stop =32.5;

//
// Basic RTI/federation connection info
//

// Configure the CRC for the Pitch RTI.
THLA.federate.local_settings = "crcHost = localhost\n crcPort = 8989";

THLA.federate.name         = "receiver";
THLA.federate.FOM_modules  = "FOM.xml";
THLA.federate.federation_name = "simplesine";

THLA.federate.lookahead_time = THLA_DATA_CYCLE_TIME;
THLA.federate.time_regulating = true;
THLA.federate.time_constrained = true;
THLA.federate.multiphase_init_sync_points = "Phase1, Phase2";

THLA.federate.enable_known_feds   = true;
THLA.federate.known_feds_count    = 2;
THLA.federate.known_feds          = alloc(2);
THLA.federate.known_feds[0].name  = "receiver";
THLA.federate.known_feds[0].required = true;
THLA.federate.known_feds[1].name  = "sender";
THLA.federate.known_feds[1].required = true;

// TrickHLA debug messages.
THLA.manager.debug_handler.debug_level = THLA_LEVEL2_TRACE;


// DSES simulation configuration.
THLA_INIT.dses_config.owner          = "receiver";
THLA_INIT.dses_config.run_duration   = 15.0;
THLA_INIT.dses_config.num_federates  = 1;
THLA_INIT.dses_config.required_federates = "receiver";
THLA_INIT.dses_config.start_year     = 2007;
THLA_INIT.dses_config.start_seconds  = 0;
THLA_INIT.dses_config.scenario       = "Nominal";
THLA_INIT.dses_config.mode           = "Unknown";

// Simulation Configuration for DSES Multi-phase Initialization.
THLA.manager.sim_config.FOM_name     = "SimulationConfiguration";
THLA.manager.sim_config.name         = "SimConfig";
THLA.manager.sim_config.packing      = &THLA_INIT.dses_config;
THLA.manager.sim_config.attr_count   = 8;
THLA.manager.sim_config.attributes   = alloc(8);

THLA.manager.sim_config.attributes[0].FOM_name = "owner";
THLA.manager.sim_config.attributes[0].trick_name = "THLA_INIT.dses_config.owner";
THLA.manager.sim_config.attributes[0].publish = true;
THLA.manager.sim_config.attributes[0].subscribe = true;
THLA.manager.sim_config.attributes[0].rti_encoding = THLA_UNICODE_STRING;

THLA.manager.sim_config.attributes[1].FOM_name = "run_duration";
THLA.manager.sim_config.attributes[1].trick_name = "THLA_INIT.dses_config.run_duration_microsec";
THLA.manager.sim_config.attributes[1].publish = true;
THLA.manager.sim_config.attributes[1].subscribe = true;
```

```
64  THLA.manager.sim_config.attributes[1].rti_encoding = THLA_LITTLE_ENDIAN;

66  THLA.manager.sim_config.attributes[2].FOM_name = "number_of_federates";
67  THLA.manager.sim_config.attributes[2].trick_name = "THLA_INIT.dses_config.num_federates";
68  THLA.manager.sim_config.attributes[2].publish = true;
69  THLA.manager.sim_config.attributes[2].subscribe = true;
70  THLA.manager.sim_config.attributes[2].rti_encoding = THLA_LITTLE_ENDIAN;

72  THLA.manager.sim_config.attributes[3].FOM_name = "required_federates";
73  THLA.manager.sim_config.attributes[3].trick_name = "THLA_INIT.dses_config.required_federates";
74  THLA.manager.sim_config.attributes[3].publish = true;
75  THLA.manager.sim_config.attributes[3].subscribe = true;
76  THLA.manager.sim_config.attributes[3].rti_encoding = THLA_UNICODE_STRING;

78  THLA.manager.sim_config.attributes[4].FOM_name = "start_year";
79  THLA.manager.sim_config.attributes[4].trick_name = "THLA_INIT.dses_config.start_year";
80  THLA.manager.sim_config.attributes[4].publish = true;
81  THLA.manager.sim_config.attributes[4].subscribe = true;
82  THLA.manager.sim_config.attributes[4].rti_encoding = THLA_LITTLE_ENDIAN;

84  THLA.manager.sim_config.attributes[5].FOM_name = "start_seconds";
85  THLA.manager.sim_config.attributes[5].trick_name = "THLA_INIT.dses_config.start_seconds";
86  THLA.manager.sim_config.attributes[5].publish = true;
87  THLA.manager.sim_config.attributes[5].subscribe = true;
88  THLA.manager.sim_config.attributes[5].rti_encoding = THLA_LITTLE_ENDIAN;

90  THLA.manager.sim_config.attributes[6].FOM_name = "scenario";
91  THLA.manager.sim_config.attributes[6].trick_name = "THLA_INIT.dses_config.scenario";
92  THLA.manager.sim_config.attributes[6].publish = true;
93  THLA.manager.sim_config.attributes[6].subscribe = true;
94  THLA.manager.sim_config.attributes[6].rti_encoding = THLA_UNICODE_STRING;

96  THLA.manager.sim_config.attributes[7].FOM_name = "mode";
97  THLA.manager.sim_config.attributes[7].trick_name = "THLA_INIT.dses_config.mode";
98  THLA.manager.sim_config.attributes[7].publish = true;
99  THLA.manager.sim_config.attributes[7].subscribe = true;
100 THLA.manager.sim_config.attributes[7].rti_encoding = THLA_UNICODE_STRING;

102 //
103 // Object class/attribute info.
104 //
105 THLA.manager.obj_count = 0;


108 //
109 // Interaction info
110 //
111 THLA.manager.inter_count = 1;
112 THLA.manager.interactions = alloc(1);

114 THLA.manager.interactions[0].FOM_name = "SimplesineParameters";
115 THLA.manager.interactions[0].publish = false;
116 THLA.manager.interactions[0].subscribe = true;
117 THLA.manager.interactions[0].handler = &subscriber.interaction_handler;
118 THLA.manager.interactions[0].param_count = 3;
119 THLA.manager.interactions[0].parameters = alloc(3);

121 THLA.manager.interactions[0].parameters[0].FOM_name = "A";
122 THLA.manager.interactions[0].parameters[0].trick_name = "subscriber.simplesine.params.A";
123 THLA.manager.interactions[0].parameters[0].rti_encoding = THLA_LITTLE_ENDIAN;

125 THLA.manager.interactions[0].parameters[1].FOM_name = "w";
126 THLA.manager.interactions[0].parameters[1].trick_name = "subscriber.simplesine.params.w";
127 THLA.manager.interactions[0].parameters[1].rti_encoding = THLA_LITTLE_ENDIAN;

129 THLA.manager.interactions[0].parameters[2].FOM_name = "phi";
```

```
130 THLA.manager.interactions[0].parameters[2].trick_name = "subscriber.simplesine.params.phi";
131 THLA.manager.interactions[0].parameters[2].rti_encoding = THLA_LITTLE_ENDIAN;
```

Listing B.2: `SIM_simplesine_hla_receiveInt` input file