SISO-STD-018-2020

Standard for Space Reference Federation Object Model (SpaceFOM)

Version 1.0

25 October 2019

Prepared by The SpaceFOM Product Development Group (PDG)

P.O. Box 781238
Orlando, FL 32878-1238, USA

SISO Inc. Board of Directors
P.O. Box 781238
Orlando, FL 32878-1238, USA

Revision History

Version	Section	Date (MM/DD/YYYY)	Description
1.0	All	10/25/2019	Initial approved release version of document.

Participants

At the time this product was submitted to the Standards Activity Committee (SAC) for approval, the Space Reference Federation Object Model (SpaceFOM) Product Development Group had the following membership and was assigned the following SAC Technical Area Director:

Product Development Group

Björn Möller (Chair)
Alfredo Garro (Vice-Chair)
Rick Severinghaus (Secretary)

Chris McGroarty (SAC Technical Area Director)

Fredrik Antelius	Alberto Falcone	Rick Severinghaus
Edwin Z. Crues	Alfredo Garro	Fred Webber
Dannie Cutts	Michael Madden	James Wing
Jerry Davis	Björn Möller	
Dan Dexter	David Ronnfeldt	

The Product Development Group would like to especially acknowledge those individuals that significantly contributed to the preparation of this product as follows:

PDG Drafting Group

Edwin Z. Crues (Editor)

Dan Dexter	Alfredo Garro	Björn Möller
Alberto Falcone	Michael Madden	David G. Ronnfeldt

The following individuals comprised the ballot group for this product.

Ballot Group

Edwin Z. Crues	Paul T. Grogan	Björn Möller
Dannie Cutts	Mikael Karlsson	Katherine L. Morse
Dan Dexter	Jean-Phillipe Lebel	Laurent Prignac
Uwe Dobrindt	Reed Little	David G. Ronnfeldt
Alberto Falcone	Paul W. Maassel	Jonica Tramposch
Alfredo Garro	Chris McGroarty	

When the Standards Activity Committee approved this product on 18 December 2019, it had the following membership:

Standards Activity Committee

Katherine L. Morse (Chair)
David G. Ronnfeldt (Vice Chair)
Lance Marrou (Secretary)

Grant Bailey	Patrice Le Leydour	Katherine Ruben
John Daly	Chris McGroarty	John Stevens
Brad Dillman	Thom McLean	
David Graham	Laurent Prignac	

When the Executive Committee approved this product on 09 January 2020, it had the following membership:

Executive Committee

Michael O'Connor (Chair)
Robert Lutz (Vice Chair)
Jeff Abbott (Secretary)

Paul Gustavson	Lana McGlynn	Randy Saunders
Kenneth Konwin	Katherine L. Morse	Roy Scrudder
Chris Metevier	Stefan Sandberg	Robert Siegfried

Introduction

The aerospace industry has a long history of using simulation in the design, development, training, and operation of air and space systems. In recent years, the industry began using distributed simulation as a methodology for simulating these large-scale complex integrated systems. Many of these distributed simulations are based on the Institute of Electrical and Electronics Engineers (IEEE) 1516 High Level Architecture (HLA) standard for distributed simulation along with the concept of a Federation Object Model (FOM) [1][2][3]. HLA provides the interoperability infrastructure while a FOM codifies the data types, rules, and processes that must be adhered to when participating in HLA based simulations. Many of the large-scale HLA simulations developed for defense systems are based on a standard called the Real-time Platform Reference Federation Object Model (RPR FOM) [4]. However, the RPR FOM has known limitations in its ability to model systems much beyond the Earth's atmosphere. While the RPR FOM might work for ground based and atmospheric systems, it is not well suited to the space domain.

Regardless, space domain based distributed simulations are being built, many based on HLA. However, without a standardized FOM, HLA is insufficient to ensure a priori interoperability. As a result, the Simulation Interoperability Standards Organization (SISO) is developing a Space Reference FOM (SpaceFOM) [5][6][7][8]. The SpaceFOM is developed and maintained by the Space Reference FOM Product Development Group (SpaceFOM PDG), which is composed of members from several countries. They contribute experiences from projects within the National Aeronautics and Space Administration (NASA), the European Space Agency (ESA), and other organizations and represent government, academia, and industry.

The SpaceFOM defines a collection of HLA-compliant data constructs, modeling standards, and execution control process standards that support interoperability between simulations in the space domain. It is designed to link simulations of discrete physical entities into distributed collaborative simulations of complex space related systems. The initial version of the SpaceFOM provides the following:

- Definitions of specific roles and responsibilities of federates within a federation execution;
- Definitions of common data types useful in the space domain;
- Definitions of common time lines and time scales needed for time homogeneity;
- Definitions for specific time-stepped focused time management approaches;
- A flexible positioning system, using reference frames for locating arbitrary bodies in space;
- A naming convention for operational reference frames;
- Support for physical entities (e.g., space vehicles and astronauts);
- Support for physical interfaces (e.g., docking ports and sensor locations);
- A synchronized execution control strategy and framework;
- Rules defining the requirements for SpaceFOM compliance;
- A core base set of FOM modules needed for a SpaceFOM-compliant federation execution.

This document also encapsulates the design rationale for and guidance in the use of the SpaceFOM. It provides descriptions of FOM classes, data types, and rules for accomplishing specific distributed simulation tasks.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF RULES

Standard for Space Reference Federation Object Model

1. Overview

The Space Reference Federation Object Model (SpaceFOM) consists of this explanatory and prescriptive text based document and a collection of computer-parsable Federation Object Model (FOM) module files. The FOM modules are the authoritative definition of the principal data constructs used by the HLA Run-Time Infrastructure (RTI). This document encapsulates the design rationale for and guidance in the use of the SpaceFOM. It defines a collection of HLA-compliant data constructs, modeling standards, and execution control process standards that support interoperability between simulations in the space domain.

1.1. Scope

A FOM is a specification defining the information exchanged at runtime to achieve a given set of federation objectives. This includes object classes, object class attributes, interaction classes, interaction parameters and other relevant information. This document accompanies the SpaceFOM modules. It provides the usage rules for the SpaceFOM, and the definitions, descriptions and rationale not otherwise specified within the standard FOM module format. It also describes how to use and extend the SpaceFOM.

1.2. Purpose

The SpaceFOM is designed to support interoperability between simulations in the space systems domain. This includes simulations executing in real-time as well as those executing in logical-time (including as-fast-as-possible). The principal intended areas of use are training, analysis, mission support and engineering. However, the development group anticipates other areas of use, like test and concept exploration.

The benefits of the SpaceFOM include:

Interoperability: The ability for several simulations, each focusing on particular tasks, to interoperate and jointly create a collaborative simulation with wider and richer contexts.

Composability: The ability to build collaborative simulations from components that can be combined in different ways, with new or existing simulations, to reach a particular goal.

Reusability: The ability to use existing simulations in new contexts. The SpaceFOM enables the construction of generic and reusable simulations and tools for the space domain.

The SpaceFOM builds upon many years of experiences of simulations by professionals in government organizations, industry, and academia. Early prototypes of the SpaceFOM have been tested in the SISO/SCS programs called the “Simulation Exploration Experience” (formerly known as “Smackdown”) [10].

1.3. Objectives

The HLA infrastructure is a general-purpose mechanism allowing differing approaches to distributed simulation (i.e., there are many ways to accomplish the same thing). The SpaceFOM was developed as a common approach to distributed simulations, with the end goal of fostering interoperability. Its objectives are to support the development of interoperable simulations of complex space systems and missions, and enhance a priori interoperability among SpaceFOM users.

1.4. Intended Audience

The intended audiences for this document are individuals or groups involved in developing distributed space systems federations using the SpaceFOM.

1.5. Relationship to the HLA standard

The SpaceFOM provides an implementation for one of the principal components of the IEEE 1516-2010 “High Level Architecture” (HLA) standard. Detailed descriptions of HLA concepts and services are provided in the HLA standard [1][2][3]. However, the following simplified figure (see Figure 1-1) and descriptions of the principal HLA concepts and components are sufficient for this discussion:

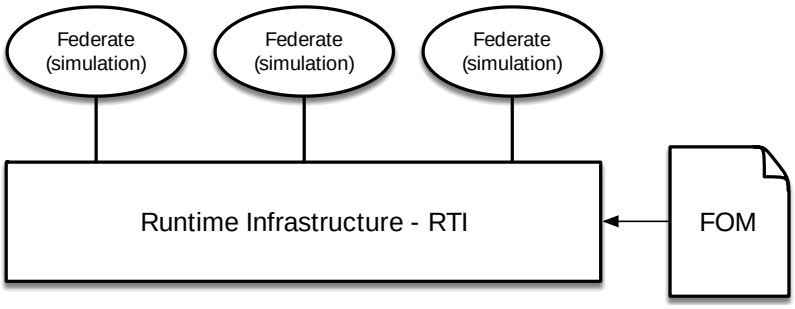

Figure 1-1: Main Concepts of the HLA Standard

Federate: An application that may be or is currently coupled with other software applications under a FOM Document Data (FDD) and an RTI. Examples of federates include space vehicle simulations, space environment simulations, virtual cockpits with joysticks, visualization tools, and data loggers.

Federation: A named set of federate applications and a common FOM that are used as a whole to achieve some specific objective.

Federation Execution: The actual operation, over time, of a set of joined federates that are interconnected by an RTI.

Federation Object Model (FOM): A specification defining the information exchanged at runtime to achieve a given set of federation objectives [17]. This information includes object classes, object class attributes, interaction classes, interaction parameters, and other relevant information. The FOM is specified to the RTI using one or more FOM modules. The RTI assembles a FOM using these FOM modules and one Management Object Model (MOM) and Initialization Module (MIM), which is provided automatically by the RTI or, optionally, provided to the RTI when the federation execution is created. The SpaceFOM provides a standardized data model suitable for space simulations. The SpaceFOM is provided as a collection of dependent FOM modules.

Run-Time Infrastructure (RTI): The software that provides common interface services during an HLA federation execution for synchronization and data exchange. An RTI library, called a Local RTI Component (LRC), provides these services used by each federate. The LRC is responsible for communicating with other LRCs in order to provide RTI services. Each federate connects to a common RTI executable, called a Central Runtime Component (CRC), in order to exchange data and provide other services.

1.6. Services provided by the HLA Run-Time Infrastructure

A Federation based on the SpaceFOM uses a number of services provided by the RTI. The most important services are:

Federation Management: These services allow for the coordination of federation-wide activities throughout the life of a federation execution. Such services include federation execution creation and destruction, federate application joining and resigning, federation synchronization points, and save and restore operations. A federate only participates in one federation execution. Several federation executions may exist at the same time. An example is when several teams in an organization simulate different scenarios in parallel.

Declaration Management: These services allow joined federates to specify the types of data they will supply to, or receive from, the federation execution. This process is done via a set of publication and subscription services along with some related services.

Object Management: These services support the life-cycle activities of the objects and interactions used by the joined federates of a federation execution. These services provide for registering and discovering object instances, updating and reflecting the instance attributes associated with these object instances, deleting or removing object instances as well as sending and receiving interactions and other related services.

Time Management: These services allow joined federates to operate with a logical concept of time and to jointly maintain a distributed virtual clock. These services support discrete event simulations and assurance of causal ordering among events. This enables federates to execute and manage their time advance and data exchange in such a way that causality is maintained and repeatability is possible. Time may be advanced in real-time, scaled real-time or as fast as possible.

Ownership Management: These services are used to establish a specific joined federate's privilege to provide values for an object instance attribute as well as to facilitate dynamic transfer of this privilege (ownership) to other joined federates during a federation execution.

The Management Object Model (MOM): A group of predefined HLA constructs (object and interaction classes) that provide the following:

a) Access to federation execution operating information;

b) Insight into the operations of joined federates and the RTI;

c) Control of the functioning of the RTI, the federation execution, and the individual joined federates.

Additional HLA services may be used at each federate developer's choice.

1.7. Basic SpaceFOM concepts, principals and patterns

The SpaceFOM relies on a collection of data type definitions, concepts, principals, and patterns to provide a consistent, coherent and extensible framework. All SpaceFOM-compliant federations are expected to use this framework to manage time, spatial data, and execution. At the highest level, this framework consists of the following topic areas:

Roles and Responsibilities: The SpaceFOM defines specific, and sometimes specialized, roles for federates participating in a SpaceFOM-compliant federation execution. These roles and responsibilities are discussed in Section 3.2.

Common Data Types: The SpaceFOM defines a collection of common data types to be used by all SpaceFOM-compliant FOM modules that either derive from or extend the standard SpaceFOM modules. These are discussed in general in Section 3.3 and in detail in Appendix C.

Time Lines and Time Representations: The SpaceFOM defines a set of time lines used to coordinate a synchronized time homogeneous federation execution. In addition, the SpaceFOM defines a standard time representation used to coordinate physical representations of time. This standard time representation is tied to the standard time lines and can be used to map to other known standard time representations. These time lines and time representations are discussed in Chapter 4 with some detailed time discussions presented in Appendix D.

Time Management: The SpaceFOM defines a specific set of time-step based time management approaches in the form of patterns for advancing time in a SpaceFOM-compliant federation execution. The time management patterns are described in Section 4.4.

Reference Frames: The SpaceFOM defines a flexible positioning system using hierarchical relationships between reference frames for locating arbitrary bodies in space. Reference frames are discussed in Chapter 5 and with some additional mathematical detail in Appendix E.

Reference Frame Naming Convention: In addition to the reference frames themselves, the SpaceFOM also defines a naming convention for reference frame and definitions of well-known reference frames. The reference frame naming convention is presented in Section 5.3. A discussion of well-known reference frames is presented in Appendix F.

Entities and Interfaces: The principal objective of a space systems simulation is to model the behavior of a complex collection of space systems. At their most fundamental level, these systems are represented as a collection of physical or dynamical entities and physical interfaces. Physical entities represent the basic state representation of where a space system is in time and space. A physical interface represents some location with respect to a physical entity or other physical interface. Physical interfaces are used to establish relative placements for things associated with a space system (e.g., docking ports or sensors). Physical entities and interfaces are discussed in Chapter 6.

Execution Control: The SpaceFOM provides an execution control framework for time-stepped, synchronized runtime execution with support for both non-real-time and real-time pacing. It provides an initialization model that supports the concepts of required federates, multiphase initialization, and late joining federates. It also supports synchronized runtime execution mode transitions between run, freeze, and shutdown. SpaceFOM execution control is discussed in Chapter 7.

FOM Modules: The FOM modules are a set of Extensible Markup Language (XML) formatted files, separate from this document, that provide the authoritative definition of the principal data constructs used by the HLA RTI. There are five (5) FOM modules that compose the core base for the SpaceFOM: the switches module (SISO_SpaceFOM_switches), the data types module (SISO_SpaceFOM_datatypes), the management module (SISO_SpaceFOM_management), the environment module (SISO_SpaceFOM_environment), and the entities module (SISO_SpaceFOM_entity). These modules are discussed at the end of each related chapter and are presented in overview in Appendix C.

Rules and Guidance: The descriptive text associated with the SpaceFOM is captured in the following chapters. At the end of principal descriptive chapters, there is a section on rules associated with the topic of that chapter. When appropriate, the rules section may be followed by a section on guidance associated with the topic of that chapter. The rules are prescriptive (normative) and define a context for compliance with the SpaceFOM. The guidance is informative and provides some background, rationale and/or best practice regarding the associated topic of the chapter.

1.8. Building Federations using the SpaceFOM

The SpaceFOM provides a starting point for building federations for the space domain. In addition to this, it is strongly recommended that each particular federation development team produce a Federation Execution Specific Federation Agreement (FESFA) that specifies additional design information (see Section 3.1 and Appendix A). This may include:

- The purpose of the Federation;
- Range of scenarios to be supported;
- Identification of principal federation execution data and points of contact;
- Federation composition, including federates playing principal roles;
- Time management policies;
- Constituent reference frame structure;
- Additional FOM modules and principal object instances;
- Common data or databases that are used;
- Additional services and conventions for services exchange;
- Technical configuration data, such as networking and host information;
- Test and integration procedures.

In addition to the FESFA, it is strongly recommended that each participating federate developer produce a Federate Compliance Declaration (FCD) document that specifies the information needed to successfully integrate their federate into a SpaceFOM-compliant federation execution (see Section 3.1 and Appendix B). This may include:

- The purpose of the Federate;

- Range of scenarios to be supported;
- Identification of principal federate data and points of contact;
- Capability to fulfill specified SpaceFOM principal Roles;
- Time management policies and types supported;
- Dependent and provided reference frames;
- Additional FOM modules and principal object instances;
- Common data or databases that are used;
- Additional services and conventions for services exchange;
- Test and integration procedures;
- Compliance statement.

In order to develop a SpaceFOM federation, development teams are encouraged to use and follow the IEEE 1730-2010 Distributed Simulation Engineering and Execution Process (DSEEP) [9]. It provides a proven seven-step process, from establishing the goals and constraints for the federation to the final execution.

1.9. Extending the SpaceFOM

Clearly, the SpaceFOM will not provide all possible data types and object constructs needed for all possible space systems federations. However, when SpaceFOM-compliant federations need to add additional capabilities not already present in the SpaceFOM, they should build upon and extend the existing SpaceFOM. Specifically, the needed extensions should be created by adding more FOM modules and not by modifying the existing standardized FOM modules. The rules for adding extensions are described in Figure 1-2 and Rule 1-5.

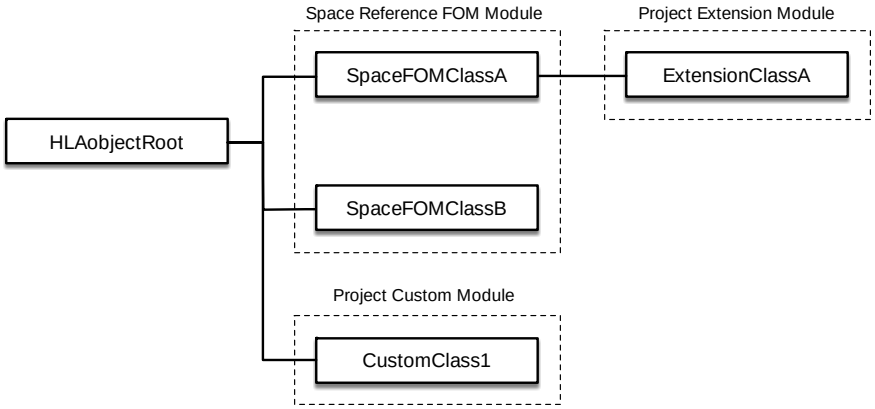

Figure 1-2: Adding Extensions to the SpaceFOM

The main rules are:

- Object and Interaction classes that provide specializations of a standardized SpaceFOM class shall be defined as subclasses of these standardized classes. They shall be defined in a project specific extension module. See in Figure 1-2 that ExtensionClassA defines a specialization of the SpaceFOMClassA.
- Object and Interaction classes that do not provide specializations of a standardized SpaceFOM class shall be defined as subclasses of HLAobjectRoot. They shall be defined in a project specific custom module. See ProjectClass1 in Figure 1-2.
- Any new data types shall be defined in project specific extension modules.

As a general guideline, class and data type extensions should be grouped into extension modules based on functional areas.

1.10. Rules

This SpaceFOM document presents a series of descriptive discussions from which an associated set of prescriptive (normative) rules is inferred. The general form for these rules are a leading numbered cross-referenced **Rule** title, followed by a *Requirement* statement of the rule, and then ending with a *Rationale* for the specified rule. The following is a generalized example of a SpaceFOM rule specification:

Rule <C>-<#>: Brief Cross-Referenced Numbered Rule Title

Requirement: A rule shall have a brief numbered rule title, a requirement "shall" statement, and a rationale statement. The rule title number is a two part hyphenated number with the first number <C> corresponding to the chapter number and the second number corresponding to the ordered occurrence of that rule within the chapter.

Rationale: This rule format permits the unique specification of a SpaceFOM rule. The **Title** provides a brief descriptive phrase appropriate for listing in the prefatory section of the document. The *Requirement* provides a statement of the rule using "shall" based requirements language. This requirement is the authoritative statement for the rule. The *Rationale* provides context and explanation of the rule as it pertains to the SpaceFOM.

Based on the discussion in the preceding sections of this chapter, we can infer the following general rules for SpaceFOM-compliant HLA federation executions and federates:

Rule 1-1: Federation Execution Compliance with HLA Rules

Requirement: A SpaceFOM-compliant federation execution shall comply with the HLA Rules pertaining to an HLA federation execution [1].

Rationale: A SpaceFOM-compliant federation execution is an HLA federation execution and must comply with the general HLA rules.

Rule 1-2: Federate Compliance with HLA Rules

Requirement: A SpaceFOM-compliant federate shall comply with the HLA Rules pertaining to an HLA federate [1].

Rationale: A SpaceFOM-compliant federate is an HLA federate and must comply with the general HLA rules.

Rule 1-3: Federate Compliance with the HLA OMT Specification.

Requirement: A SpaceFOM-compliant federate shall comply with the HLA Object Model Template (OMT) Specification [3].

Rationale: A SpaceFOM-compliant federate is an HLA federate and must comply with the HLA OMT specification.

Rule 1-4: Do Not Modify SpaceFOM Modules

Requirement: A SpaceFOM-compliant federation execution shall use the provided unmodified SpaceFOM modules.

Rationale: Modifying the existing SpaceFOM modules modifies the standard, in essence breaking the standard.

Rule 1-5: Extend SpaceFOM With Added FOM Modules

Requirement: A SpaceFOM-compliant federation execution shall provide additional capabilities not already provided in the SpaceFOM with additional FOM modules that extend the existing SpaceFOM modules whenever possible. This includes the following:

(i) Object and Interaction classes that provide specializations of a standardized SpaceFOM class shall be defined as subclasses of the standardized classes; they shall be defined in a project specific extension module;

(ii) Object and Interaction classes that do not provide specializations of a standardized SpaceFOM class shall be defined as subclasses of HLAobjectRoot; they shall be defined in a project specific custom module;

(iii) any new data types shall be defined in project specific extension modules.

Rationale: The nature of the IEEE 1516-2010 HLA standard use of modular FOMs is to promote extension through specialization and not replication. Replicating a capability already present in an existing SpaceFOM module increases the likelihood of conflicts and incompatibilities. Building on or extending from the existing SpaceFOM base promotes a priori interoperability.

2. Definitions, Acronyms, and Abbreviations

English words are used in accordance with their definitions in the latest edition of Webster's New Collegiate Dictionary [14] except when special SISO Product-related technical terms are required.

The current version of the Department of Defense (DoD) Modeling and Simulation (M&S) Glossary [15] is applicable for most terms in this standard. Any definitions included in this section are specific to the SpaceFOM standard. The IEEE Standards Dictionary Online [16] should be consulted for terms not defined in this section.

2.1. Definitions

Term	Definition
A priori Interoperability	Federated operation requiring no special adjustment when changing federations.
Buffer Compatible	Producing identical 'on the wire' data encoding.
Endian	Referring to the byte storage order of multibyte data.
Elastic Pacing	If frame overruns occur, the pacing federate will run as fast as possible until synchronization of HLA logical time and real-time has been reestablished.
Early Joiner	A federate that joins the federation execution during initialization early enough in the initialization process to participate in multiphase initialization.
Federate	The federated behavior of the simulation exposed through the HLA interface.
Federate Time Step	Delta when performing TAR as well as lookahead of a federate (Logical time).
Federation Time Step	Federate Time Step of Pacing Federate.
Firm Real-Time	A limited number of overruns are allowed. Shall use elastic pacing.
Frame Overrun	Failure to complete processing of a HLA logical time frame during the desired real-time frame.
Hard Real-Time	No frame overruns are allowed.
Late Joiner	A federate that joins into the federation after initialization is complete.
Least Common Time Step	The least common value of all the federate time steps for the time regulating and time constrained federates in a federation execution.
Master Federate	The federate responsible for coordinating federation execution initialization and managing federation execution mode transitions.
Object Instance ID	The string used to uniquely identify an object, implemented using the HLA object instance name.
Optional Attribute	An attribute that does not need to be provided to the federation.
Pacing Federate	A specialized SpaceFOM federate that actively manages the advancement of HLA logical time during a SpaceFOM-compliant federation execution.
Real-time Pacing Federate	A federate that actively manages the advancement of HLA logical time in relation to real-time, as provided by a reference time source. The pacing can be performed one-to-one to real-time, using scaled real-time, or any other relationships.
Reflection	Update of attributes at an HLA federate.
Required Attribute	An attribute that has to be provided to the federation for each instance of the Object Class that has this attribute. Typically, the attribute is provided right after the object instance is registered.
Required Federate	A federate that must be present for the federation execution to proceed.
Root Reference Frame	The top-level reference frame in the reference frame tree.
Publishing Federate	The federate responsible for publishing the root reference frame.
Simulation	Application that mimics the behavior of a process or system, including external interfaces.
Simulation Time Step	Native time step used inside simulation when state is propagated (for example Dynamics Rate).

Soft Real-Time	An unlimited number of overruns are allowed. Shall use elastic pacing.

2.2. Acronyms and Abbreviations

Acronym or Abbreviation	Meaning
AFAP	As Fast As Possible
BE	Big Endian
CCT	Computer Clock Time
CRC	Central Runtime Component
CTE	Central Timing Equipment
DAG	Directed Acyclic Graph
DoD	United States Department of Defense
ESA	European Space Agency
ExCO	Execution Configuration Object
FCD	Federate Compliance Declaration
FDD	FOM Document Data
FESFA	Federation Execution Specific Federation Agreement
FOM	Federation Object Model
FST	Federation Scenario Time
GALT	Greatest Available Logical Time
GMST	Greenwich Mean Sidereal Time
GPS	Global Positioning System
GPST	GPS Time
HLA	High Level Architecture
HLT	HLA Logical Time
HLTB	HLA Logical Time Boundary
ID	Identifier
IEEE	Institute of Electrical and Electronics Engineers
LCTS	Least Common Time Step
LE	Little Endian
LRC	Local RTI Component
MIM	MOM and Initialization Module
MOM	Management Object Model
MPI	Multi-Phase Initialization
MTR	Mode Transition Request
M&S	Modeling and Simulation
NASA	National Aeronautics and Space Administration
OMT	Object Model Template
PDF	Portable Document Format
PDG	Product Development Group
PT	Physical Time
RRFP	Root Reference Frame Publisher
RTI	Runtime Infrastructure
SAC	Standards Activity Committee
SCS	The Society for Modeling and Simulation International
SET	Simulation Elapsed Time
SI	International System of Units
SISO	Simulation Interoperability Standards Organization
SpaceFOM	Space Reference Federation Object Model
SST	Simulation Scenario Time
TAI	Atomic Time
TAR	Time Advance Request
TCB	Barycentric Coordinated Time

TCG	Geocentric Coordinated Time
TJD	Truncated Julian Date
TSO	Timestamp Order
TT	Terrestrial Time
UNXT	Unix Time
UT	Universal Time
UTC	Coordinated Universal Time
XML	Extensible Markup Language

3. Technical Overview

The SpaceFOM is, in principal, a document and collection of files. This document defines and describes the semantic content of the SpaceFOM and is also accompanied by a collection of XML FOM modules in the form of IEEE 1516-2010 compliant FOM files [3]. This document and these files support the development of SpaceFOM-compliant federations and help to ensure a basic level of interoperability between federates. In order to preserve interoperability, it is expected that the SpaceFOM will be used as is. Federations with specific additional needs should provide those in the form of additional FOM modules that extend the basic capabilities provided in the SpaceFOM.

The IEEE 1516-2010 compliant SpaceFOM modules shall be considered normative. The human-readable Portable Document Format (PDF) and all other data files shall be considered informative annexes. If any statement is interpreted to be in conflict with the IEEE 1516-2010 compliant FOM modules, the FOM modules shall take precedence.

3.1. General Concepts

The SpaceFOM provides a core semantic and FOM framework as a starting point for the development of federation agreements and additional FOMs that meets the specific needs of a particular application. It is possible to use the SpaceFOM as is, implementing all parts or a subset. It is also possible to extend the SpaceFOM and add more concepts, requirements, and design principles.

The SpaceFOM puts a number of firm requirements on SpaceFOM-compliant federation executions and the associated federates:

Federation Execution Requirements: These are requirements applicable to a federation execution; in particular to the set of federates and associated data.

Federate Requirements: These are requirements applicable to any federate claiming compliance with the SpaceFOM. Some federate requirements provide for levels of support. For instance, a federate can choose the role it is capable of fulfilling (Master Federate, Pacing Federate, etc.) or what time management type it supports (no pacing, scaled pacing, etc.). These choices will, in part, determine in which SpaceFOM-compliant federation execution it can participate.

When developing federations based on the SpaceFOM there are two important documents that are required to ensure interoperability: the FESFA and the FCD (see Rule 3-1 and Rule 3-2).

Federation Execution Specific Federation Agreement (FESFA): The FESFA is a document that provides the additional specific configuration data necessary to achieve interoperability. Several rules in the SpaceFOM put requirements on what data need to be recorded in the Federation Execution Specific Federation Agreement. A template for a FESFA is provided in Appendix A.

Federate Compliance Declaration (FCD): The FCD describes which capabilities a federate has and which roles it can play in a SpaceFOM-compliant federation execution. A template for a FCD is provided in Appendix B.

3.2. Federate Roles and Responsibilities

A SpaceFOM-compliant federation execution can have any number of federates. However, there are a number of specified roles that a participating federate can play and a number of specialized roles that must be assumed by one or more federates in the federation execution. This is related to the fulfillment of specific responsibilities associated with the management of the federation execution and the control of participating federates.

The general federate roles in a SpaceFOM-compliant federation execution are: Master, Pacing, Root Reference Frame Publisher (RRFP), and Other. There are also classifications associated with federates regarding when they join into a federation execution: Early Joiner and Late Joiner. The Master, Pacing and RRFP roles are special roles and must be Early Joiner federates for reasons discussed in detail in Chapter 7. A federate may play one or more of these roles. However, all these roles must be represented in a SpaceFOM-compliant federation execution.

In addition to the specialized roles, a federation execution may have other federates required to meet the federation execution's operational objectives. Federates that fulfill the specialized roles and federates that are required for a successful federation execution need to be documented in the FESFA.

Having covered the general concepts of roles and responsibilities, we can now discuss the specific specialized federate roles of Master, Pacing and RRFP.

3.2.1. Master Federate Responsibilities

The Master Federate is the federate in a SpaceFOM-compliant federation execution that fulfills the Master role. The primary responsibility of the Master Federate is to manage the federation execution. The principal mechanism for managing a federation execution is a singleton object instance called the Execution Configuration Object (ExCO), which is discussed in detail in Chapter 7.

3.2.2. Pacing Federate Responsibilities

The Pacing Federate is the federate in a SpaceFOM-compliant federation execution that fulfills the Pacing role. The primary responsibility of the Pacing Federate is to actively manage the advancement of HLA logical time during execution as specified in Chapter 4.

3.2.3. Root Reference Frame Publisher Federate Responsibilities

The Root Reference Frame Publisher (RRFP) Federate is the federate in a SpaceFOM-compliant federation execution that fulfills the RRFP role. The primary responsibility of the RRFP Federate is to publish the root reference frame object instance. The root reference frame is the basis for the federation execution's reference frame tree. Reference frames in general, and the reference frame tree specifically, are discussed in detail in Chapter 5.

3.3. SpaceFOM Modules

The SpaceFOM standard is composed of this document and a collection of FOM files in an XML format [17]. This document provides the rules and semantic descriptions of the key concepts and components of the SpaceFOM. The FOM files provide the HLA required computer parsable definitions of SpaceFOM related HLA constructs.

The SpaceFOM standard defines 5 constituent FOM modules: SISO_SpaceFOM_switches, SISO_SpaceFOM_datatypes, SISO_SpaceFOM_management, SISO_SpaceFOM_environment, and SISO_SpaceFOM_entity. All FOM models are presented in Appendix C. The Data Types FOM module is a general module used across the other modules and is discussed in this section. The Switches FOM module is discussed in Chapter 8, the Management FOM module is discussed in Chapter 7, the Environment FOM module is discussed in Chapter 5, and the Entity FOM module is discussed in Chapter 6.

3.3.1. The Data Types FOM Module (SISO_SpaceFOM_datatypes)

This SpaceFOM module provides a number of essential data types common to the other SpaceFOM modules. These are used for object attributes as well as interaction parameters. The data types use the International System of Units (SI) [18] wherever possible.

The SpaceFOM Data Types Module (SISO_SpaceFOM_datatypes) defines simple data types, array data types, and fixed record types used throughout the other SpaceFOM modules. The data types in this FOM module are presented in Appendix Section C.2 along with tables describing the types and their relationship to one another.

3.4. Rules

Based on the preceding discussion, we can infer some general and specific rules associated with SpaceFOM documentation, federate roles and federate responsibilities.

3.4.1. Documentation Rules

Rule 3-1: The Federation Execution Specific Federation Agreement (FESFA)

Requirement: Each SpaceFOM-compliant federation execution or class of federation executions shall have a FESFA document.

Rationale: The FESFA is a document that provides the additional specific configuration data necessary to achieve interoperability. Several rules in the SpaceFOM put requirements on what data need to be recorded in the Federation Execution Specific Federation Agreement. A template for a FESFA is provided in Appendix A.

Rule 3-2: The Federate Compliance Declaration (FCD)

Requirement: Each SpaceFOM-compliant federate shall have an FCD document.

Rationale: The FCD describes which capabilities a federate has and which roles it can play in a SpaceFOM-compliant federation execution. A template for a FCD is provided in Appendix B.

Rule 3-3: Documentation of Required Federates in FESFA

Requirement: Federates required for a successful federation execution shall be documented in the associated FESFA.

Rationale: The general execution control strategy for a SpaceFOM-compliant federation execution requires that one or more federates fulfill specific specialized roles: Master, Pacing and RRFP. In addition, a specific federation execution may require other federates, not necessarily fulfilling one of the specialized roles, in order to successfully operate and generate meaningful results. Documenting the required federates in the FESFA makes it possible to check for the presence of these federates during the initialization process discussed in detail in Chapter 7.

Rule 3-4: Specification of Federate Role Capabilities in FCD

Requirement: The roles a SpaceFOM-compliant federate is capable of fulfilling shall be specified in its associated FCD.

Rationale: When formulating a SpaceFOM-compliant federation execution, it is necessary to have federates that fill all the necessary special roles of Master, Pacing, and RRFP. The FCD provides this information.

Rule 3-5: Specify Master Federate in FESFA

Requirement: The Master Federate for a SpaceFOM-compliant federation execution shall be documented in the FESFA.

Rationale: Since only a single Master Federate shall exist in a SpaceFOM-compliant federation execution, it is important to know which federate will be responsible for that particular and important role. The FESFA document is specifically designated to contain this type of critical information.

Rule 3-6: Specify Pacing Federate in FESFA

Requirement: The Pacing Federate for a SpaceFOM-compliant federation execution shall be documented in the FESFA.

Rationale: Since only a single Pacing Federate shall exist in a SpaceFOM-compliant federation execution, it is important to know which federate will be responsible for that particular and important role. The FESFA document is specifically designated to contain this type of critical information.

Rule 3-7: Specify Root Reference Frame Publishing Federate in FESFA

Requirement: The Root Reference Frame Publisher Federate for a SpaceFOM-compliant federation execution shall be documented in the FESFA.

Rationale: Since only a single RRFP Federate shall exist in a SpaceFOM-compliant federation execution, it is important to know which federate will be responsible for that particular and important role. The FESFA document is specifically designated to contain this type of critical information.

3.4.2. Role and Responsibility Rules

Rule 3-8: Only the Master Federate Creates and Publishes the ExCO

Requirement: Only the Master Federate shall create and publish a single Execution Configuration Object instance with the reserved object instance name “ExCO”.

Rationale: The Master Federate is responsible for coordinating and controlling a SpaceFOM-compliant federation execution. The Master Federate does this with the ExCO. Having a single federate, the Master Federate, perform this responsibility simplifies the control logic. This is covered in detail in Chapter 7.

Rule 3-9: Only the Master Federate Manages Mode Transitions

Requirement: Only the Master Federate shall coordinate and control federation execution mode transitions using the ExCO.

Rationale: While the Master Federate can receive mode transition requests from other federates in a SpaceFOM-compliant federation execution, only the Master Federate can publish the ExCO. Mode transitions are managed through the ExCO. Therefore, only the Master Federate can manage mode transitions. This is covered in detail in Chapter 7.

Rule 3-10: Only One Master Federate in a Federation Execution

Requirement: One and only one Master Federate shall exist in a SpaceFOM-compliant federation execution.

Rationale: The principal role for the Master Federate is to coordinate and control a SpaceFOM-compliant federation execution. Having a single federate, the Master Federate, perform this responsibility simplifies the coordination and control logic.

Rule 3-11: The Pacing Federate Regulates HLA Time Management

Requirement: The Pacing Federate shall regulate the progression of HLA Logical Time (HLT) for the federation execution using a defined timing source and HLA time management mechanisms.

Rationale: SpaceFOM-compliant federation executions are designed to support the synchronized progression of time and data exchange for both reliable and repeatable results. The HLA time management services provide these capabilities to HLA federation executions in general, provided participating federates implement those time management interfaces [2]. For federation executions that run as-fast-as-possible, the slowest time regulating federate will determine the rate at which all other time constrained federates will run. The Pacing Federate is a time regulating federate. The SpaceFOM relies on the Pacing role, and the associated Pacing Federate, to tie federation execution time advancement to a defined regulating timing mechanism. This may be the Pacing Federate's Computer Clock Time (CCT) or some form of Central Timing Equipment (CTE). This is discussed in detail in Chapter 4 and in Chapter 7.

Rule 3-12: Only One Pacing Federate in a Federation Execution

Requirement: One and only one Pacing Federate shall exist in a SpaceFOM-compliant federation execution.

Rationale: The principal role for the Pacing Federate is to regulate the advancement of HLT using HLA Time Management mechanisms and therefore regulate the progression of time in a SpaceFOM complicate federation execution. In many cases, time progression is managed with respect to an external physical time source. Having more than one federate managing to different physical time sources will often lead to binding between the multiple regulating federates. Therefore, only one federate can fulfill the Pacing role.

Rule 3-13: Only One Root Reference Frame Publisher Federate in a Federation Execution

Requirement: One and only one Root Reference Frame Publisher Federate shall exist in a SpaceFOM-compliant federation execution.

Rationale: The principal role for the RRFP Federate is to provide the definition of the root frame of the reference frame tree in a SpaceFOM-compliant federation execution. Since the reference frame tree is a directed acyclic graph with a single definitive root node, having more than one root reference frame published is contradictory.

3.4.3. Data Rules

Rule 3-14: Use of Common Data Types

Requirement: A SpaceFOM-compliant FOM module shall use the existing data types defined in the SpaceFOM standard FOM modules when an appropriate data type definition is available.

Rationale: The intent of the SpaceFOM is to promote a priori interoperability for SpaceFOM-compliant federates. Redefinition of types already defined in the standard SpaceFOM modules will lead to inconsistency in data representations and incompatibilities between federates.

4. Time

At first glance, time seems like it should be a fairly simple concept. For a simulation, it is often translated into a sequentially increasing count of cycles in an execution loop. However, that is not really time, that is a count. The logical progression toward actual time would be to assign a specified quantity of time change with each execution cycle. However, what time does that represent? Is it the simulation time? What time scale does it represent? Does it always start at zero? How does that time relate to modeling the motions of physical object like the Sun, Earth and Moon? How does that time relate to the world we live in?

This section of the document presents the concepts, standards and conventions necessary to understand the time elements associated with the SpaceFOM. This includes definitions and discussions on time lines, time representations, time scale definitions, time scale relationships and time management.

4.1. Concepts of Time: Newtonian versus General Relativity

This section will define time lines, divided into discrete steps, in which all federates agree on the increment of time for each step. This is the Newtonian concept of time. Federates, however, may represent entities with differing positions in a gravity field and may travel at differing velocities and speeds relative to each other. Under the General Theory of Relativity, such entities would observe time dilation in the clocks of the other entities. Nevertheless, using current space transportation technology, time dilation tends to be on the order of microseconds per day. Given other real-world sources of inaccuracies in coordinated operations, such as the accuracy of on-board oscillators used to clock the periodic execution of avionics, time dilation does not become significant except over long spans of time that are not typical of distributed space simulations. This does not mean, however, that federate models cannot adjust for time dilation, especially when establishing initial state. In some situations, it may be necessary. For example, ephemeris models must account for relativistic effects to remain accurate for time ranges spanning decades. The Newtonian approximation to propagating time in the federation simply imposes Newtonian time on federate processing and interaction.

4.2. Time Lines

Most discussions about time start with the question: What time is it? However, the first question about time in simulation should probably be: What time line is it? There are a number of active time lines in a simulation and even more when working with distributed simulations. This section discusses and defines the principal time lines associated with a SpaceFOM based distributed simulation.

Let us consider three forms of time lines. The first is a bidirectional infinite time line that extends left and right from a line position called the present (see Figure 4-1). All points to the left are considered to be in the past and extend off to negative infinity. All points to the right are considered the future and extend off to positive infinity. The second is a unidirectional infinite time line with a starting point or epoch (see Figure 4-2). Like the bidirectional time line, there is a past, present and future; however, the past is limited to the beginning, or epoch, of the time line. The third time line is a finite line segment with a beginning (epoch), past, present, future and end (see Figure 4-3).

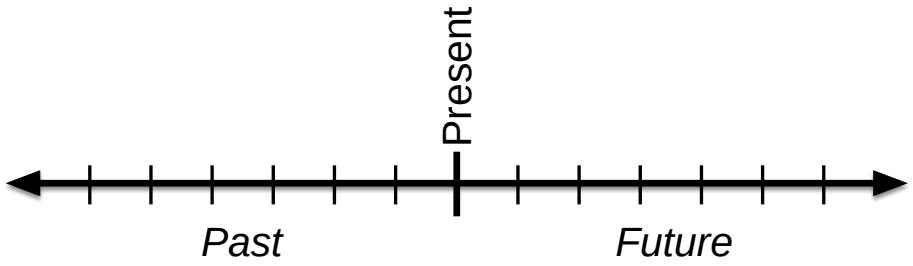

Figure 4-1: Bidirectional Infinite Time Line

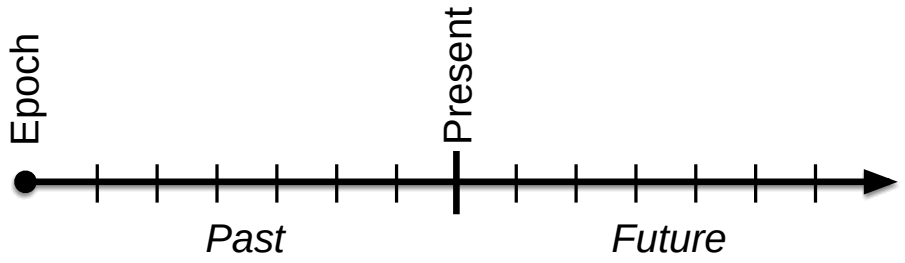

Figure 4-2: Unidirectional Infinite Time Line

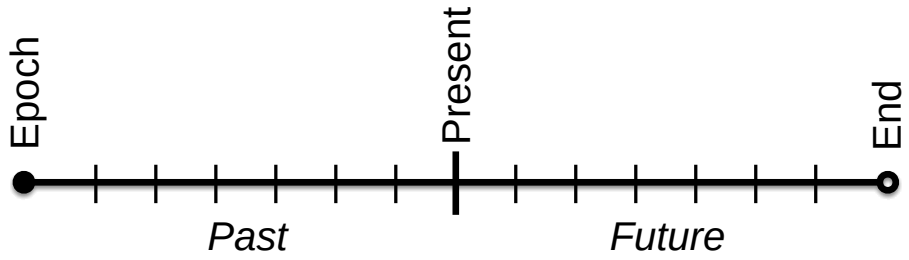

Figure 4-3: Finite Time Line

We can use these line models of time to discuss various important concepts of time when applied to modeling physical systems in a numerical computer simulation.

For this discussion, we will consider 6 principal time lines: Physical Time (PT), Computer Clock Time (CCT), Simulation Elapsed Time (SET), Simulation Scenario Time (SST), HLA Logical Time (HLT) and Federation Scenario Time (FST). It is important to understand the similarities and differences in these time lines when developing a physics based distributed simulation using the SpaceFOM. The time lines are described in the following text and illustrated in Figure 4-4.

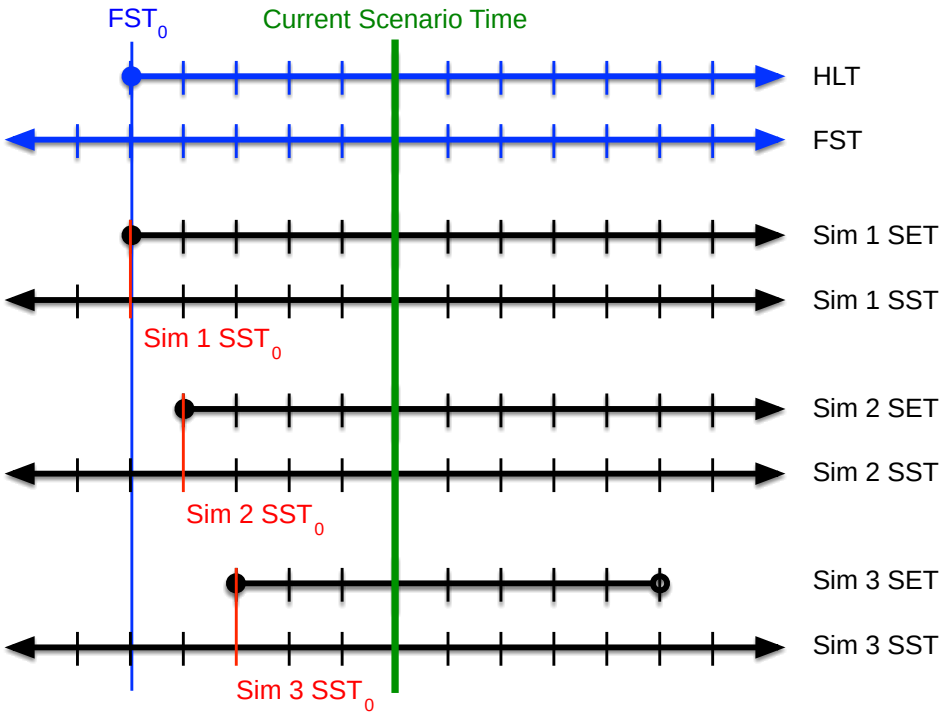

Figure 4-4: Relationships Between Time Lines

4.2.1. Physical Time (PT)

Modern definitions of time are based on the relativistic concept of spacetime. Physical time is the non-spatial dimension associated with our spacetime continuum in which events are ordered in irreversible succession from the past to the present to the future. This is the fourth dimension in a spacetime reference frame; the other three being the spatial dimensions representing position. For a given observer, physical time can be conceptualized as the bidirectional infinite time line in Figure 4-1. This is the time line in which we live and work. This is sometimes referred to as “real world” time. The SpaceFOM treats physical time under the classical Newtonian concept of absolute time. As explained in Section 4.1, this simplification works well for distributed space simulations utilizing modern space transportation technologies since relativistic effects become significant only over very large time scales.

Any epoch or relative rate of time passage is assigned as attributes to the time scale associated with the time line (see Appendix Section D.1).

4.2.2. Computer Clock Time (CCT)

Unfortunately, physical time cannot be measured as an absolute value. The best that we can do is model the passage of time, usually with some form of oscillator. We count the oscillations with respect to some arbitrary epoch and use that as our model of time. This is often referred to as “Wall Clock” time (WCT). This is how a computer “measures” time. Since any clock, including a computer, counts from some defined starting point (epoch); the unidirectional infinite time line is the corresponding representation here.

Note the following example of the time maintained by a computer running a Unix based operating system. The zero point or epoch for a POSIX compliant Unix system is 00:00:00 Coordinated Universal Time (UTC), Thursday, 1 January 1970 [19]. The system clock then maintains a count of clock ticks since this epoch. These clock ticks are associated through calibration to a corresponding amount of time in SI seconds. The operating system date and time services are based on algorithms using the epoch and the count.[1]

4.2.3. Simulation Elapsed Time (SET)

Simulation elapsed time is the time measure associated with an individual simulation starting at zero and advancing monotonically in quantifiable steps. With cyclic executives, this is often based on some integer or floating-point counter. The counter is incremented by a predetermined amount on each cycle of the executive. This is sometimes referred to as executive time. Note that discrete event simulations usually make non-uniform steps in time. A simulation with no specified stop time would correspond to the unidirectional infinite time line in Figure 4-2. A simulation with a specified stop time would correspond to the finite time line in Figure 4-3.

It is important to note that, at this point in the discussion, there is no substantive correlation between the passage of Simulation Elapsed Time, Computer Clock Time or Physical Time. Any correlation of Simulation Elapsed Time to the passage time in the real world will be established in the time management policies (see Section 4.4). Simulation Elapsed Time will progress at whatever rate the simulation is capable of running on a given computer.

4.2.4. Simulation Scenario Time (SST)

Simulation Scenario Time is a model within a simulation that associates the Simulation Elapsed Time with a representation of the problem’s Physical Time. This model may provide mappings to and between multiple physical time scales (see Appendix Section D.1). However, this is not the same Physical Time in the real world. This is the Physical Time in the problem space; which may be in the past, present or future.

[1] The date and time services are also functions of leap second adjustments, which are often incorporated with a look-up table. Simulations might ignore a leap second that occurs after simulation start.

Simulation Scenario Time is sometimes referred to as dynamic time when used as the independent variable in a numeric state propagation. This sometimes corresponds to an Ephemeris Time when used to "look-up" a priori state information for an entity in the federation execution (e.g., the position and velocity of Mars at a given time).

Simulation Scenario Time is related to Simulation Elapsed Time by the starting modeled physical time, or epoch, of the simulation. Designating the simulation epoch as SST_0 results in the following relationship between Simulation Scenario Time and Simulation Elapsed Time:

$$SST = SST_0 + SET$$

4.2.5. HLA Logical Time (HLT)

HLA Logical Time is the time line used by HLA to order messages, regulate execution time advance (TAR & TAG), and enable deterministic behavior in distributed simulation. Often, HLA Logical Time has a starting epoch of 0 ($HLT_0 = 0$); while this is not an HLA requirement, it is a SpaceFOM requirement. HLA Logical time can be thought of as a federation wide Simulation Elapsed Time. However, HLA Logical Time and individual federate Simulation Elapsed Time will not correspond if the federate is a late joiner.

HLA logical time has units of microseconds and is represented as an HLAInteger64Time data type. HLAInteger64Time is equivalent to HLAInteger64BE (big-endian 64-bit integer). This gives HLT a range of approximately ±292,271 Julian years. This resolution and range should accommodate all space simulations based on Newtonian mechanics.

4.2.6. Federation Scenario Time (FST)

Federation Scenario Time is a conceptual time associated with the physical systems being modeled in the participating federates in the federation execution. Federation Scenario Time is conceptual in that it is never computed anywhere in the federation execution but is implied by the Simulation Scenario Time of the individual federates and their time management schemes. This time should also be related to the HLA Logical Time by a constant offset FST_0.

$$FST = FST_0 + (HLT/1000000)$$

This time should match Simulation Scenario Time across all federates within the federation execution to within the accuracy of the time management mechanisms.

4.3. Time Representations

The preceding section on time lines helps to identify what time is being considered. This section discusses and defines how time is represented in a simulation time line. Just as distance can be measured from different starting points and with different units (e.g., meters vs. feet), time can be measured from different starting points (epochs) and with different units (seconds, angles, or days).

4.3.1. Epochs

At its essence, a time is a count in some specified unit from some arbitrary starting point. The starting point is called an epoch. It is a specified point in time, or origin, of a specified time line (see Figure 4-2). There are a number of important epochs, or starting dates, used by various time scales. These are usually defined as an attribute of the time scale.

There are a number of important epochs for the various time lines in a SpaceFOM-compliant federation execution. Five particularly important epochs are the Federation Scenario Time epoch (FST_0), the Simulation Scenario Time epoch (SST_0), the HLA Logical Time epoch (HLT_0), the Computer Clock Time epoch (CCT_0), and the Central Timing Equipment epoch (CTE_0) (see Section 4.2).

Federation Scenario Time Epoch (FST_0): The Federation Scenario Time epoch corresponds to the instant in FST that the federation execution begins (see Section 4.2.6 and Figure 4-4).[2]

Simulation Scenario Time Epoch (SST_0): Each federate in the federation execution will also have a Simulation Scenario Time epoch, which corresponds to the instant in SST when the federate joins the federation execution (see Section 4.2.4 and Figure 4-4).[3]

HLA Logical Time epoch (HLT_0): The HLA Logical Time epoch corresponds to the instant in HLT that the federation execution begins and is defined to be zero ($HLT = 0$) for SpaceFOM-compliant federation executions.[4]

Computer Clock Time epoch (CCT_0): The Computer Clock Time epoch corresponds to the instant in CCT that marks the reference point for regulating the passage of SST with respect to PT (real-time). This is really only relevant for the Pacing Federate. Note that this epoch is reset between federation run-freeze-run mode transitions.

Central Timing Equipment epoch (CTE_0): In cases where multiple SpaceFOM-compliant federates are running on widely dispersed computational platforms and require highly accurate coordinated real-time execution, CTE may be employed. CTE provides a specialized form of CCT that is coordinated to high accuracy between distributed computational platforms. For these cases, there is a specialized CCT_0 called the CTE epoch and designated CTE_0. Like CCT_0, CTE_0 has to be reset between run-freeze-run mode transitions. Unlike CCT_0, CTE_0 is relevant to every federate in the federation execution that needs to use CTE.

4.3.2. Time Stamps and Time Tags

Like an epoch, a time tag or a time stamp also represent an instant in time associated with some specific event. Specifics about the data representations for these are discussed in more detail in Section 8.6.

Time Stamp: For the purposes of the SpaceFOM, a Time Stamp is defined as an instant in HLT associated with an event in the federation execution. In general, time stamps are the HLT values assigned to an HLA attribute update using the HLA time management interfaces.

For example, this functionality can be used to account for network transport delays in exercises where federates are synchronized to a common external clock. For time-regulated federates, HLA's Time Management mechanisms provide for controlling the advancement of each federate to deliver information in a causally correct and ordered fashion. While this makes the time stamp redundant for time regulated federates, the inclusion of a time stamp, particularly in dynamics related attributes, provides a reliable mechanism for data consistency regardless of the federate's time management status. Therefore, all federates shall transmit this field, even if they do not use it themselves, so that other federates can use its value.

Time Tag: For the purposes of the SpaceFOM, a Time Tag is defined as an instant in time associated with an event in the federation execution that corresponds to a known FST (a known SST for a specific federate). In general, time tags are used as the time values in SpaceFOM object class attributes for time.

The SpaceFOM relies on epochs, time tags and time stamps to mark instants in time that are needed to correlate time lines and events that occur within a federation execution and across the participating federates.

---

[2] All scenario times in a SpaceFOM-compliant federation execution are expressed in the Terrestrial Time (TT) time scale.
[3] SST_0 and FST_0 will be the same for early joining federates and different for late joining federates.
[4] All HLA Logical Times in a SpaceFOM-compliant federation execution are represented as HLAinteger64Time datatypes expressed in microseconds since the start of the federation execution.

4.3.3. Seconds vs. Angles

For the purpose of the SpaceFOM, the fundamental measure of time will be the SI second. The SI second is currently defined as the duration of 9,192,631,770 periods of the radiation corresponding to the transition between the two hyperfine levels of the ground state of the cesium 133 atom at 0°K at the surface of the Earth's geoid [28].

While this may seem an obvious choice, there are measures of time based on angles. This makes more sense when you consider the astronomical origins of time keeping. The original time keeping mechanisms are the Sun, Moon and Earth. The Sun defines the cyclic nature of seasons: spring, summer, winter and fall. The Moon defines the cyclic passing of the months. The Earth's rotation defines a day and is used to track the cyclic passing of the hours in a day. These, and the Earth in particular, are based on angles of rotation.

However, modern technologies, particularly space systems, require higher levels of accuracy and uniformity than are available in the angle based systems. Therefore, the SpaceFOM uses SI seconds.

4.3.4. Julian Dates

Historically, astronomers have used a time measure based on days instead of seconds. This time measure is called the Julian day. A Julian day number is the number of days that have elapsed since the Julian epoch. The Julian epoch is noon on January 1, 4713 BC, in the Julian calendar or noon on November 24, 4714 BC, in the Gregorian calendar [28]. Note that the Julian day starts at noon and not midnight. This makes sense for an astronomer's time measure since observations are typically taken at night. This presents a continuous time measure across midnight; since, midnight is in essence the middle of an astronomer's day.

This makes the Julian day number for the day starting at 12:00 UT on January 1, 2000, the Julian Day Number 2,451,545 or JD 2,451,545. Hours past noon are represented as fractional parts of a day. So 18:00 UT on January 1, 2000 is JD 2,451,545.25.

4.3.4.1. Modified Julian Date (MJD)

One of the drawbacks to a Julian Date in terms of Julian day numbers is the size of the integer part of the day number. When representing Julian Dates as floating point numbers in a numerical computer, this can have significant impact on the accuracy for representing the fractional part or a day. If you can typically get 15 significant figures out of a double precision floating-point number, this limits the fractional part of a Julian Date to just 8 significant figures. Since there are 86,400 seconds in a day, that limits the accuracy of Julian Date representations to milliseconds. While that may be enough for many applications, it is typically not enough for many space systems applications.

To address this issue, in 1957 the Smithsonian Astrophysical Observatory proposed a Modified Julian Date (MJD). This is the Julian Date less 2,400,000.5 days. The epoch for MJD is 00:00:00 Nov 17, 1858. This reduced the size of the numbers and increased the accuracy of computations. This also changed the 0 hour from noon to midnight. This makes the Modified Julian Date for the day starting at 12:00 UT on January 1, 2000, the Julian Day Number 2,451,545.0 or MJD 51,544.5. This allows for sub-millisecond accuracies.

4.3.4.2. Truncated Julian Date

In 1979, NASA proposed and adopted the Truncated Julian Date (TJD). This is the Julian Date less 2,440,000.5. The epoch for TJD is 00:00:00 May 24, 1968. This further reduced the size of the numbers and increased the accuracy of computations. This makes the Truncated Julian Date for the day starting at 12:00 UT on January 1, 2000, the Julian Day Number 2,451,545.0 or TJD 11544.5. This allows for microsecond accuracies.

4.3.5. Time Scales

The preceding sections address what time is being considered and how it is being represented but does not define absolute time. A time scale is used to define absolute time. The Terrestrial Time (TT) time scale is used to represent absolute time for all SpaceFOM scenario physical time stamps. However, based on the technologies and needs of the historical time periods, a number of different time scales have been defined. These alternate time scales may also be important within federates in a federation execution. Appendix D provides brief descriptions of common standard time scales and their relationships to one another. These are important to federates that require a time scale other than TT.

4.4. Time Management

The SpaceFOM standard supports time stepped simulation based on fixed uniform time steps. Specifically, it relies on the use of the HLA Time Management services to coordinate the time-based progression of participating federates using uniform fixed increments of HLA Logical Time. This leads to the following hierarchical categorization of time management types in the SpaceFOM:

1. No Pacing (as-fast-as-possible)
2. Pacing (regulated by a clock)
    a. Scaled Pacing
    b. Real-time Pacing
        i. Strict/Conservative Real-time (no frame overruns)
        ii. Elastic Real-time (catch-up on overruns)
            1). Limited Overruns
            2). Unlimited Overruns

The two principal categories of time management are No-Pacing and Pacing.

The No-Pacing category is also known as “As-Fast-As-Possible” (AFAP). In this case, the Pacing Federate does not regulate the advancement of time but progresses from one time step to the next without regard for the physical passage of time. In this mode, the federation execution will run as fast as the slowest time regulating federate, which may or may not be the Pacing Federate.

The Pacing category is a bit more involved in that it has a number of subcategories or options. However, in all cases, the Pacing Federate will regulate the advancement of time between one time step and the next by using some mechanism to “measure” the passage of time; this is typically some form of CCT (see Section 4.2.2).

There are two principal subcategories to Pacing: Scaled Pacing and Real-time Pacing. The Scaled Pacing category represents the case where the progression of time is regulated in some proportion to the passage of time as registered by the Pacing Federate’s CCT. The Real-time Pacing category represents the special case where the scaled time proportion is 1.0; specifically, time progresses at the same rate as the Pacing Federate’s CCT.

The Real-time Pacing category has two further subcategories: Strict/Conservative Real-time and Elastic Real-time. Strict/Conservative Real-time does not allow for any overruns.[5] This mode of operation is usually only used when critical systems are being managed by one or more federates in the federation execution. Elastic Real-time allows for overruns.

[5] An overrun is the condition when a frame of execution cannot be completed within the associated time step duration.

Finally, the Elastic Real-time category has two further subcategories: Limited Overruns and Unlimited Overruns. The Limited Overruns category permits a specified limit on the number of overruns. It will permit a federation execution to fall behind real-time for a specified period but the federation execution must catch back up to real-time within the specified overrun limit. The Unlimited Overruns category attempts to maintain a real-time rate of execution but permits the federation execution to fall behind and not catch up.

When a federation runs strict real-time or limited overruns real-time, then the FESFA shall define the response to a federate exceeding the permissible frame overruns. See the overrun sections of the FESFA and FCD templates in Appendix A and Appendix B.

This hierarchical categorization of time management leads to 5 identifiable time management types:

- No Pacing
- Scaled Pacing
- Real-time Pacing with Unlimited Overruns
- Real-time Pacing with Limited Overruns
- Strict/Conservative Real-time Pacing

This leads to the rules associated with time management and time management types defined in Section 4.5.3.

Given the time management types described above, how does a federate go about managing various aspects of time? Along with the time management types, the SpaceFOM relies on epochs, time stamps, time tags, and HLA time management services.

4.4.1. Example

At this point, it might be helpful to illustrate the relationship between the FST_0, FST, SST_0, SST, HLT_0, HLT, a time tagged event, and a time stamped event with an example. In this example, there are two SpaceFOM-compliant federates (Sim1 and Sim2) participating in a SpaceFOM-compliant federation execution. The starting epoch for this federation execution is 7000.0 seconds. Sim1 will be the first to enter the simulation and can perform all duties required to create, join, and initialize the federation execution and start advancing time. Sim2 will join into the federation late at 5.0 seconds from the beginning. At 10.0 seconds, Sim1 will generate an event that needs to be time stamped and time tagged (see Figure 4-5).

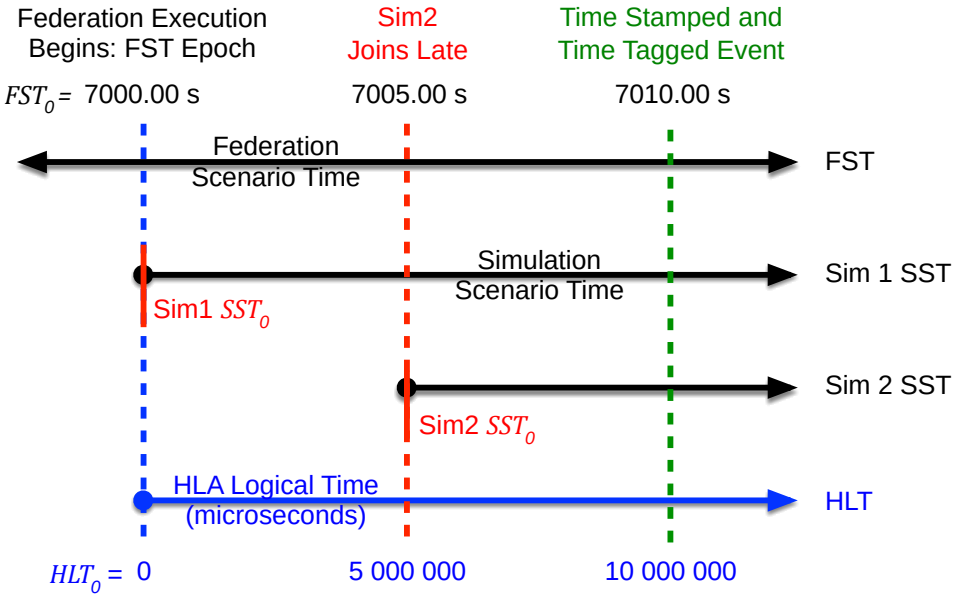

Figure 4-5: Time Lines, Epochs, and Time Stamps Example

The federation execution starts when the first simulations join and begins to advance time. In this example the first simulation to join is the Sim1 federate with a Simulation Scenario Time (SST) of 7000.0 seconds. As a result, the Federation Scenario Time (FST) for the federation execution becomes 7000.0 seconds. This results in both the Federation Scenario Time epoch (FST_0) and the Simulation Scenario Time epoch (SST_0) of Sim1 to be 7000.0 seconds. At this point, the HLA Logical Time epoch (HLT_0) is defined as zero and HLT begins from zero.

At a point 5 seconds into the federation execution, a second simulation (Sim2) joins the federation execution and begins to advance time. At this point HLT is 5,000,000 with a corresponding value of 7005.0 for both FST and Sim1's SST. Like the scenario times for the federation execution and Sim1, the SST for Sim2 is also 7005.0.[6] However, the Simulation Scenario Time epoch (SST_0) for Sim2 is 7005.0.

At a point 10 seconds into the federation execution, a published event occurs in Sim1 that needs to be time tagged and time stamped.[7] The Sim1 Simulation Scenario Time (SST) time tag for this event will be 7010.0. The corresponding HLT and associated Time Stamp will be 10,000,000.

The numerical relationships between the various time lines, epochs, time tags and time stamps explored in this example are also shown in Table 4-1.

Time Line	Federation Start	Sim2 Joins	Sim1 Event
HLT	0	5,000,000	10,000,000
FST_0	7000.0	7000.0	7000.0
FST	7000.0	7005.0	7010.0
Sim1 SST_0	7000.0	7000.0	7000.0
Sim1 SST	7000.0	7005.0	7010.0
Sim2 SST_0		7005.0	7005.0
Sim2 SST		7005.0	7010.0
Time Tag			7010.0
Time Stamp			10,000,000

Table 4-1: Time Lines, Epochs and Time Stamps Example

4.4.2. Time Steps

Time steps are an important concept in the SpaceFOM. This section defines different types of time steps, how they are related, and rules for selecting them.

Here are a few prerequisite definitions:

Simulation: An application that mimics the behavior of a process or system. A simulation may also include external interfaces.

Federate: The federated behavior of the simulation, exposed through the external HLA interface of the simulation. Also see detailed definition in Section 1.5.

Pacing Federate: A specialized SpaceFOM federate that actively manages the advancement of HLA logical time during a SpaceFOM-compliant federation execution.

[6] FST and SST are the same within the limits of the HLA time management services.
[7] Time stamps are stored/recorded in SST.

From the preceding definitions, a federate is essentially a simulation. A common type of simulation, and the type referred to throughout this document, is a continuous time numerical simulation. However, most continuous time numerical simulations are not really continuous in time. A simulation approximates continuous time by choosing to evaluate the simulation state at discrete instants of time separated by small steps in time. The simulation uses numerical propagation algorithms to compute the state at the next time step from the state at the current time step. The performance and fidelity of the simulation is often directly dependent on the selection of the simulation time step.

Here are some important time step concepts and definitions:

Simulation Time Step: The native time step used inside a simulation when state is propagated.[8] This corresponds to the smallest quantum time step in a simulation. Complete simulation states are only available at simulation time steps.

Federate Time Step: The difference between the next desired HLT and most recently granted current HLT when performing a Time Advance Request (TAR).

Federation Time Step: The Federate Time Step of the Pacing Federate.

Lookahead: A nonnegative value that establishes a lower value on the timestamps that can be sent in timestamp order (TSO) messages by joined time regulating federates. Each joined time-regulating federate must provide a lookahead value when becoming time regulating [2].

Greatest Available Logical Time (GALT): The greatest HLA logical time to which the HLA RTI guarantees it can grant an advance without having to wait for other joined federates to advance [2].

Least Common Time Step (LCTS): The least common value of all the federate time steps for the time regulating and/or time constrained federates in a federation execution. This value is set by the Master Federate, published in the ExCO, and does not change during the federation execution. The LCTS is used in the computation to find the next HLA Logical Time boundary ($HLTB$) available to all federates in the federation execution.

HLA Logical Time Boundary (HLTB): An absolute HLA Logical Time value common to the collection of all HLA Time Managed federates in a SpaceFOM-compliant federation execution. An HLTB is a common HLT position in an execution time line that serves as a boundary condition for aligning participating federates. The basic equation for computing the next common $HLTB$ is

$$HLTB = (floor(GALT/LCTS) + 1) * LCTS$$

where $GALT$ is the Greatest Available Logical Time. Note: This implies that SpaceFOM-compliant time managed federates can only join on HLTBs and can only advance in time steps compatible with the HLTB computation.

The following picture illustrates the relationship of federation, federate and simulation time step for a sample federation:

[8] The simulation time step is also known as the dynamics rate.

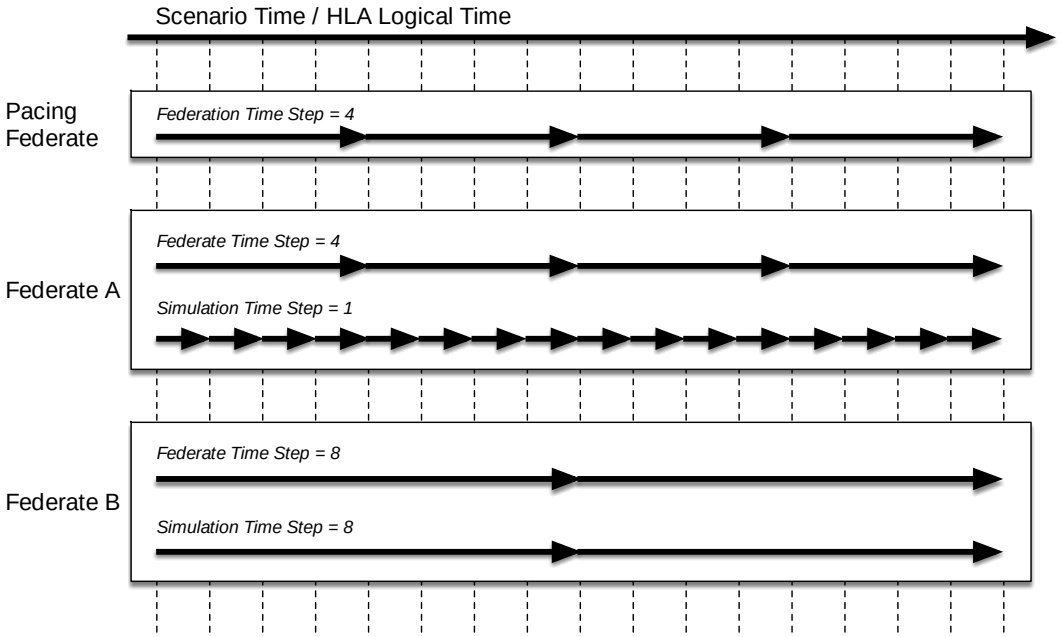

Figure 4-6: Time Steps in a Federation

The pacing federate advances the time in time steps of 4. This is the Federation Time Step. Federate A has a native Simulation Time Step of 1, but when exchanging information with other federates in the federation it uses the Federate Time Step 4. Federate B has a native Simulation Time Step of 8 and uses the same value for its Federate Time Step.

Derived rules that apply for time steps can be found in Section 4.5.3.

4.4.3. HLA Time Management

This section specifies rules when HLA Time Management is used. The following figure illustrates how HLA Time Management is used in a federate for a basic time stepped federate.

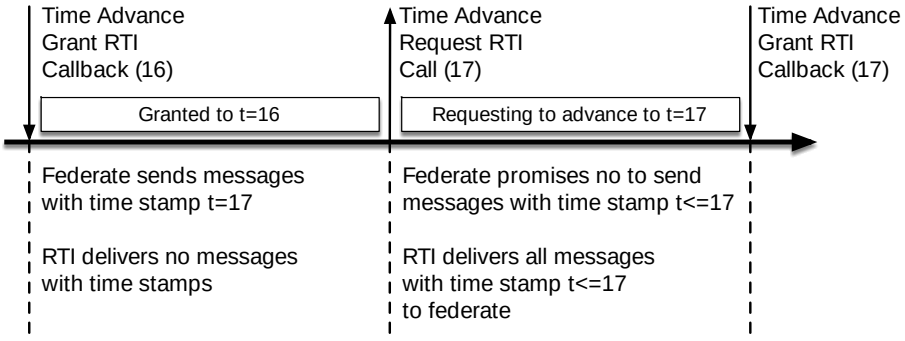

Figure 4-7: Sample Use of Time Management

A frame starts with the RTI sending the Time Advance Grant (TAG) callback to the federate, indicating that all data from other federates with time stamp 16 have now been delivered. The federate can now calculate the state and send outgoing data with time stamp 17. When the federate is done with this, it sends a TAR call to the RTI, asking to advance to time step 17. Thereby it also promises that it will not send any more data with timestamp less than or equal to 17. The RTI now delivers all incoming data with time stamp less than or equal to 17 from other federates. When there is no more data to deliver, the RTI sends a Time Advance Grant callback to the federate, since it is now safe to start calculating data for the next time step. The TAR/TAG cycle continues this way.

4.5. Rules

Based on the preceding discussion, we can infer some general and specific rules associated with SpaceFOM time, time lines, time representations and time management.

4.5.1. Time Line Rules

Rule 4-1: HLA Logical Time Starts at Zero (0)

Requirement: Early joining federates shall start HLA Logical Time at zero (0) in a SpaceFOM-compliant federation. An early joining federate shall not advance time during initialization.

Rationale: Starting the HLA Logical Time for a federation execution at zero (0) makes the time computations easier and eliminates an additional configuration parameter.

Rule 4-2: Use HLA Logical Time and Federation Scenario Time Epoch to Coordinate Time Lines

Requirement: All federates shall use HLT and the Federation Scenario Time epoch (FST_0) for coordinating time lines.

Rationale: In general, HLA Logical Time (HTL) is the underlying time line used by HLA to synchronize time constrained and time regulating federates using the HLA time management interfaces. However, Federation Scenario Time is the time line associated with the simulated scenario time of the physics based models in a SpaceFOM-compliant federate. For instance, a planetary ephemeris will not be based off of an arbitrary HLA logical time but off of a standard time scale like TT. This is the time scale for FST. By design, HLT and FST are directly tied together with the FST epoch (FST_0) using the SpaceFOM time management approach discussed in Section 4.4.

4.5.2. Time Representation Rules

Rule 4-3: Specify Federation Scenario Time Epoch in FESFA

Requirement: The Terrestrial Time representation of the Federation Scenario Time epoch (FST_0) shall be documented in the FESFA.

Rationale: Some potential participating federates may have limitations or dependencies on the time frame (start date and run duration) in which they operate. This will be important in determining which federates can operate successfully in a given federation execution. In addition, FST epoch is required for SpaceFOM time line coordination.

Rule 4-4: Document Federates Time Frame Dependencies in FCD

Requirement: A federate shall document any operational time frame dependencies or restrictions in their associated FCD.

Rationale: Some federates may have restrictions or dependencies on the time frame (start dates and run durations) in which they can operate. This needs to be documented in the federate's FCD to inform a federation execution designer of potential limits to valid epochs and run durations for a federation execution that includes that federate.

Rule 4-5: Fixed Federation Scenario Time Epoch

Requirement: The Federation Scenario Time epoch (FST_0) shall remain fixed throughout the federation execution.

Rationale: The Federation Scenario Time epoch (FST_0) is the FST time value that fixes the FST time line to the HLT time line. Changes in FST_0 during a federation execution will invalidate the necessary linkage between FST and HLT. Having FST_0 remain fixed is key to the correlation between FST and HLT and all other time lines.

Rule 4-6: Master Federate Publishes Federation Simulation Time Epoch

Requirement: The Master Federate shall publish the federation execution's Federation Scenario Time epoch (FST_0) in the ExCO.

Rationale: The Federation Simulation Time epoch (FST_0) is a critical piece of information in coordinating federate and federation execution time lines (see Figure 4-4 and Figure 4-5). It is also critical in the equations that relate various time lines. This information is required by many federates participating in a SpaceFOM-compliant federation execution. The Master Federate and the role of the ExCO are described in detail in Chapter 7.

Rule 4-7: Use Federation Scenario Time Epoch to Coordinate Simulation Scenario Time

Requirement: All federates shall use the Federation Scenario Time epoch (FST_0) to compute the relationship between HLA Logical Time (HLT), their Simulation Elapsed Time (SET), and their Simulation Scenario Time (SST).

Rationale: The FST_0 is specified as an independent attribute of each federation execution; it can be changed with each execution. This FST_0 is a key piece of information in establishing the relationship between the federation execution HLT and a simulation's SST (see Section 4.2). The only reliable way for a federate to coordinate a simulation's SST with the federation execution's FST is to use FST_0 and HLT. Note that SST_0 is derived from FST_0 at the start of the simulation where $HLT = HLT_0$. The general relationship is $SST = FST_0 + HLT_0 + SET = SST_0 + SET$.

Rule 4-8: Use of Time Stamps when Exchanging Data

Requirement: When exchanging data, federates shall use the time stamp parameter in the HLA services to specify the time for which the data is valid.

Rationale: Time stamps are useful in correlating data for federates that are not time constrained.

Rule 4-9: Time Stamp Format

Requirement: Time stamps shall be specified as HLAinteger64Time data representation and expressed as microseconds past the start of the federation execution.[9]

Rationale: Time stamps can be used to correlate data generated by federates in a federation execution. The format of the time representation needs to support the accuracy of the underlying HLA Logical Time infrastructure for a SpaceFOM-compliant federation execution. For the SpaceFOM, HLA Logical Time is also a 64 bit big endian integers and expressed as microseconds.[10]

Rule 4-10: Scenario Time Expressed in the Terrestrial Time (TT) Scale

Requirement: The time tags in SpaceFOM object class instances shall be expressed in the TT scale using the Truncated Julian Date (TJD) format.

Rationale: The intent of the SpaceFOM is to promote a priori interoperability for SpaceFOM-compliant federates. Inconsistent time representation will lead to inconsistencies and incompatibilities between federates. By requiring all times to be represented in the TT scale, this problem is avoided. The TT scale was selected as the closest time representation to a "true" dynamical time representation (see Section D.1.8). The TT time values are expressed in the Truncated Julian Date format to preserve as much accuracy as possible in a double precision floating point representation.

---

[9] HLAinteger64Time is a standard HLA time representation that corresponds to HLAinteger64BE. HLAinteger64BE is a 64 bit Big Endian integer representation [7].

[10] Although the timestamp is defined with a precision of microseconds, its accuracy may be less in a given federate, in part, because federation time does not adjust for possible time dilation due to relativity (see Section 4.1).

4.5.3. Time Management Rules

Rule 4-11: Federate Declaration of Supported Time Management Types in FCD

Requirement: A federate shall declare the time management types it supports in its FCD. One or more types may be specified.

Rationale: When designing a federation execution, it is important to understand what time management types are supported by constituent federates. The time management types supported by the federation execution will be determined by the time management types supported in its federates. A federation execution designer can use the information in the potential participating federates' FCDs to determine which federates can participate and what will be the available time management types of the resulting federation execution.

Rule 4-12: Federation Execution Specification of Supported Time Management Type in FESFA

Requirement: A federation execution shall specify the time management types it supports in its FESFA. One or more types may be specified.

Rationale: Just as a federate may support multiple time management approaches, a federation execution may also support multiple time management types of execution. It is important for the participating federates in a federation execution to understand intended time management modes of operation.

Rule 4-13: Federation Execution Contains Only Federates That Support Time Management Types

Requirement: A federation execution shall only contain federates that support the time management types specified in the FESFA.

Rationale: In order for a federation execution to successfully operate in a specified time management mode, all the participating federates must support that particular time management type.

Rule 4-14: Definition of the Federation Time Step in FESFA

Requirement: The Federation Time Step shall be agreed upon and documented in the FESFA.

Rationale: The Federation Time Step is defined by the Pacing Federate and used to control the quantized progression of HLT in the federation execution. This information is needed by other federates in the federation execution to be compliant with Rule 4-19.

Rule 4-15: Pacing Federate Uses Federation Time Step as Federate Time Step

Requirement: The Pacing Federate shall use the Federation Time Step as its Federate Time Step.

Rationale: By definition, the Pacing Federate's federate time step is the Federation Time Step.

Rule 4-16: Constant Time Steps

Requirement: All federates shall use a constant federate time step throughout a federation execution.

Rationale: The computation required for determining a common HLTB requires the knowledge of the federate time step of every federate participating in the federation execution. If federate time steps are published in the FCDs, then a Master Federate can easily compute an HLTB for mode transitions, and Late Joiner federates can compute the next common HLTB for joining a running federation execution. If federate time steps can vary during a federation execution, the evaluation process for a common HLTB becomes difficult if not impossible.

Rule 4-17: Definition and Publication of Least Common Time Step (LCTS)

Requirement: The LCTS shall be specified in the FESFA and published by the Master Federate in the ExCO.

Rationale: In order to facilitate the synchronized joining of federates and coordinated federation mode transitions on common HLT boundaries, the next available common HLTB must be computed by the Master Federate and published in the ExCO. The LCTS value is necessary for computing a common HLTB. It is used by the Master Federate to compute the next common HLTB for federation execution mode transitions and by late joining federates to compute a coordinated time to join into a running federation execution.

Rule 4-18: Relationship Between Federate Time Step and Simulation Time Step

Requirement: The Federate Time Step shall be a positive nonzero integer multiple of its Simulation Time Step.

Rationale: Given that the simulation time step is the smallest quantum unit of time advancement in a given federate and that the federate must be able to step to every federate time step, then the federate time step must be a positive non-zero integer multiple of the simulation time step. This leads to the following mathematical relationship:

$$Federate\ Time\ Step\ =\ n\ *\ Simulation\ Time\ Step$$

where n is a positive nonzero integer value ($n\ >=\ 1$). This implies the following relationship between the federate time step and the simulation time step:

$$Federate\ Time\ Step\ >=\ Simulation\ Time\ Step$$

Rule 4-19: Relationship Between Federate Time Step and Federation Time Step

Requirement: The federate time step shall be a positive nonzero integer multiple of the federation time step.

Rationale: Given that the federation time step is the smallest quantum unit of time advancement in a given federation execution and that the federate must be able to step to every federate time step, then the federate time step must be a positive non-zero integer multiple of the federation time step. This leads to the following mathematical relationship:

$$Federate\ Time\ Step\ =\ m\ *\ Federation\ Time\ Step$$

where m is a positive nonzero integer value ($m\ >=\ 1$). This implies the following relationship between the federate time step and the federation time step:

$$Federate\ Time\ Step\ >=\ Federation\ Time\ Step$$

Rule 4-20: Federate Time Management Lookahead Matches Federate Time Step

Requirement: A federate shall use the same value for the time management lookahead as for its federate time step.

Rationale: A mismatch between the federate time step and the federate's lookahead value can lead to issues with coordinating mode transitions. If the lookahead value matches the time step, then SpaceFOM-compliant federates can compute common HLA logical time boundaries on which a federation execution can go to freeze, and late joining federates can join a SpaceFOM-compliant federation execution on a coordinated time boundary.

Rule 4-21: Synchronized Federates Use HLA Time Management TAR and TAG Services

Requirement: Synchronized federates in a SpaceFOM-compliant federation execution shall advance their HLA logical time using the HLA time management TAR and TAG services.

Rationale: The HLA time management services are the only mechanism available in all time management modes to coordinate synchronized federates in a SpaceFOM-compliant federation execution. This requires that any federate that wants to be synchronized with other federates in the federation execution must use HLA time management TAR and TAG services to advance its HLA logical time.

Rule 4-22: Time Regulating Pacing Federate

Requirement: The Pacing Federate shall enable HLA Time Regulating.

Rationale: HLA requires that a federate must be time regulating in order to regulate the advancement of HLA logical time. Since that is a principal responsibility of the Pacing Federate, it must enable HLA Time Regulating.

Rule 4-23: Time Constrained Pacing Federate

Requirement: The Pacing Federate shall enable HLA Time Constrained.

Rationale: HLA requires that a federate must be time constrained for it to be constrained by the advancement of HLA logical time. Since the Pacing Federate needs to detect when the federation execution is synchronized in time through the HLA time management TAG service, it must enable HLA Time Constrained.

Rule 4-24: Publication of Time Stamp Ordered Data

Requirement: A federate that wishes to publish and produce TSO data shall enable HLA Time Regulating.

Rationale: This is an HLA requirement.

Rule 4-25: Subscription to Time Stamp Ordered Data

Requirement: A federate that wishes to subscribe to and receive TSO data with ordering shall enable HLA Time Constrained.

Rationale: This is an HLA requirement.

Rule 4-26: Time Stamp Order Delivery for Updates and Interactions

Requirement: All attribute updates and interactions in the federation shall be sent TSO unless otherwise specified in the SpaceFOM.

Rationale: In order to facilitate repeatable behavior in a SpaceFOM-compliant federation execution, all federates must be synchronized and all their updates and interactions must be sent TSO. This ensures that the data is received in order and on a synchronized time boundary.

Rule 4-27: Receive Order Delivery for Updates and Interactions

Requirement: All data in the federation related to managing the scenario execution shall be sent Receive Order (RO).

Rationale: There are sections within the SpaceFOM execution control flow that HLT is not advancing and the federate must still receive ExCO updates for mode transition management. This requires that the Master Federate send the ExCO updates out Receive Order (RO). This is also true of other federates and ModeTransitionRequest interactions.

Rule 4-28: All Federates Enable Asynchronous Delivery

Requirement: All SpaceFOM-compliant federates shall Enable Asynchronous Delivery.

Rationale: All federates need to receive SpaceFOM management interactions, which are sent Receive Order (RO). Therefore, all federates must Enable Asynchronous Delivery.

Rule 4-29: When to Call Time Advance Request

Requirement: A federate shall call TAR when it has produced all TSO data for the next frame.

Rationale: This is an HLA requirement for synchronized TSO delivery of attribute updates and interactions.

5. Reference Frames

Reference frames are a fundamental concept for representing when and where any physical entity exists in time and space. This representation is referred to as the state of the entity. In order to represent the state of something, it is necessary to express that state with respect to some time scale and some referent coordinate system. This combination of time and coordinate system is referred to as a spacetime coordinate or reference frame.

5.1. A Simple Reference Frame Scenario

To start with, we will focus on concepts of location and the role of reference frames in specifying a location. We can start the discussion of reference frames and their importance in the SpaceFOM with the following scenario: you have invited a friend from Big City to dinner at 6:00 PM at your home in Old Town.

How do you describe to someone where you live? Typically, you will describe where your home is in terms of distance and direction from some commonly known landmark. There are many ways to do this. One common approach is to use a locally orthogonal set of ordinal directions like North-South and East-West. That allows for specifying a position from the common landmark using a directional set of numbers called coordinates. For instance, you may live 2 kilometers East and 1 kilometer North of the Old Oak tree in Old Town's central square (2.0, 1.0). This would give anyone, with knowledge of the Old Oak, a direction to where you live.

What happens if your visitor is not familiar with the Old Oak? Then you will probably begin the process of determining some other commonly known landmark or reference point. In this case, you both know the intersection of Big City Highway and Old Town Road; the Old Oak is 4 kilometers East and 1 kilometer South of the intersection (4.0, -1.0). Now your visitor can find the Old Oak and then proceed from the Old Oak to your home.

These common landmarks provide reference frames used to express the location of other reference frames and ultimately your home. Another way of saying this is that the location of your home is expressed in the Old Oak reference frame. If you are communicating to someone who is not familiar with the Old Oak reference frame, then you need to provide the location of the Old Oak frame with respect to the reference frame defined by the intersection of Big City Highway and Old Town Road. This process can be continued as needed with established relationships between locally known reference frames.

Other than not arriving late for dinner, time does not typically factor into finding the location of your home with respect to the Old Oak. In most cases, the reference frames we chose for directions do not move. The reference frames are fixed with respect to one another. In contrast, for space applications, reference fames often move with respect to one another. For instance, the Moon moves with respect to the Earth. This means that in order to specify the location of something referenced in an Earth fixed frame to something referenced in a Moon fixed frame, you also need to know when. With moving reference fames, time becomes an important attribute when specifying the reference frame.

5.2. SpaceFOM Reference Frames

The preceding scenario is a two dimensional example. Space based applications typically require three dimensions. In addition to knowledge of the positional state, many applications require knowledge of the relative attitude of one frame with respect to another. This results in 3 dimensions of position (translational state), 3 dimensions of attitude (rotational state) and 1 dimension of time. These are the three constituent fields in the SpaceFOM's SpaceTimeCoordinateState fixed record data type (see Section 5.4.3). The SpaceFOM's ReferenceFrame object class is composed of this SpaceTimeCoordinateState, a unique identifying name and a name identifying the parent frame (see Section 5.4.2). The parent frame is the frame in which the state attributes (position and attitude) of the reference frame are expressed; remember how the Old Oak reference frame was expressed in the Big City Highway and Old Town Road intersection reference frame.

5.2.1. Limitations of a Single Common Reference Frame

As discussed above, the purpose of a reference frame is to provide a basis for describing the state of another reference frame or a physical entity. Specifically, the state is expressed in terms of a current time with respect to a known epoch, a position with respect to a known reference origin and coordinate basis and an attitude with respect to a known coordinate basis (usually the same coordinate basis as the positional state but not always). If all physical entities have their states expressed in a single common reference frame, then state comparisons are relatively easy. However, it is not always convenient, and sometimes numerically restricting, to express everything in a single common reference frame. Here are two simple examples.

5.2.1.1. Computational Convenience

We have a simulation that is modeling mining operations on the Moon. The simulation models the logistics chain from Earth orbit to the surface of the Moon and back. There are systems in Earth orbit, systems transiting between the Earth and Moon, systems orbiting the Moon and systems operating on the lunar surface. An inertial frame centered on the Earth is chosen as the single common reference frame.

One entity in this scenario is a simple rover on the surface of the Moon. The rover only operates within the local confines around Rima Hadley (the landing site for Apollo 15). It transports mining resources between the excavation sites and a supply depot. As a result, the rover only needs to know about things within the lunar fixed reference frame around Rima Hadley.

However, in order for the rover simulation to express its state in the common Earth centered inertial reference frame, it will have to model the rotation of the Moon on its axis and model the orbit of the Moon about the Earth. Neither of these is required for the rover to model its tasks at Rima Hadley. The use of a single common reference frame adds considerable computational complexity and inconvenience to an otherwise simple rover simulation. The lunar fixed reference frame is a more natural reference frame for modeling the rover. Likewise, the lunar fixed reference frame does not make sense for use by some of the other elements in this scenario. They may have their own natural reference frames. In this case, convenience supports abandoning the use of a single common reference frame.

5.2.1.2. Computational Precision

We have a simulation that is modeling a space vehicle that transits from the Earth to Mars. The vehicle starts out docked with a supply ship in lunar orbit, departs the Earth/Moon system, transits to Mars, enters Mars orbit, and rendezvous with a return vehicle in Mars orbit. What would be the common reference frame for this scenario? We could choose the Earth, Mars or perhaps the solar system barycenter. Any of these are valid reference frames. As a start, we can decide to model everything in an Earth inertial reference frame. That is where the mission begins with the vehicle docked to the supply ship in lunar orbit.

We will start with the assumption that the state data are represented in the model with double precision floating point numbers. In general, a double precision floating-point number has 15 decimal places of precision. The characteristic distance from the Earth to the Moon is about 384,400 km (238,900 mi) or 3.844×10^8 meters. That means that 9 digits are required for the integer meters with 6 digits remaining for decimal meters. So, for Earth/Moon distances, a double precision floating-point point number can provide a precision of 10^{-6} meters. However, the distance from the Earth to Mars ranges from about 55 million km (5.5×10^{13} m) to about 400 million km (4.0×10^{14} m). That only leaves 1 decimal remaining for decimal meters. That means that the state of a vehicle in Mars orbit would only have precision to 10 cm. For most space applications, 10 cm of precision is not enough.

If we repeat the same process for any of the other frames, we have similar results. If we chose a Mars inertial frame, the reduced precision is on the Earth end of the mission profile instead of the Mars end. If we choose a solar system barycenter inertial frame, we are limited to about 1 cm of precision throughout the transit. So, regardless of our choice of reference frame, we will be limited to 1 cm of precision. Again, for most space applications, 1 cm of precision is not enough.

In general, for space applications, you want the highest precision possible. Given the limitations described above, how can we achieve better precision? We can start by looking at local precision instead of global precision. When operating at Earth/Moon distances, we have already established that we have 10^{-6} m (10^{-4} cm) of precision. When at Mars, if we perform computations in the local Mars inertial frame, we can get 10-8 m (10-6 cm) precision in Mars orbit. However, we are still limited to 1 cm precision when comparing entities in the Earth/Moon system with entities in the Mars system.

How does this help us? It helps by giving us a method for maintaining local precision for objects in the proximity of the local reference frame. The link between characteristic distance and numerical precision can be used to define what constitutes an acceptable “proximity”. In this case, maintaining computational precision supports abandoning the use of a single common reference frame.

If we no longer have a single common reference frame, how can we relate the state of an entity expressed in one local reference frame with another entity expressed in a different local reference frame? The short answer is by establishing and maintaining the relationships between the local reference frames. The SpaceFOM does this by establishing Reference Frame Trees.

5.2.2. Reference Frame Trees

The SpaceFOM’s representation of reference frames is intended to provide an unambiguous method for defining the relationship between any arbitrary set of reference frames in a SpaceFOM-compliant federation execution. This is accomplished through the use of the parent_name attribute in the ReferenceFrame object class to form a specialized type of directed acyclic graph (DAG) called a directed rooted labeled tree. This reference frame tree is formed from a single base root node with directed paths from an arbitrary number of immediate child nodes. These child nodes can then have directed paths from other arbitrary sets of child nodes (see Figure 5-1).

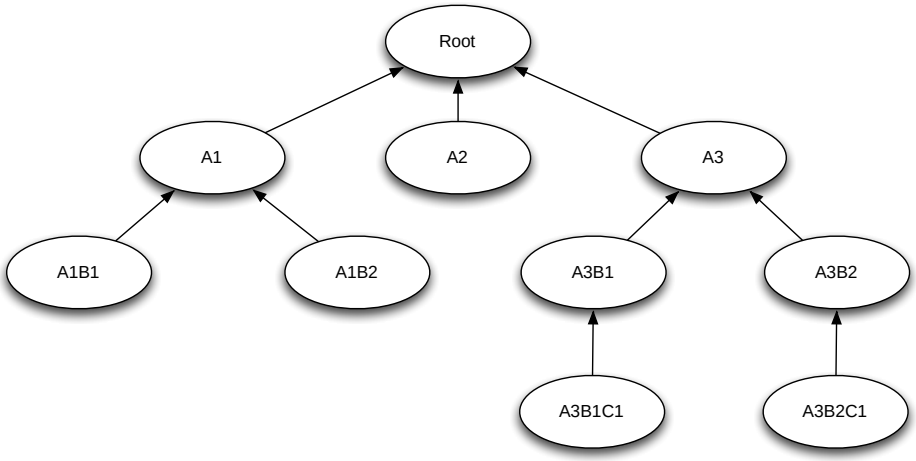

Figure 5-1: Directed Rooted Labeled Tree

A relationship between any two nodes in the tree can be established by walking the tree from the nodes in question until a common node is identified. Because of the directed acyclic nature of the tree, a common node will always exist (see Figure 5-2).

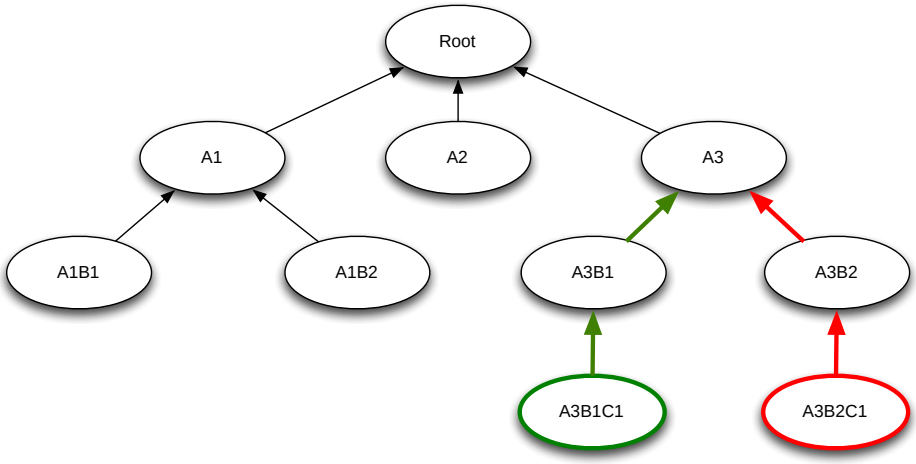

Figure 5-2: Traversing A Tree

Once the path between intermediate reference frames is established, the known directional relationships between those frames can be used to formulate transformations between the frames of interest.

In the SpaceFOM, the reference frames are often associated with the celestial bodies of interest to the particular federation execution. A simple example reference frame tree for a federation execution that models a trip to Mars might look like the one in Figure 5-3.

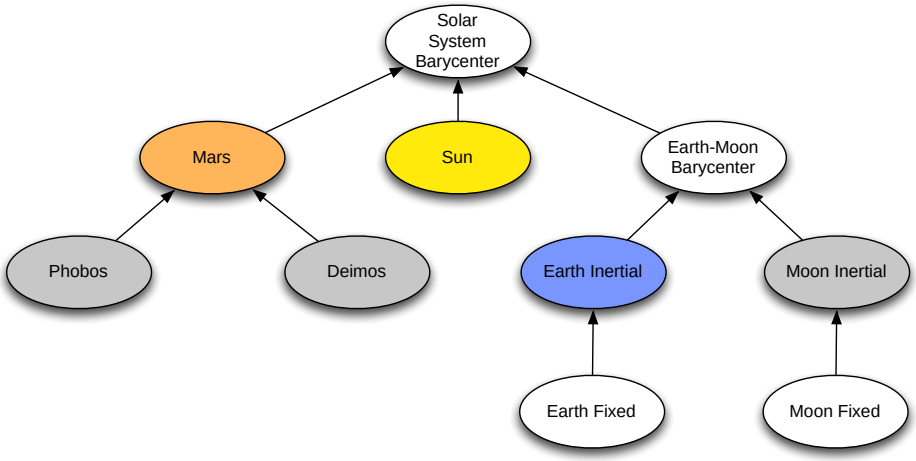

Figure 5-3: Example Reference Frame Tree

This tree provides the necessary information to transform to or from any given reference frame in the tree from or to any other frame in the tree.

5.2.3. Reference Frame Transformations

Each bubble in the reference frame tree diagrams above corresponds to a SpaceFOM ReferenceFrame object class instances (see Section 5.4). The constituent data include a unique name for the reference frame, the name of the parent frame in which this reference frame's state is expressed, and the actual state in the form of a SpaceFOM SpaceTimeCoordinateState fixed record data type (see Section 5.4.3). The corresponding state consists of translational state information, rotational state information, and the time to which the state corresponds.

The translational state data are in a SpaceFOM fixed record data type ReferenceFrameTranslation (see Section 5.4.3.2). This provides a position vector from the origin of the parent reference frame to the position of this reference frame. It also provides a velocity vector for the motion of this reference frame with respect to the parent frame. Both of these vectors are expressed in parent reference frame. These can be used to describe the translational position and motion of this frame with respect to its parent.

Similarly, the rotational state data are in a SpaceFOM fixed record data type ReferenceFrameRotation (see Section 5.4.3). This provides an attitude quaternion that describes the attitude of this reference frame with respect to its parent frame. It also provides an angular velocity vector that describes the rotational motion of this reference frame with respect to the parent frame expressed in this frame's coordinates. It is important to note that the angular velocity vector is expressed in the subject frame's coordinate and NOT the parent frame's coordinates. These can be used to describe the attitude and rotational motion of this frame with respect to its parent.

This translational and rotational information can be used to transform a vector expressed in a given reference frame into a vector expressed in its parent frame. In turn, the vector now expressed in the parent frame can be expressed in the parent's parent frame or in another child frame of the parent frame. Chaining together sequences of transformations using the relationships established in the reference frame tree allows for transformation between any two frames in the reference frame tree.

So, how are these reference frame transformations formulated and how do they work? The following equation defines the transformation of a position vector expressed in a child reference frame into a position vector expressed in the parent reference frame:

$$\vec{\boldsymbol{r}}_{parent} = \vec{\boldsymbol{r}}_{0\_parent} + \widetilde{\boldsymbol{Q}}(\vec{\boldsymbol{r}}_{child})$$

See Appendix E for a more detailed discussion of reference frame transformations.

5.3. SpaceFOM Reference Frame Naming Conventions

While the preceding sections cover the general motivations and mechanics for the use of reference frames, there remains a question on how to distinguish between a potential proliferation of dependent reference frames. Specifically, what policy should be used to name references frames? There are at least two important characteristics for distinguishable reference frame names: that the names be unique and that the names be descriptive.

This section provides a set of syntax rules for defining well-structured reference frame names suitable for use in a SpaceFOM-compliant federation execution. The following set of rules are defined using Extended Backus-Naur Form (EBNF) notation:

Non-terminals	Written as italicized text.
Terminals	Written in quotes, as in "Earth".
|	The left and right sides are alternatives.
( )	Parentheses are used for grouping.
[ ]	Brackets enclose optional symbols.
{ }	Braces enclose symbols to be repeated 0 or more times.
=	Defines symbol on left by notation on the right.
.	Ends each definition.
spaces	Used only for readability.

ReferenceFrameName = *FrameOrigin FrameOrientation.*

FrameOrigin [11] = *Origin* [“Centric” | “Centered”] | “SolarSystemBarycentric” | *CelestialBody* {*CelestialBody*} [“Barycentric”].

FrameOrientation [12] = [*Origin*] (“Fixed” | “Inertial” | “Rotational” | “Rotating”) [*AxesType*] | *AxesType*.

Origin = *CelestialBody* | *LagrangianPoint* | *HumanMadeBody* | *UserDefinedOriginName*.

CelestialBody = *SolarSystemBody* | *ExtraSolarSystemBody*.

SolarSystemBody = “Sun” | *SolarSystemPlanet* | *SolarSystemDwarfPlanet* | *SolarSystemNaturalSatellite* | *SmallSolarSystemBody*.

SolarSystemPlanet = “Mercury” | “Venus” | “Earth” | “Mars” | “Jupiter” | “Saturn” | “Uranus” | “Neptune”.

SolarSystemDwarfPlanet = “Pluto” | “Ceres” | “Haumea” | “Makemake” | “Eris” | <valid name from the IAU (International Astronomical Union) DB>.

SolarSystemNaturalSatellite = “Moon” | “Lunar” | <valid name from the IAU (International Astronomical Union) DB>.

---

[11] The term "Centric" or "Centered" is assumed to mean the center of mass, though this may be too limiting to describe reference frame origins for a *HumanMadeBody* or irregular *SmallSolarSystemBodies* (e.g., 433 Eros). If the Origin is a body and not a point, then the Origin's center of mass is the frame origin regardless of whether the term "Centric" or "Centered" follows, thus "Earth" and "EarthCentered" mean the same thing. In the third option of the rule, if just one celestial body is provided with "Barycentric", then it is interpreted as the barycenter of the named body and its satellites, e.g., SaturnBarycentric would be the barycenter of Saturn, its rings, and its moons. The non-terminal *CelestialBody* is used rather than the non-terminal *Origin* for Barycentric origins because not all *Origin* options can form a barycenter, e.g., Lagrange points. Although it is possible to create a barycentric reference system from two human made bodies, the rules assume that such a reference frame is rarely used.

[12] This rule states that the *FrameOrientation* must consist either of the term "Fixed", "Inertial", "Rotational", or "Rotating" optionally accompanied by an *Origin* and/or an *AxesType* OR only an *AxesType*. This rule therefore has the following assumptions:

1) If "Fixed" et. al. is used without an *Origin*, then it refers to the *FrameOrigin*, e.g., "EarthFixed" and "EarthCenteredEarthFixed" have identical meaning;
2) If the *AxesType* is not used, then there must be an unambiguous dominant standard for axis directions associated with the *FrameOrigin* or *Origin*, or the governing document (SpaceFOM or federation agreement) must unambiguously define the reference frame. Thus, one can use the name "EarthCenteredEarthFixed" without further elaboration because there exists a dominant standard for Earth's geographic axes. However, one could not use "EarthCenteredInertial" without further defining it, because there are numerous definitions for inertial axes that can be placed at the Earth Center. If the naming rules can produce an unambiguous name, then that name should be given preference;
3) An *AxesType* unambiguously defines both frame orientation and the direction of the three orthogonal Cartesian axes. Thus, "EarthCenteredJM2000Eq" and "EarthCenteredInertialMJ2000Eq" are equivalent because the axes type MJ2000Eq defines both an inertial orientation and the direction of the three axes.

SmallSolarSystemBody = "67P/Churyumov-Gerasimenko" | "Hale-Bopp" | <valid name from the IAU (International Astronomical Union) DB>.

ExtraSolarSystemBody = <valid name from the IAU (International Astronomical Union) DB>.

LagrangianPoint = L1 | L2 | L3 | L4 | L5.

HumanMadeBody [13] = *String.*

UserDefinedOriginName = *String.*

String = "a..z,A..Z,$,_"{"a..z,A..Z,$,_,0..9, Unicode character over 00C0"}.

AxesType [14] = "MJ2000Eq" | "MJ2000Ec" | "ICRF" | "ITRF" | "MODEq" | "MODEc" | "TODEq" | "TODEc" | "MOEEq" | "MOEEc" | "TOEEc" | "ObjectReferenced" | "Equator" | "BodyFixed" | "BodyInertial" | "GSE" | "GSM" | "Topocentric" | "Topodetic" | "BodySpinSun" | *UserDefinedAxisTypeName.*

UserDefinedAxisTypeName = *String.*

A set of well-known or standard reference frames along with their associated semantics are presented in Appendix section F.1. The Axis Types listed in the naming convention above are defined, along with their associated semantics, in Appendix section F.3.

5.4. Reference Frame Representation

Having described the concepts and mathematical backing for the "*where*" represented by a reference frame, we will now define the actual data types used to define a ReferenceFrame object class in the SpaceFOM.

There are three principal top-level attributes that define a ReferenceFrame: name, parent and state.

5.4.1. Name

Name - The principal identifier for a ReferenceFrame is a required *name* attribute string that must be unique throughout the federation execution. This name must correspond to the object class instance name in the federation execution. This will ensure that it is unique throughout the federation execution. The name attribute is used to identify an instance of a ReferenceFrame distinct from any other ReferenceFrame.

---

[13] The reference frames associated with *HumanMadeBodies* can follow additional conventions than those currently implied by the *Frame Origin* rule. Indeed, in that rule, the term "Centric" or "Centered" is assumed to mean the center of mass; however, reference frames for human made bodies may use the geometric center or an arbitrary structural reference as their center. This would require the inclusion of additional qualifiers other than "Centric" such as "SRF" (Structural Reference Frame).
[14] For the semantics of the different Axes Types see Table F-2.

5.4.2. Parent Reference Frame

Other than the root reference frame in the reference frame tree, a reference frame is expressed with respect to a parent reference frame.

Parent Name - A ReferenceFrame object class has a required *parent_name* attribute. This is a string representation of the name of this frame's parent reference frame. If this frame has no parent (i.e., is a 'root' reference frame), then this string must be empty; otherwise the non-empty string must correspond to a name attribute of some other ReferenceFrame object instance in the federation execution.

5.4.3. SpaceTimeCoordinate State

The state of a reference frame is defined with respect to its parent reference frame (see Section 5.4.2).

State – A ReferenceFrame object class has a required *state* attribute. The state attribute is of type SpaceTimeCoordinate. Specifically, the state of one reference frame is expressed with respect to a parent reference frame in terms of time, translational state, and rotational state. The SpaceFOM requires right-handed coordinate systems for ReferenceFrame instances and SpaceTimeCoordinateState elements (see Appendix C and Appendix F for details).

5.4.3.1. Time

The time attribute of the SpaceTimeCoordinate data type is described in this section. See Chapter 4 for a more detailed discussion of time.

Time – A SpaceTimeCoordinate has a required *time* attribute. This value serves as a 'time tag' that specifies the simulated time (TT) to which the attributes values correspond. It may be used by federates that do not use HLA time management but still need to know when the attributes were valid. For example, a plotting federate that isn't time regulating or time constrained would need the time tag in order to plot a time series.

5.4.3.2. Translational State

The SpaceTimeCoordinate data type contains a required *translational_state* attribute that is of type ReferenceFrameTranslation. The ReferenceFrameTranslation data type contains two attributes, one for position and one for velocity.

Position - A ReferenceFrameTranslation has a required *position* attribute. This is a 3-vector that specifies the position of the frame origin with respect to the parent reference frame. The components of this vector are resolved onto the coordinate axes of the parent frame.

Velocity - A ReferenceFrameTranslation has a required *velocity* attribute. This is a 3-vector that specifies the velocity of the frame origin with respect to the parent reference frame. This is the time derivative of the position vector as seen by an observer fixed in the parent frame. The components of this vector are resolved onto the coordinate axes of the parent frame.

5.4.3.3. Rotational State

The SpaceTimeCoordinate data type contains a required *rotational_state* attribute that is of type ReferenceFrameRotation. The ReferenceFrameRotation data type contains two attributes, one for attitude and one for angular velocity.

Attitude Quaternion - A ReferenceFrameRotation has a required *attitude_quaternion* attribute. This is an attitude quaternion of the reference frame ('subject frame') with respect to its parent reference frame ('referent frame').

Angular Velocity - A ReferenceFrameRotation has a required *angular_velocity* attribute. This is a 3-vector that specifies the angular velocity of the entity body frame with respect to the parent reference frame. The components of this vector are resolved onto the coordinate axes of the reference frame (‘subject frame’).

5.5. The Environment FOM Module (SISO_SpaceFOM_environment)

The SpaceFOM Environment module (SISO_SpaceFOM_environment) provides the fundamental data types used to represent the basic physical environmental properties associated with space-based simulations. For instance, any position of an entity in a SpaceFOM simulation is related to a particular reference frame. Many different reference frames can be used in a federation execution. The Environment FOM Module specifies how these are represented. The environment types in this FOM module are presented in Appendix Section C.4 along with tables describing the types and their relationship to one another.

5.6. Rules

Based on the preceding discussion, we can infer some general and specific rules associated with ReferenceFrame object instances and the federation execution's associated reference frame tree.

Rule 5-1: Unique ReferenceFrame Names.

Requirement: The *name* attribute and object instance name of a ReferenceFrame object class instance shall match exactly and be unique throughout a federation execution.

Rationale: These names are identifiers for the ReferenceFrame instances in the reference frame tree. The name attribute is used to distinguish one instance of a ReferenceFrame from any other ReferenceFrame and should therefore be unique across the federation execution. This uniqueness ensures the ability to establish an unambiguous transformation chain between any two ReferenceFrames in the tree.

Rule 5-2: Root ReferenceFrame Parent Name

Requirement: The *parent_name* attribute of the 'root' ReferenceFrame object class instance shall be a zero-length string (often referred to as an empty or null string).

Rationale: Every reference frame tree must have a single root reference frame. By definition, the root reference frame does not have a parent reference frame. The zero-length string is a way of marking this as the root reference frame.

Rule 5-3: Only One Root Reference Frame

Requirement: One and only one (1) root reference frame shall exist within a SpaceFOM-compliant federation execution.

Rationale: The reference frame tree is a special form of Directed Acyclic Graph (DAG) called a directed rooted labeled tree. The nature of this type of DAG dictates that it has one and only one root node. Therefore, the reference frame tree must have one and only one root node.

Rule 5-4: Fixed Root Reference Frame

Requirement: A federation execution shall use the same root reference frame for the duration the federation execution.

Rationale: Changing the root reference frame during a federation execution would require a potentially expensive reorganization of the reference frame tree and possibly invalidate previously computed relative state computations.

Rule 5-5: ReferenceFrame Parent Name Existence

Requirement: With the exception of the root reference frame, the *parent_name* attribute in a ReferenceFrame object class instance shall exactly match the name of some ReferenceFrame instance in the federation execution's reference frame tree.

Rationale: The reference frame tree is a special form of Directed Acyclic Graph (DAG) called a directed rooted labeled tree. This form of DAG requires a mechanism to associate elements within the graph. The *parent_name* attribute is the associative mechanism for the reference frame tree DAG. The *parent_name* attribute is a reference to a ReferenceFrame object instance in the reference frame tree. The only element in a DAG that does not need a reference to another element is the 'root' element. For the SpaceFOM reference frame tree, this is the root reference frame.

Rule 5-6: All Parent ReferenceFrames Must Exist

Requirement: All parent reference frames shall exist as owned and published object instances when the federation execution is running (e.g., advancing time).

Rationale: Federates in a SpaceFOM-compliant federation execution can use the reference frame tree to determine the spatial and temporal association of entities associated with reference frames. To do this, a federate will traverse the tree accumulating transformation information and applying it to compute a mathematical transformation between nodes in the tree. The traversal process requires "walking" up and down the tree. Walking up the tree requires the use of a reference frame's parent frame. If this does not exist, then the traversal process is broken.

Rule 5-7: Document All Federation Reference Frames in FESFA

Requirement: All reference frames used in a federation execution shall be documented in the FESFA.

Rationale: Reference frames provide the fundamental framework for understanding when and where objects are within a federation execution. In order to make effective use of this information, a federate needs to understand the content and topology of the federation execution's reference frame tree. This information must be provided in the FESFA.

Rule 5-8: Coherent ReferenceFrame Updates

Requirement: All attributes of a ReferenceFrame object class instance shall be provided in a single time coherent update.

Rationale: In order to ensure time homogeneity of the associated parameters, all the attributes of a ReferenceFrame object class instance must be provided as a single time coherent update.

5.7. Guidance

Reference frames should use the recommended standard reference frames in Appendix F.

Any non-standard reference frames should be named using the naming conventions defined in Section 5.3.

6. Entities and Interfaces

Having discussed the meaning of "*when*" in Chapter 4 (Time) and the meaning of "*where*" in Chapter 5 (Reference Frames), it now makes sense to discuss the meaning of "*what*".[15] For the SpaceFOM, this starts with two basic concepts: entities and interfaces.

6.1. Entities

The concept of an entity captures the highest-level abstraction of a "*what*" in the SpaceFOM. Any physical object that exists within space and time should be representable as an entity. In this version of the SpaceFOM, the concept of an entity is represented as a two-level hierarchy of PhysicalEntity and DynamicalEntity.[16] A physical entity is the base element and is represented with the basic attributes necessary to locate that entity in time and space. A dynamical entity has all the attributes of a physical entity with additional attributes to facilitate the characterization of the entity's physics-related dynamic behavior. These object classes are intended to provide base representations that can be extended to represent more detailed and specific entity types.

6.1.1. Physical Entities

A PhysicalEntity is the highest-level object class in the SpaceFOM entity hierarchy. This object class provides attributes to describe an entity's location in time and space. It also contains attributes to uniquely identify it individually from all other physical entities in the federation execution.

Physical entities have two intrinsically associated reference frames: a 'structural frame' and a 'body frame'. These are not registered in the Federation Execution's Reference Frame tree (see Section 5.2.2) but are used to place and orient the entity in space with respect to a reference frame in that tree. The origin of the structural frame is located at some arbitrary but known point on the entity.[17] The body frame origin is at the entity's center of mass. The body frame is located with respect to the entity's structural reference frame by a vector from the origin of the structural reference frame to the center of mass of the entity. This vector is expressed in the entity's structural reference frame. The orientation of the entity's body frame with respect to the entity's structural reference frame is defined by an attitude quaternion.[18]

The position and attitude of an entity is therefore defined by the position and attitude of the entity's body frame with respect to the entity's *parent_reference_frame*, which must be a reference frame instance in the Federation Execution's Reference Frame Tree. This, along with time, the *center_of_mass* vector and *body_wrt_structural* attitude quaternion, can be used to unambiguously locate the entity in time and space.

The descriptions of the attributes of the PhysicalEntity object class are separated into three principal categories: identifying informational attributes, entity state, and structural frame definition.

6.1.1.1. Identifying and Informational Attributes

The identifying and informational attributes of the PhysicalEntity object class are described in detail in this section. These include applicable rules associated with those attributes.

[15] We will leave it as an exercise for the reader to contemplate "*why*".
[16] Subsequent versions of the SpaceFOM will most likely expand and extend this admittedly limited hierarchy.
[17] While the location of the structural reference frame is not specified directly in the FOM, it should be described in the FCD document associated with the Federate publishing that PhysicalEntity.
[18] This relative orientation is usually fixed for a specific entity and never changes. In most cases, this orientation is some trivial combination of 90-degree rotations about principal axes but not always.

Name - The principal identifier for a PhysicalEntity is a required *name* attribute string that must be unique throughout the federation execution. This name should correspond to the object class instance name in the federation execution. This will ensure that it is unique throughout the federation execution. The name attribute is used to identify an instance of a PhysicalEntity from any other PhysicalEntity.

Type - A PhysicalEntity has an optional *type* attribute string that is used to identify the underlying nature of a particular instance of a PhysicalEntity. This is provided as a means of introspection for the object class instance. It can be used to provide a means to obtain a more specific description of the underlying object. This can be useful for federates that are set up to only discover instances of the PhysicalEntity object class but want to display information that is more relevant to the actual type of the underlying object. One example might be for a graphics federate that wants to represent the objects with relevant 3D models.

Status – A PhysicalEntity has an optional *status* attribute. This is provided as a means to convey a high-level status for the entity. By design, the *status* attribute is unstructured. A federation execution is free to define and restrict the format of the underlying string to meet the federation execution's needs. Since this attribute is both optional and unstructured, a federate receiving an update to this attribute should be prepared to handle unrecognized status values.

6.1.1.2. Entity State

Time, position and attitude define the state of a PhysicalEntity. These concepts are represented as a SpaceTimeCoordinate state of the entity's body frame with respect to a defined parent reference frame (see Section 5.4.3).

Parent Reference Frame – A PhysicalEntity has a required *parent_reference_frame* attribute. This is a non-empty string that identifies the reference frame with respect to which the kinematic state attributes of this entity are calculated.

State – A PhysicalEntity has a required *state* attribute. This is a SpaceTimeCoordinate representation of the time, position and attitude of the entity's body frame with respect to the *parent_reference_frame* described above (see Section 5.4.3).

In addition to the SpaceTimeCoordinate state attribute, the PhysicalEntity object class also contains translational acceleration and rotational acceleration attributes to enable more accurate state propagation between state updates.

Acceleration - A PhysicalEntity has an optional *acceleration* attribute. This is a 3-vector that specifies the acceleration of the entity body frame origin (i.e., the entity's center of mass) with respect to the parent reference frame. This is the time derivative of the velocity vector as seen by an observer fixed in the parent frame. The components of this vector are resolved onto the coordinate axes of the parent frame. When not present, the acceleration *attribute* should be assumed to be a zero vector.

Rotational Acceleration - A PhysicalEntity has an optional *rotational_acceleration* attribute. This is a 3-vector that specifies the angular acceleration of the entity body frame with respect to the parent reference frame. This is the time derivative of the angular velocity vector as seen by an observer fixed in the parent frame. The components of this vector are resolved onto the coordinate axes of the entity body frame. When not present, *the rotational_acceleration* attribute should be assumed to be a zero vector.

6.1.1.3. Structural Frame Definition

Center of Mass - A PhysicalEntity has a required *center_of_mass* attribute. This is a 3-vector that specifies the position of the entity's center of mass (the body frame origin) with respect to the origin of the entity's structural frame. The components of this vector are resolved onto the coordinate axes of the structural frame.

Body With Respect To Structural - A PhysicalEntity has an optional *body_wrt_structural* attribute. This is an attitude quaternion that specifies the orientation of an entity's body frame with respect to the entity's structural frame. This attitude quaternion should be set at initialization and never changed. When not present, the *body_wrt_structural* attribute should be assumed to be the identity quaternion (no rotation).

6.1.2. Dynamical Entities

A DynamicalEntity is the second level in the SpaceFOM entity object class hierarchy, inheriting from PhysicalEntity. In addition to all the attributes associated with the PhysicalEntity object class, this object class provides attributes to characterize an entity's physics based behavior subject to external non-conservative forces and torques as well as changes in mass properties.

Like physical entities, dynamical entities have two intrinsically associated reference frames: a 'structural frame' and a 'body frame' (see Section 6.1.1). An item of note for the DynamicalEntity object class is that the *center_of_mass* attribute is likely to change over time due to the mechanics associated with the generation of forces and torques. This raises the question of why the *center_of_mass* attribute is in the PhysicalEntity object class as opposed to the DynamicalEntity object class. The *center_of_mass* attribute is in the PhysicalEntity object class since it is necessary for the determination of the body and structural frames for the PhysicalEntity.

The descriptions of the attributes of the DynamicalEntity object class are separated into two principal categories: mass properties and external forces and torques.

6.1.2.1. Mass Property Attributes

The mass properties attributes of the DynamicalEntity object class are described in detail in this section.

Mass - A DynamicalEntity has a required *mass* attribute. This attribute is a scalar value that represents the mass of the DynamicalEntity.

Mass Rate - A DynamicalEntity has an optional *mass_rate* attribute. This attribute is a scalar value that represents the time rate of change of the DynamicalEntity's mass. When not present, this attribute is assumed to be zero.

Inertia - A DynamicalEntity has a required *inertia attribute*. This attribute is a 3x3 matrix that specifies the centroid moments and coefficients of inertia with respect to the coordinate axes of the DynamicalEntity's body frame.

Inertia Rate - A DynamicalEntity has an optional *inertia_rate* attribute. This attribute is a 3x3 matrix that specifies the time rate of change of the parameters in the InertiaMatrix. The elements in this matrix correspond directly to the elements in the InteriaMatrix. When not present, all elements of this matrix can be assumed zero.

6.1.2.2. Force and Torque Attributes

The force and torque attributes of the DynamicalEntity object class are described in detail in this section.

Force - A DynamicalEntity has an optional *force* attribute. This attribute is a 3-vector that specifies the total non-conservative external force on the entity. Force is expressed and applied in the entity's structural reference frame. Note that this does not include conservative forces on the entity like forces derived from accelerations due to gravitational fields. When not present, the *force* attribute should be assumed to be a zero vector.

Torque - A DynamicalEntity has an optional *torque* attribute. This attribute is a 3-vector that specifies the total non-conservative external torque on the entity. It is expressed in the entity's structural reference frame. Note that this does not include conservative torques on the entity that may arise from rotation and asymmetries in the mass distribution of the entity. When not present, the *torque* attribute should be assumed to be a zero vector.

6.2. Physical Interfaces

It would be convenient if the PhysicalEntity object class captured all the necessary concepts of a "*what*" in the SpaceFOM, but this is not the case. The concept of a physical entity implies independence in position and attitude over time. However, there are some elements in the description of an entity that are dependent on and defined with respect to the entity. Specifically, there may be locations on an entity that define interface locations on the entity that may be important to it and/or other entities. For instance, if two vehicles in space are going to dock, each needs to be aware of the position and orientation of both their docking port and the other vehicle's docking port. The PhysicalInterface object class defines this entity dependent geometrical definition of an entity interface.

Any geometrically definable interface associated with an entity should be representable as a PhysicalInterface object class instance. In this version of the SpaceFOM, a PhysicalInterface is the base element and is represented with the basic attributes necessary to locate and orient that interface with respect to its parent entity or another physical interface on the same entity. This object class is intended to provide a base representation that can be extended to represent more detailed and specific entity interfaces.

6.2.1. Physical Interface Object Class Attributes

Name - The principal identifier for a PhysicalInterface is a required *name* attribute string that must be unique throughout the federation execution. This name should correspond to the object class instance name in the federation execution. This will ensure that it is unique throughout the federation execution. The name attribute is used to identify an instance of a PhysicalInterface from any other PhysicalInterface. See section 6.5.2 for further guidance on constructing PhysicalInterface names.

Parent Name – A PhysicalInterface has a required *parent_name* attribute. This is a non-empty string that identifies the parent PhysicalEntity or PhysicalInterface with respect to which the kinematic state attributes of this interface are defined (see *position* and *attitude* attributes below).

Position – A PhysicalInterface has a required *position* attribute. This is a 3-vector that specifies the position of the interface reference frame origin with respect to the parent structural reference frame. The components of this vector are resolved onto the coordinate axes of the parent frame.

Attitude – A PhysicalInterface has a required *attitude* attribute. This is an attitude quaternion of the interfaces reference frame ('subject frame') with respect to its parent structural reference frame ('referent frame').

6.3. The Entity FOM Module (SISO_SpaceFOM_entity)

The SpaceFOM Entity module (SISO_SpaceFOM_entity) provides the fundamental data types used to represent basic physical object properties associated with space-based simulations. The Entity FOM Module specifies how these are represented. The entity types in this FOM module are presented in Appendix Section C.5 along with tables describing the types and their relationship to one another.

6.4. Rules

Based on the preceding discussion, we can infer some general and specific rules associated with PhysicalEntity, DynamicalEntity, and PhysicalInterface object instances.

6.4.1. PhysicalEntity Rules

Rule 6-1: Unique PhysicalEntity Names

Requirement: The name attribute and object instance name of a PhysicalEntity object class instance shall match exactly and be unique throughout a federation execution.

Rationale: The principal identifier for a PhysicalEntity is a required *name* attribute string. The name attribute is used to identify an instance of a PhysicalEntity from any other PhysicalEntity.

Rule 6-2: PhysicalEntity Types Described in FESFA

Requirement: All expected values for a *type* attribute in a PhysicalEntity must be defined and described in the FESFA.

Rationale: The PhysicalEntity *type* attribute is used as an informal mechanism to provide a clue to the underlying type of the PhysicalEntity instance when the actual type is derived from PhysicalEntity. This is commonly referred to as type introspection. This can be useful for a federate that wants to discover all instances that derive from PhysicalEntity but still identify the actual derived instantiated type. However, for this to be effective, any federate wanting to use the form of introspection will have to know in advance the possible derived types available. This requires that the possible values of the *type* attribute be listed in the FESFA. See the guidance section below.

Rule 6-3: PhysicalEntity Type Set at Initialization

Requirement: The value for the *type* attribute in a PhysicalEntity object instance shall be set at initialization and not changed during a SpaceFOM-compliant federation execution.

Rationale: Since the underlying derived data type of a PhysicalEntity object instance cannot change during a federation execution, the *type* attribute should not change.

Rule 6-4: PhysicalEntity Status Defined and Described in FESFA

Requirement: All expected values for a *status* attribute in a PhysicalEntity shall be defined and described in the FESFA.

Rationale: By definition, PhysicalEntity *status* attributes are unstructured attributes. Therefore, any semantic value from the status attribute needs to be defined in the FESFA.

Rule 6-5: Handling Unrecognized PhysicalEntity Status Values

Requirement: A federate receiving a PhysicalEntity *status* attribute update shall be prepared to handle unrecognized values.

Rationale: By definition, PhysicalEntity *status* attributes are unstructured attributes. Therefore, any SpaceFOM-compliant federate needs to be prepared to handle unrecognized status value attribute updates.

Rule 6-6: Existence of PhysicalEntity Parent Reference Frame

Requirement: The *parent_reference_frame* attribute in a PhysicalEntity object class instance shall exactly match the name of some ReferenceFrame instance in the federation execution's reference frame tree.

Rationale: The state information associated with a PhysicalEntity is expressed with respect to a defined and known reference frame. Therefore, that known frame must exist. While the absence of a known parent reference frame might not directly affect the propagation of the state of the underlying PhysicalEntity object instance, it will make it impossible to relate the state of that PhysicalEntity object instance to any other PhysicalEntity object instance in the federation execution.

Rule 6-7: Default PhysicalEntity Body Frame Specification

Requirement: When not specified, the PhysicalEntity *body_wrt_structural* attributes shall be defined to be an identity attitude quaternion.

Rationale: Typically, the state of a PhysicalEntity will be propagated and given in the entity's body frame. Note that, while the body frame can move with respect to the entity's structural reference frame, it may also have a different native orientation. The specification of the entity's body frame orientation with respect to the entity's structural reference frame is required for the unambiguous computation of locations expressed in the entity's structural reference frame. When an entity's *body_wrt_structural* attribute is not specified, the definition of it as an identity attitude quaternion resolves any ambiguity.

Rule 6-8: Coherent PhysicalEntity Updates

Requirement: All attributes of a PhysicalEntity object class instance shall be provided in a single time coherent update.

Rationale: In order to ensure time homogeneity of the associated parameters, all the attributes of a PhysicalEntity object class instance must be provided as a single time coherent update.

6.4.2. DynamicalEntity Rules

Note that since DynamicalEntity is a derived data type from PhysicalEntity, all the rules that apply to a PhysicalEntity also apply to a dynamical entity. However, the following additional rules apply to a DynamicalEntity:

Rule 6-9: Specifying DynamicalEntity Force and Acceleration Together

Requirement: The associated acceleration attribute shall also be present when the *force* attribute is present in a DynamicalEntity attribute update.

Rationale: In a Newtonian dynamical system, externally applied forces will generate associated translational accelerations and residual torques. The residual torques will generate corresponding rotational accelerations. Therefore, force and acceleration must be specified together.

Rule 6-10: Specifying DynamicalEntity Torque and Rotational Acceleration Together

Requirement: The associated *rotational_acceleration* attribute shall also be present when the *torque* attribute is present in a DynamicalEntity attribute update.

Rationale: In a Newtonian dynamical system, externally applied torques will generate associated rotational accelerations. Therefore, torque and rotational acceleration must be specified together.

Rule 6-11: Coherent DynamicalEntity Updates

Requirement: All attributes of a DynamicalEntity object class instance shall be provided in a single time coherent update.

Rationale: In order to ensure time homogeneity of the associated parameters, all the attributes of a DynamicalEntity object class instance must be provided as a single time coherent update.

6.4.3. PhysicalInterface Rules

Rule 6-12: Unique PhysicalInterface Names

Requirement: The name value in a PhysicalInterface object class instance must be unique throughout a federation execution.

Rationale: The principal identifier for a PhysicalInterface is a required *name* attribute string. The name attribute is used to identify an instance of a PhysicalInterface from any other PhysicalInterface.

Rule 6-13: PhysicalInterface Naming Convention Specified in the FESFA

Requirement: The naming convention used to ensure unique names for PhysicalInterface object class instances throughout a federation execution shall be defined in the FESFA.

Rationale: By requirement, the PhysicalInterface *name* attribute must be unique and match the object instance name declared in the federation execution. However, generating a unique name that is readily identifiable may prove challenging in some cases. Establishing a defined naming convention will help to disambiguate PhysicalInterface object instance names (see guidance Section 6.5.2 below). This naming convention must be specified in the FESFA.

Rule 6-14: Existence of PhysicalInterface Parent Object Instance

Requirement: The PhysicalEntity or PhysicalInterface specified by the *parent_name* attribute in an instance of a PhysicalInterface object class shall exist within the federation execution.

Rationale: A PhysicalInterface will be defined with respect to either a PhysicalEntity or another PhysicalInterface. The, possibly nested, relationship between a PhysicalInterface object instance and its parent PhysicalEntity or PhysicalInterface object instance is defined through the *parent_name* attribute. This provides a direct mechanism to establish a transformation chain between a PhysicalInterface and any other PhysicalEntity or PhysicalInterface object instance in the federation execution. However, the parent object instance associated with the specified name must exist.

Rule 6-15: Coherent PhysicalInterface Updates

Requirement: All attributes of a PhysicalInterface object class instance shall be provided in a single time coherent update.

Rationale: In order to ensure time homogeneity of the associated parameters, all the attributes of a PhysicalInterface object class instance must be provided as a single time coherent update.

6.5. Guidance

This section provides some additional guidance associated with PhysicalEntity and PhysicalInterface objects.

6.5.1. PhysicalEntity Guidance

In general, the form of type string-based introspection employed through the use of the *type* attribute in the PhysicalEntity data types is discouraged. However, no formal introspection mechanism exists with HLA object instances. Note that in order for a federate to make use of any introspection mechanism, it has to have some understanding of the possible derived types. This string-based approach does lend itself to some level of runtime configuration. This leads to Rule 6-2, and the documentation of all types in the FESFA. In general, we recommend that the type attribute string used should exactly match the data type name of the underlying derived type. For instance, the type attribute for a DynamicalEntity should be a string containing “DynamicalEntity”.

6.5.2. PhysicalInterface Guidance

One way to construct the unique name for the physical interface is to prepend the unique name of the entity to which it is associated (see *parent_name* attribute in 6.1.1.1 and 6.1.2) to a generic name for the interface. For instance, two spacecraft have docking ports. These spacecraft may be two distinct instances of the same type of spacecraft (e.g., two Soyuz: “soyuz_1” and “soyuz_2”). In this case, the general name for the docking port, say “docking_port”, would collide in the federation execution. However, by prepending the spacecraft object instance name to the general name we get two distinct and descriptive instance names: “soyuz_1.docking_port” and “soyuz_2.docking_port”.

7. Execution Control

Every simulation has an executive that controls the execution of the simulation as it starts up, goes through a defined initialization sequence, transitions into various running states, and ultimately goes through a defined shutdown sequence. At some level, a distributed simulation (e.g., a federation execution) must go through analogous processes. Interoperability between federates in a federation execution requires not only the specification of the data exchange between federates but also the specification of executive behavior. With this in mind, the SpaceFOM defines some specifics of the execution control required for a SpaceFOM-compliant federate.

7.1. Overview

At the highest level, the SpaceFOM executive flow has four principal states: start, initialization, execution, and shutdown (see Figure 7-1). This is, of course, a simplified view of the SpaceFOM executive architecture. However, this provides a suitable starting point for subsequent more detailed discussions.

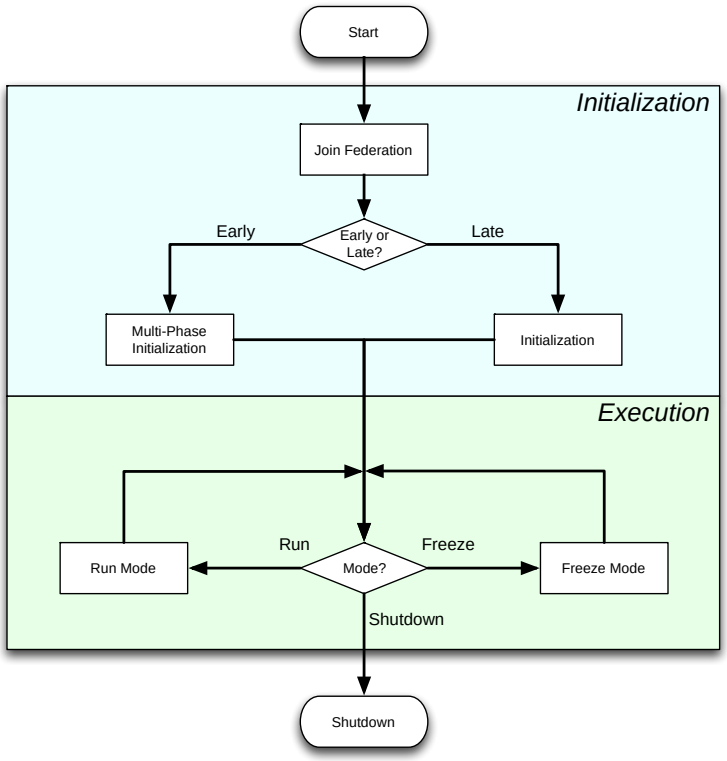

Figure 7-1: Simplified SpaceFOM Executive Flow

While the Start and Shutdown states are most likely trivial entry and exit points. The Initialization and Execution states are considerably more complex. The SpaceFOM designates the role of the Master Federate as the principal federate for controlling and coordinating the federation execution. The Master Federate makes use of three principal HLA mechanisms to manage execution control: Execution Configuration Objects, Mode Transition Request Interactions, and coordination synchronization points.

7.1.1. Time Management

Having already covered concepts of time and principal simulation time lines in section 4, it should be evident that traditional concepts of simulation time management become more nuanced with HLA based distributed simulations. Two principal time management concepts are important: 1) management of SST using HLA time management mechanisms and 2) management of the federation execution with respect to the real world passage of PT.

7.1.1.1. Simulation Scenario Time Management

The SpaceFOM relies on the HLA time management infrastructure for coordinating the progression of SST across participating federates in a given federation execution.

This ensures that all coordinated federates move forward with a coordinated SST. This helps to maintain consistent state information between participating time-managed federates. However, it does not regulate how fast the federation execution progresses with respect to PT. Simulations that are not paced with respect to PT will run as fast as possible (AFAP) base on the slowest executing time managed federate and the communication latencies between federates.

7.1.1.2. Physical Time Management

The SpaceFOM supports two concepts for controlling the progression of SST with respect to PT: a Pacing Federate and CTE. These concepts are not mutually exclusive.

7.1.1.2.1. Pacing Federate

A Pacing Federate is a federate in the federation execution that employs some kind of timed wait loop to regulate the progression of SST with respect to PT. This approach is often employed through the use of the Pacing Federate's computer clock to "pace" the progression of SST with the computer's concept of the progression of PT through the use of CCT. The Pacing Federate employs this use of CCT and the SST management mechanisms described above to "pace" the federation execution.

This method is very effective in managing the real-time progression of SST across a federation execution. However, there are limits to how well this approach can control the variation of SST progression with respect to PT across a federation execution. Specifically, the federation execution will progress in general coordination with PT but individual federates may have variations in the actual length and timing of each execution cycle. These variations are generally a result of the variability of communications latencies between federates and time management coordination implementations within the HLA RTI.

7.1.1.2.2. Central Timing Equipment

In situations where more accurate control is required for real-time performance of individual federates, the SpaceFOM supports the use of CTE. There are a number of available CTE technologies (standards). However, in general, CTE provides a highly accurate coordinated timing mechanism for each computer connected through the CTE. This provides each CTE-equipped computer with a commonly available and highly accurate definition of CCT. The federate's computer uses this CTE-defined CCT to implement a time based wait loop to control the progression of SST with respect to PT.

7.1.2. Master Federate

The Master Federate is the principal control federate in the federation execution. The Master Federate is responsible for coordinating and controlling the execution state of the federation through the use of a single instance of a published Execution Configuration Object named "ExCO" and a collection of mode transition synchronization points.

The Master Federate has specific responsibilities in managing a SpaceFOM-compliant federation execution. These responsibilities are called out in the subsequent sections. Typically, these are expressed as separate process flow lines in the descriptive diagrams.

7.1.3. Execution Configuration Object (ExCO)

An ExCO is a standard HLA object class which defines the base set of parameters necessary to coordinate federation and federate execution time lines and execution mode transitions in a SpaceFOM-compliant federation execution. The attributes associated with the ExCO are:

- **root_reference_frame:** Specifies the name of the root reference frame in the federation execution's reference frame tree. This frame shall remain fixed throughout the federation execution.

- **scenario_time_epoch:** This is the beginning epoch of the federation execution expressed in Terrestrial Time (TT), using the Truncated Julian Date (TJD) origin (1968-05-24 00:00:00 UTC) as the TT epoch.[19] This simulation scenario time (SST) epoch corresponds to HLA logical time (HLT) 0. All joining federates shall use this time to coordinate the offset between their SST, their simulation elapsed times (SET), and the HLT.
- **current_execution_mode:** This is the current running state of the federation execution in terms of a finite set of states expressed as a ExecutionMode enumeration value.
- **next_execution_mode:** This is the next running state of the federation execution in terms of a finite set of states expressed in the ExecutionMode enumeration. This is used in conjunction with the next_mode_cte_time (if used), next_mode_scenario_time, and associated synchronization point mechanisms to coordinate federation execution mode transitions.
- **next_mode_scenario_time:** This is the time for the next federation execution mode change expressed as a SST reference. Note: This value is only meaningful for going into freeze; exiting freeze is coordinated through a synchronization point mechanism.
- **next_mode_cte_time:** This is the time for the next federation execution mode change expressed as a Central Timing Equipment (CTE) time reference. The standard for this reference shall be defined in the FESFA when CTE is used.
- **least_common_time_step:** This is the time that represents the number of microseconds for the least common value of all the time step values in the federation execution (LCTS). This value is set by the Master Federate and does not change during the federation execution. This is used in the computation to find the next HLA Logical Time Boundary (HLTB) available to all federates in the federation execution.

7.1.4. Mode Transition Request (MTR) Interaction

The ModeTransitionRequest (MTR) interaction is used by participating federates, that are not the Master Federate, to request a federation execution mode transition. An MTR can be sent at any time during initialization or execution but only certain MTR requests are valid at certain times (see Section 7.4). The MTR contains one parameter:

- **execution_mode:** The execution mode requested. There are only 3 valid MTR execution mode values: EXEC_MODE_RUNNING, EXEC_MODE_FREEZE, EXEC_MODE_SHUTDOWN. Any other mode requests are automatically invalid. Of these three valid mode requests, only 7 combinations of current mode and requested mode are valid (see Section 7.4.2 and Table 7-1).

7.1.5. Coordinating Synchronization Points

The Master Federate uses a defined set of synchronization points to specify federation wide coordination points in the executive initialization and execution process flow. The SpaceFOM specifies the following 6 execution control synchronization points; 3 are used during initialization and 3 are used during execution mode transitions:

[19] Consequently, the federation and federates operate in increments of time as observed on the Earth's surface. This ignores time dilation that may exist between real-world analogs of the federates due to relativity. Section 4.1 explains why ignoring time dilation is acceptable for most space exploration simulations using current space transportation technologies.

- **initialization_started:** Used to indicate that the initialization phase of a SpaceFOM-compliant federation execution has been started. This synchronization point (sync-point) is not created until all federates required by the Master Federate have joined the federation execution. Once this occurs, the Master Federate announces this sync-point for all federates that have already joined the federation execution. All federates in the sync-point group must achieve this sync-point prior to proceeding with federate and federation execution initialization.
- **initialization_completed:** This synchronization point (sync-point) is registered by the federation execution Master Federate after all the early joining federates have achieved the "initialization_started" sync-point. This signals to any late joining federates that they can now proceed with their initialization and then on to the current execution mode of the federation execution. This sync-point will never be achieved by the Master Federate.
- **objects_discovered:** This synchronization point (sync-point) is used to mark the point at which all required objects have been discovered by all the federates taking part in the initialization process. This sync-point is used to ensure that all the necessary objects have been discovered prior to proceeding with the root reference frame discovery process and then multi-phase initialization.
- **root_frame_discovered:** This synchronization point (sync-point) is used to mark the point at which the root reference frame for this federation execution has been discovered by the Master Federate and all other federates participating in the initialization process. This is necessary prior to moving into the multi-phase initialization process.
- **mtr_run:** This is used to synchronize the mode transition to EXEC_MODE_RUNNING. This synchronization point (sync-point) is registered by the federation execution Master Federate upon receipt of a valid MTR interaction after sending out the associated ExCO update. Upon receiving the ExCO for the mode transition and at the associated transition time, all federates must achieve this sync-point prior to going into mode EXEC_MODE_RUNNING (see Section 7.4.4).
- **mtr_freeze:** This is used to synchronize the mode transition to EXEC_MODE_FREEZE. This synchronization point (sync-point) is registered by the federation execution Master Federate upon receipt of a valid MTR interaction after sending out the associated ExCO update. Upon receiving the ExCO for the mode transition and at the associated transition time, all federates must achieve this sync-point prior to going into mode EXEC_MODE_FREEZE (see section 7.4.5).
- **mtr_shutdown:** This synchronization point (sync-point) is used as a marker for the mode transition to EXEC_MODE_SHUTDOWN. This sync-point is registered by the federation execution's Master Federate to "mark" the federation execution as shutting down. This marker sync-point is used in addition to the ExCO. This sync-point is never achieved and will remain for the life of the federation execution to inform any late joining federates of shutdown and that the federates should proceed directly to their shutdown processes.

7.2. Initialization

The complexity of an initialization process for a simulation can range from something as simple as setting initial simulation parameter values to the complex cyclic evaluation of initial conditions based on an iterative determination of state dependencies. The SpaceFOM initialization framework has to have the flexibility to support initialization methodologies from the simple to the complex. An overview diagram of the SpaceFOM initialization specification is shown in Figure 7-2. Much more detailed diagrams are presented and discussed in the following section.

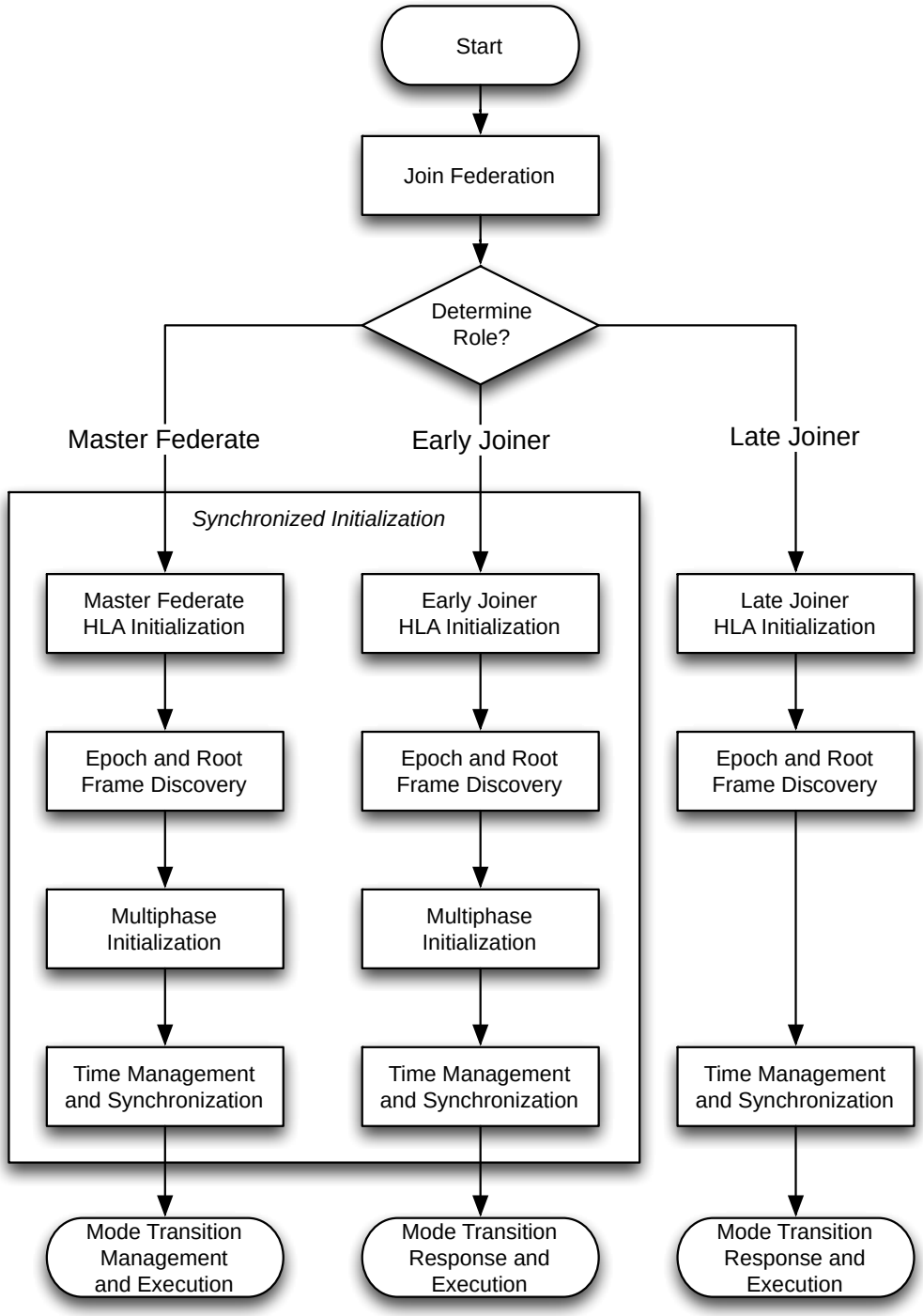

Figure 7-2: SpaceFOM Initialization Overview

The SpaceFOM initialization process begins with the Start entry point. The next step for any federate is to join the federation execution. The SpaceFOM specification for joining a federation execution is a multistep process with an iterative component to avoid a potential race condition. A flow chart of the SpaceFOM federation join process can be seen in Figure 7-3.

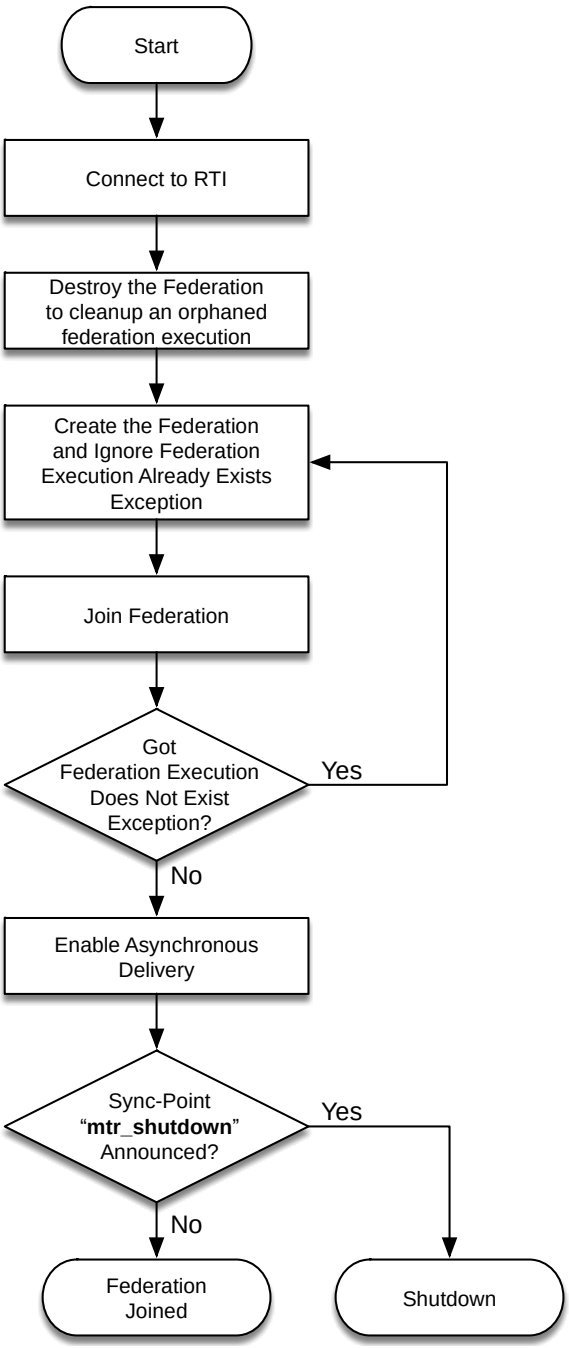

Figure 7-3: Join Federation Process

This join federation process starts with connecting to the HLA RTI. Strange as it may seem, the next step is to attempt to destroy the federation execution. This is done to clean up any orphaned federation executions that might exist. The next step is to attempt to create the federation execution, ignoring a federation already exists error. The create will only succeed for the first federate. Next, join the federation execution. Note that there is a potential race condition between the creation of a federation and joining. If another federate comes in and destroys the federation execution before the current federate joins, a "Federation Execution Does Not Exist" exception will occur. If this occurs, the federate can loop back to the create step and try again. Once the federate has successfully joined the federation, the federate shall enable asynchronous delivery (see Rule 4-28). The final step is to check for an announced "mtr_shutdown" synchronization point. This prevents a potential race condition between federation execution shutdown and late joining federates.

The "Join Federation" step is followed by the determination of the role of the participating federate. This step determines if a federate is a "Master Federate", "Early Joiner" federate, or "Late Joiner" federate. A flow chart of the role determination process can be seen in Figure 7-4.

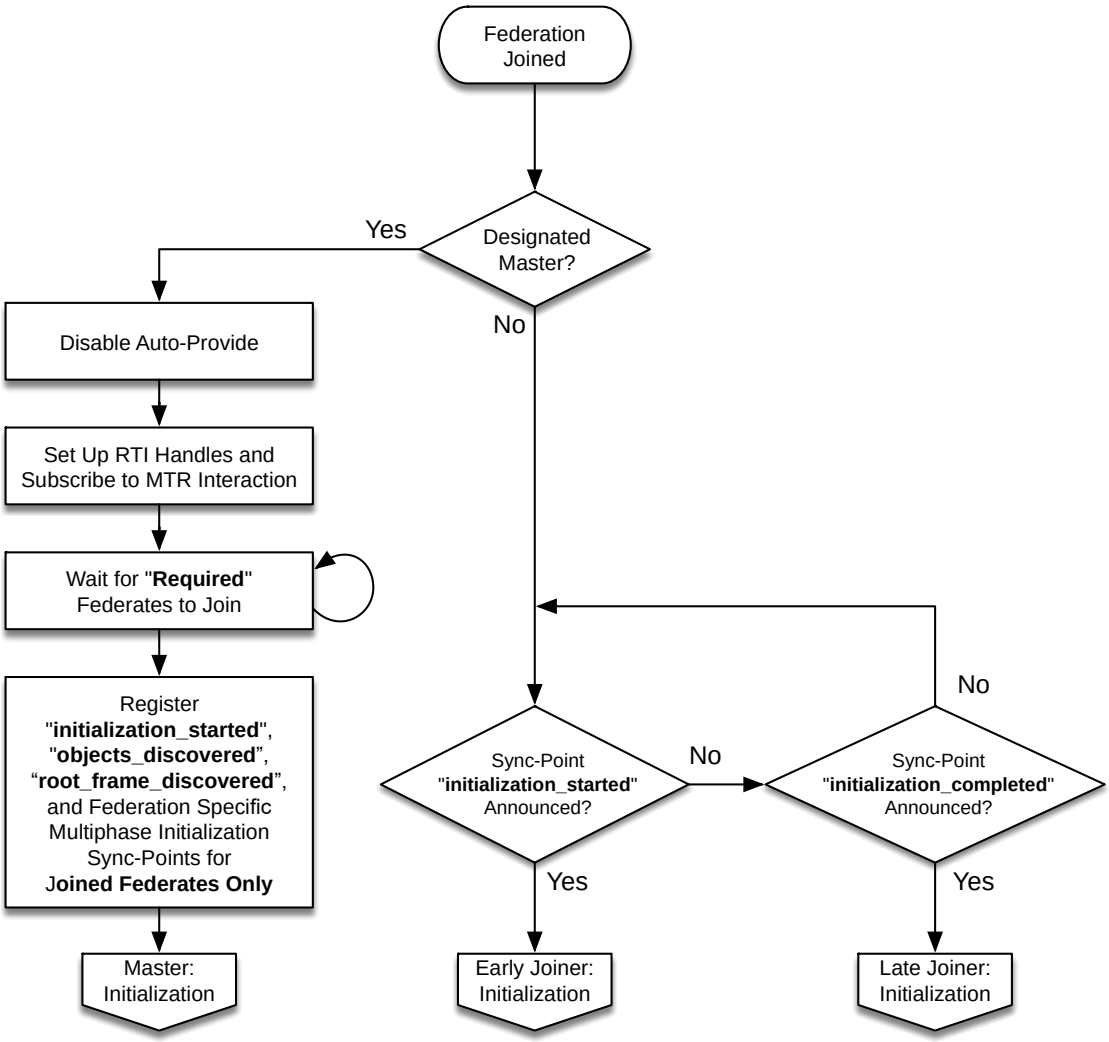

Figure 7-4: Role Determination Process

The Master Federate will be predetermined and documented in the FESFA. The role of Early Joiner versus Late Joiner will be determined by the timing of the federate's joining the federation execution and the federate's designation of either being, or not being, a required federate. The list of required federates shall be documented in the FESFA. Any federate that plays a key regulatory role in the federation execution should be a required federate. By definition, the Master Federate, Pacing Federate and Root Reference Frame Publisher Federate shall be required federates.

If the federate is a required federate, then it must always be an Early Joiner. If the federate is not a required federate but joins into the federation execution before the Master Federate discovers the last required federate, the federate will also be an Early Joiner. All other federates will be Late Joiners.

Once a federate has determined its role, each federate will proceed to its designated initialization process: Master, Early Joiner, or Late Joiner.

7.2.1. Master

The Master Federate has special responsibilities in the coordination and control of a SpaceFOM-compliant federate execution initialization process. After creating and/or joining the federation execution, the Master Federate first checks the state of the Auto-Provide attribute in the RTI, saves off the value, and then disables Auto-Provide.[20] The Master Federate then sets up the RTI handles and subscribes to the MTR interaction. The Master Federate then waits until all other required federates have joined the federation execution.[21] All required federates and any other federates that join the federation execution prior to the last required federate will be designated as Early Joiner federates (see Figure 7-4). The Master Federate then begins to register and achieve specific synchronization points (see Section 7.1.5) in a specific order to coordinate the initialization process.

Immediately after the last required federate has joined, the Master Federate registers the "initialization_started" synchronization point with only the currently joined federates as members of the synchronization set; these are Early Joiners.[22] The "initialization_started" synchronization point is used to mark the start of the initialization process and the point from which all subsequent joined federates are considered "late". This eliminates a potential race condition for joining federates. Any other federates, not in the "initialization_started" synchronization set, are designated to be Late Joiners.

The Master Federate also registers "objects_discovered" and "root_frame_discovered" synchronization points with only the Early Joiner federates as members of the synchronization set. The "objects_discovered" synchronization point is used to mark the point at which all required objects have been discovered by all the federates taking part in the initialization process. The "objects_discovered" synchronization point is used in the HLA initialization process described in Section 7.2.1.1. The "root_frame_discovered" synchronization point is used in the discovery of the Root Reference Frame in Section 7.2.1.2. To support the multiphase initialization process described in Section 7.2.1.3, the Master Federate will also register any federation execution specific multiphase synchronization points with only the Early Joiner federates. These synchronization points shall be documented in the FESFA along with the phase ordering and associated participating federates.

7.2.1.1. HLA Initialization

The next step in the SpaceFOM initialization process is to setup and initialize the HLA infrastructure needed by the Master Federate to both operate as a federate and fulfill its responsibility as the coordinating authority for the federation execution (see Figure 7-5).

---

[20] Auto-Provide must be disabled during initialization to preserve the attribute update dependencies in multiphase initialization.
[21] The Master Federate discovers required federates by name.
[22] If the Master Federate fails to register the "initialization_started" synchronization point, then there must already be a Master Federate in the federation execution. The aspiring Master Federate can either then resign, if that is their only duty, or it can continue on in other roles.

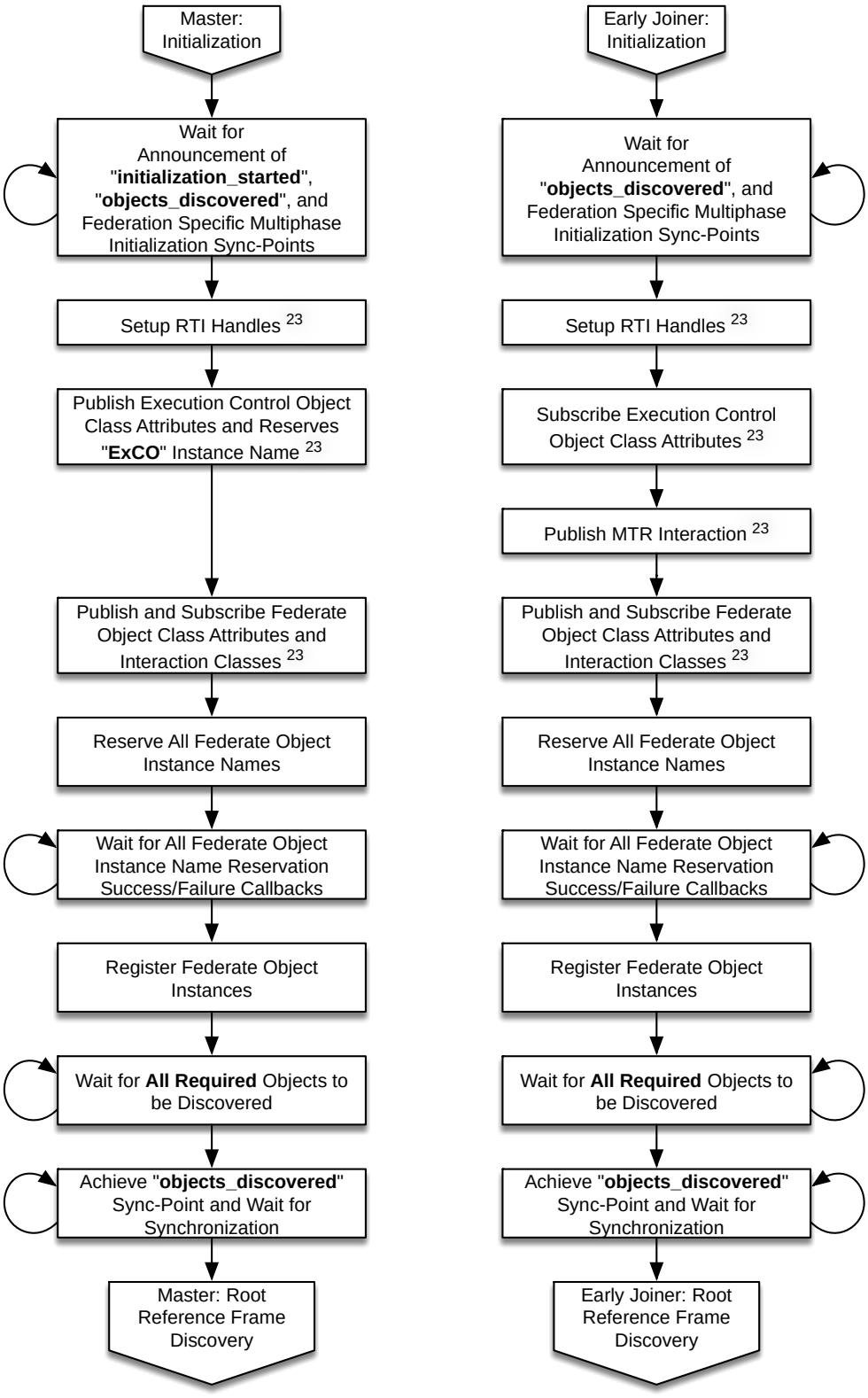

Figure 7-5: Master and Early Joiner HLA Initialization

The Master Federate begins by waiting for the announcement of the “initialization_started”, “objects_discovered”, and multiphase initialization synchronization points.

The federate then sets up any needed RTI handles. The next step is publishing the Execution Configuration Object.[23] As described above (see Section 7.1.3), the Execution Configuration Object plays an important role in the coordination and control of the federation execution. Only a single Execution Configuration Object instance shall ever exist in a given federation execution. The Master Federate will reserve the “ExCO” name for this singleton object instance (see 7.1.3).

The next step is to publish and subscribe any federate specific object classes.[23] This will be followed by the reservation of all federate specific object instance names, waiting for the instance name reservation success or failure callbacks. At this point, the federate will register the federation specific object instances. The federate now waits for all required object instances to be discovered by object instance name. Finally, the federate achieves the “objects_discovered” synchronization point and waits for synchronization. This ensures that all objects required by those federates participating in early initialization have been discovered and that those federates will receive any updates for those object instances.

In order to protect against an ExCO mode transition race condition between the Master Federate publish of the ExCO and the early joiner federates subscription to the ExCO, the Master Federate shall not send an ExCO update prior to achieving the "objects_discovered" synchronization point.

In order to prevent a potential deadlock condition and to support shutdown at any time after the federation is synchronized on the "objects_discovered" synchronization point, all federates shall check for ExCO mode transitions in any wait loops. This includes waiting for Time Advance Grant.

In order to avoid potentially complex mode recovery schemes during initialization, no mode transitions other than EXEC_MODE_SHUTDOWN are allowed prior to completion of the early joiner initialization process. Specifically, the Master Federate shall not send an ExCO mode transition other than EXEC_MODE_SHUTDOWN prior to the registration of “initialization_completed” synchronization point.

7.2.1.2. Epoch and Root Reference Frame Discovery

The Master Federate is now ready to establish the federation execution simulation scenario time epoch and discover the federation execution’s root reference frame. A flow chart of this process can be seen in Figure 7-6.

[23] Note that some HLA initialization steps in these diagrams can be performed earlier in the sequence but should be performed no later than their location in the diagrammed sequence.

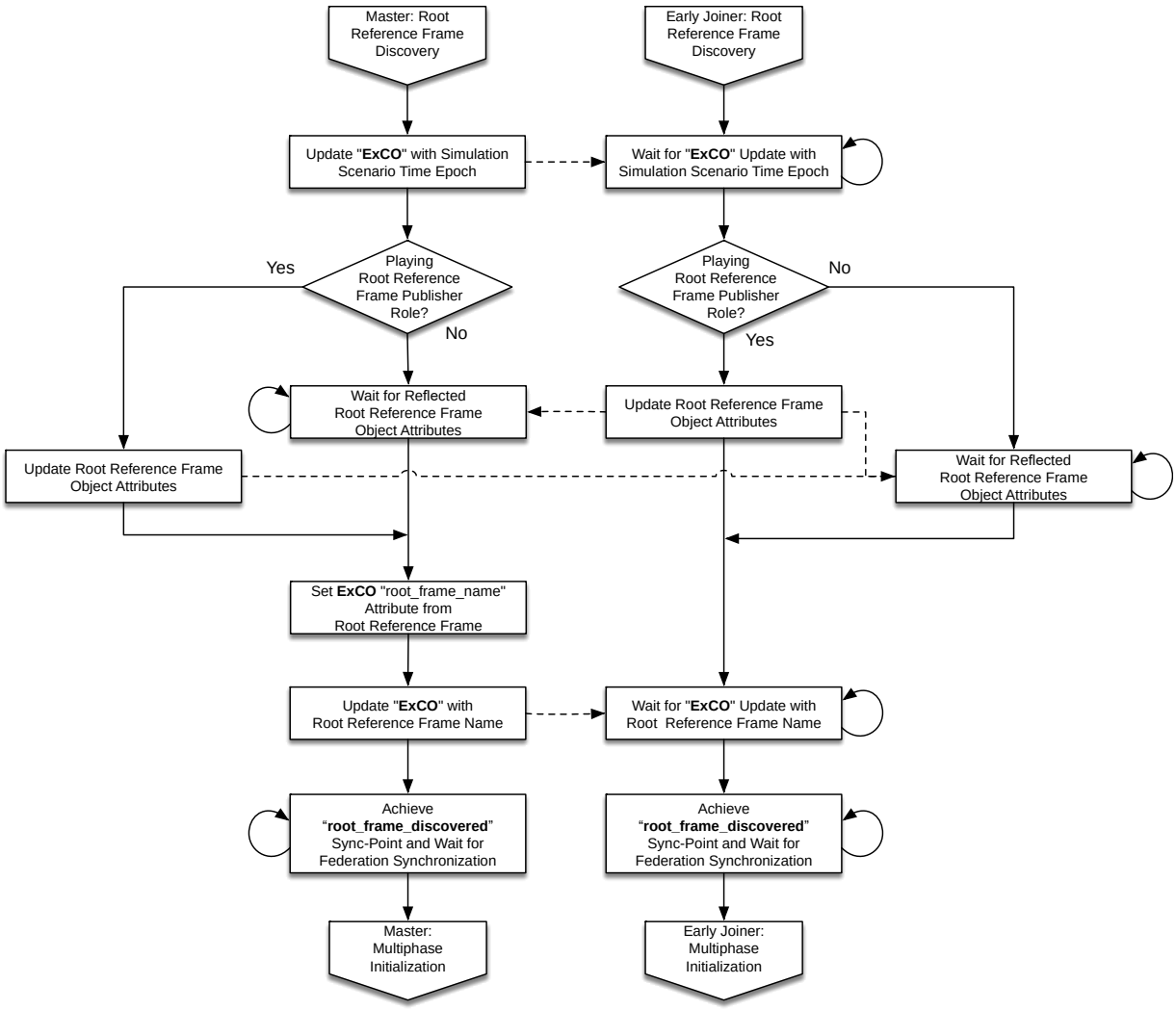

Figure 7-6: Epoch and Root Reference Frame Discovery

The Master Federate starts by updating the ExCO with the current simulation scenario time as the scenario time epoch. The Early Joiner federates will wait on this ExCO update with the simulation scenario time epoch. This provides the simulation scenario time for use in the computation of the root reference frame and the multiphase initialization process.

The Master Federate must next determine the name of the root reference frame in the federation execution's reference frame tree. The name of the root reference frame is published in the ExCO. If the Master Federate is publishing the root ReferenceFrame object, then the name is known. If the Master Federate does not publish the root ReferenceFrame object, then the Master Federate must discover the root ReferenceFrame object from the discovered ReferenceFrame objects. The Master Federate can determine the root reference frame by finding the first ReferenceFrame object instance with a null parent frame name (zero-length string). Once the Master Federate had determined the root reference frame and the associated name, the root reference frame name can be set for the next update of the ExCO.

After updating the ExCO with the discovered root reference frame name, the Master Federate will achieve and wait for the "root_frame_discovered" synchronization point. Once the federation is synchronized on the "root_frame_discovered" synchronization point, the Master Federate will be positioned to begin the multiphase initialization process.

7.2.1.3. Multiphase Initialization

The multiphase initialization part of the initialization process is a loop on a predetermined number of data exchanges between the Master Federate and any Early Joiner federates. Each loop will be guarded with a named multiphase initialization synchronization point. The multiphase initialization process shall be documented in the FESFA. Each loop (phase) in the multiphase initialization sequence has the potential for sending phase dependent initialization data, and receiving phase dependent initialization data (see Figure 7-7). Each loop (phase) is synchronized by waiting on the federation execution to achieve that phase's synchronization point.

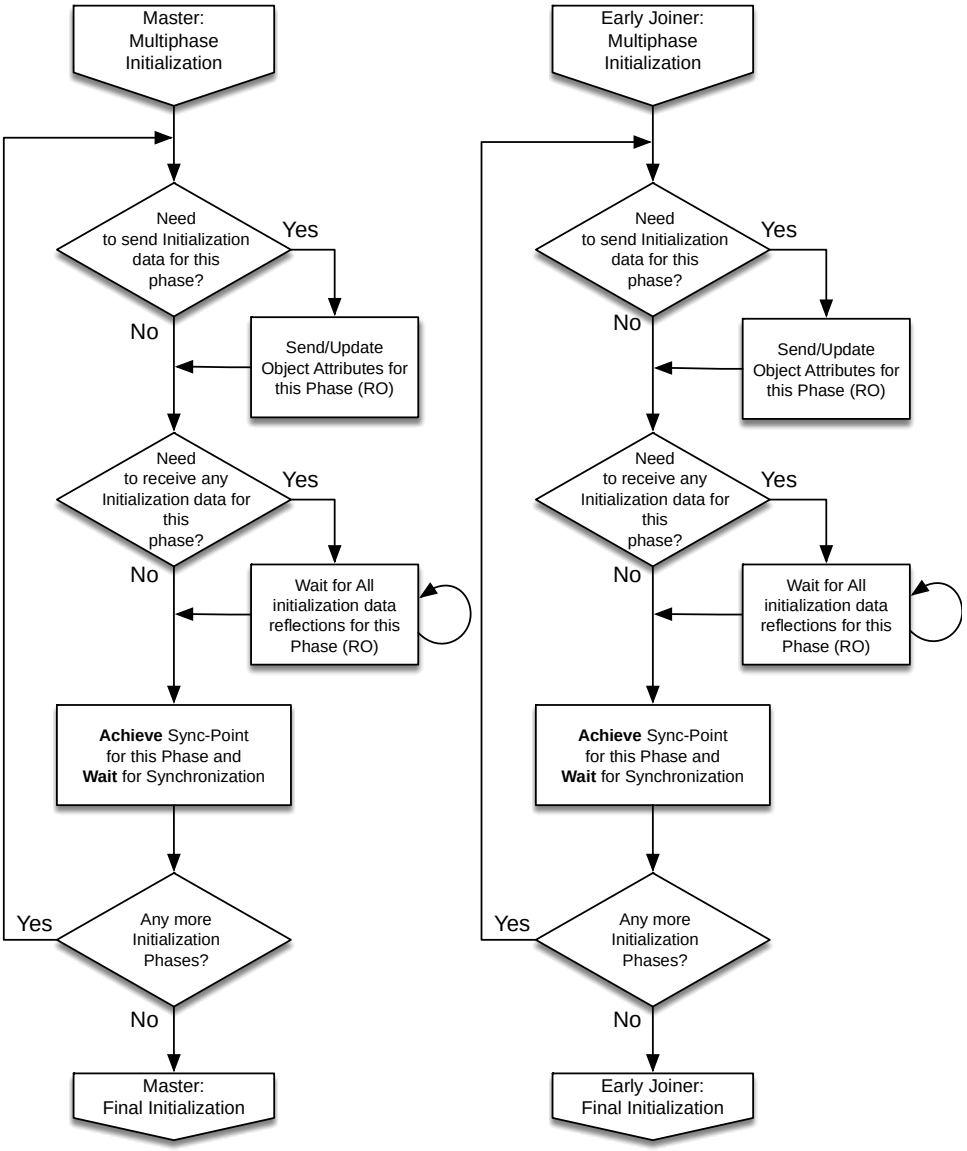

Figure 7-7: Multiphase Initialization Process

Each loop (phase) in this process provides for an exchange of data between federates participating in the federation execution. Not all federates will participate in every phase. In general, a federate will determine if it needs data provided in a given phase or if it needs to provide data in a given phase. This iterative process can be as simple or as complex a necessary. Due to the flexibility of this process, the multiphase initialization process should be documented in detail in the FESFA.

Also, an added benefit of the phase specific synchronization points acting as guards or gates in regulating the advancement of the multiphase initialization process is that the unachieved synchronization points can be useful in determining where the process went wrong.

7.2.1.4. Time Management and Synchronization

Having completed multiphase initialization, the Master Federate will then setup HLA time management and synchronize the federation execution in preparation for transitioning to an execution state (see Figure 7-8). At this point, the Master Federate will achieve and wait for the “initialization_started” synchronization point. Next, if Auto-Provide was enabled prior to the Master Federate disabling it for initialization, the Master Federate will re-enable Auto-Provide. Now, the Master Federate registers the “initialization_completed” synchronization point. This is a marker synchronization point. The Master Federate will never achieve this synchronization point. Any Late Joiner federates that have come in during the initialization process will get the announcement of “initialization_completed” and be released to start their initialization process.

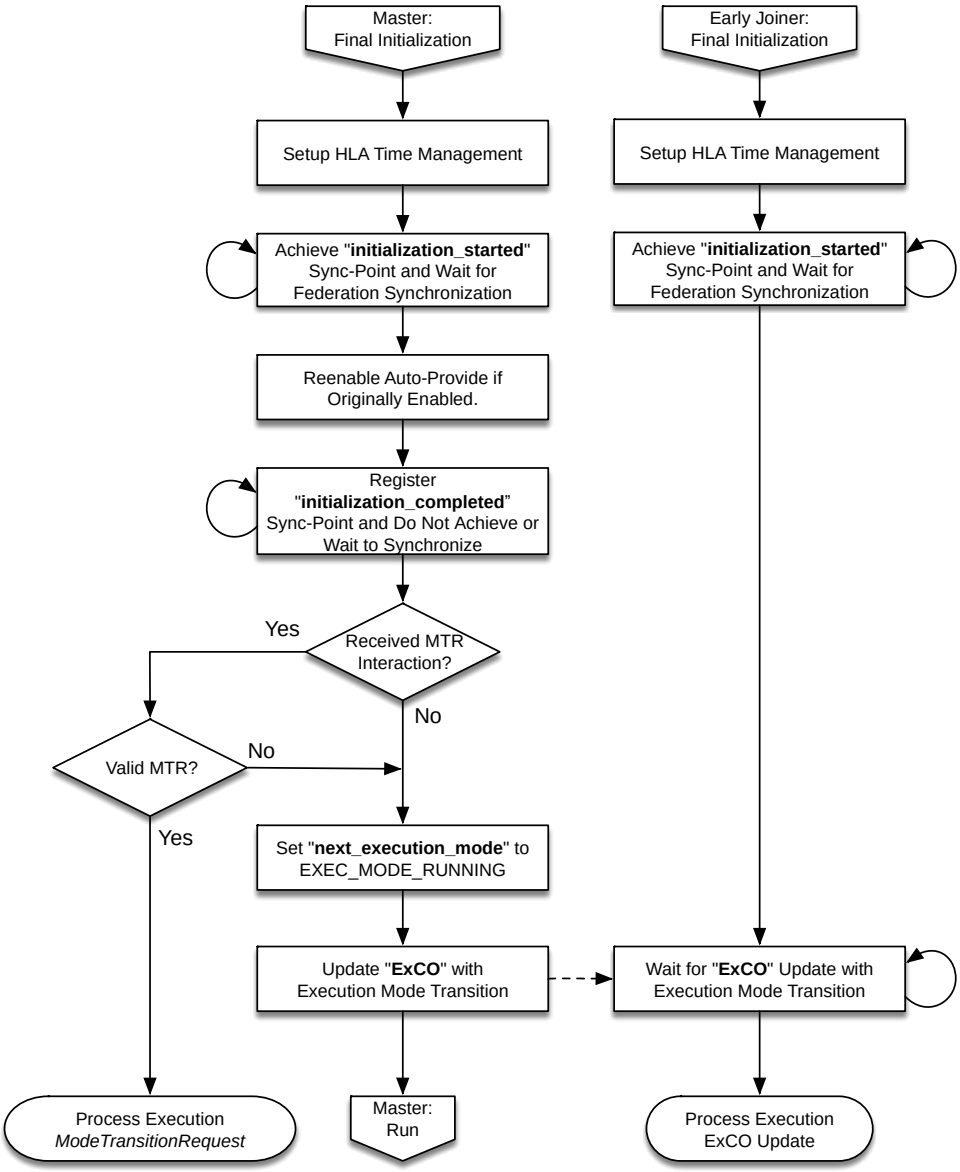

Figure 7-8: HLA Time Management, Synchronization, and Transition to Execution

At this point, the Master Federate checks for any MTRs received during initialization. If an MTR is received during initialization, the Master Federate will validate the MTR. If the MTR is valid, the Master Federate then proceeds to the ModeTransitionRequest process described in Section 7.4.2. If the MTR is not valid or no MTR is received during initialization, the Master Federate sets the “next_execution_mode” in the ExCO to EXEC_MODE_RUNNING and updates the ExCO. Then the federate proceeds to the Master run process described in Section 7.4.4.

This marks the end of the initialization process. The Master Federate will then proceed to the appropriate execution mode (see Section 7.4).

7.2.2. Early Joiner

Any federate that joins into the federation execution prior to the Master Federate recognizing the last required federate and registering the “initialization_started” synchronization point is considered an Early Joiner (see Figure 7-4). Note that the “initialization_started” synchronization point is registered with only the currently joined federates as members of the synchronization set.

7.2.2.1. HLA Initialization

The next step in the SpaceFOM initialization process is to setup and initialize the HLA infrastructure needed by the federate to operate as a federate and to coordinate with the Master Federate (see Figure 7-5).

The federate begins by waiting for announcement of the “objects_discovered”, “root_frame_discovered”, and multiphase initialization synchronization points. The “objects_discovered” synchronization point is used to mark the point at which all required objects have been discovered by all the federates taking part in the initialization process. The “root_frame_discovered” synchronization point is used to mark the point at which the root reference frame for this federation execution has been discovered by all federates participating in the initialization process. The multiphase initialization synchronization points are used in the multiphase initialization process described in Section 7.2.2.3.

The federate then sets up any needed RTI handles. The next step is to subscribe to the Execution Configuration object.[23] As described above (see Section 7.1.3), the Execution Configuration object plays an important role in the coordination and control of the federation execution. Only a single Execution Configuration object instance named “ExCO” will ever exist in a given federation execution.

The next step is to publish the MTR Interaction and then subscribe any federate specific object classes.[23] This will be followed by the reservation of all federate specific object instance names, waiting for the instance name reservation success or failure callbacks. At this point, the federate will register the federation specific object instances. The federate now waits for all required object instances to be discovered by object instance name. Finally, the federate achieves the “objects_discovered” synchronization point and waits for synchronization. This ensures that all objects required by those federates participating in early initialization have been discovered and that those federates will receive any updates for those object instances.

7.2.2.2. Epoch and Root Reference Frame Discovery

The federate is now ready to discover the federation execution simulation scenario time epoch and root reference frame. A flow chart of this process can be seen in Figure 7-6.

The federate starts by waiting for an ExCO update with the simulation scenario time epoch. This provides the simulation scenario time for use in the computation of the root reference frame and the multiphase initialization process.

At this point in the initialization process, the federate must determine if it is responsible for publishing the root reference frame in the federation execution’s reference frame tree. If so, the federate will update the root ReferenceFrame object attributes. If the federate does not publish the root ReferenceFrame object, then the federate must discover the root ReferenceFrame object from the discovered ReferenceFrame objects. The federate can determine the root reference frame by finding the first ReferenceFrame object instance with a null parent frame name (zero-length string). Alternatively, it can wait until the next ExCO update with the root reference frame name and use that.

After waiting for the ExCO update with the root reference frame name, the federate will achieve and wait for the “root_frame_discovered” synchronization point. Once the federation is synchronized on the “root_frame_discovered” synchronization point, the federate will then be positioned to begin the multiphase initialization process.

7.2.2.3. Multiphase Initialization

The multiphase initialization part of the initialization process is a loop on a predetermined number of data exchanges between the Master Federate and any other Early Joiner federates. Each loop is guarded with a multiphase initialization synchronization point. The multiphase initialization process shall be documented in the FESFA. Each loop (phase) in the multiphase initialization sequence has the potential for sending phase dependent initialization data, receiving phase dependent initialization data, and waiting on a phase dependent synchronization point (see Figure 7-7).

7.2.2.4. Time Management and Synchronization

Having completed multiphase initialization, the federate will now setup HLA time management and synchronize with the federation execution in preparation for transitioning to an execution state (see Figure 7-8). At this point, the federate will achieve and wait for the “initialization_started” synchronization point. The federate will ignore the announcement of the “initialization_completed” synchronization point. This is a marker synchronization point. It will never be achieved by the Master Federate and will therefore never be synchronized. Therefore, it can optionally be achieved by the Early Joiner federates. Any Late Joiner federates that have come in during the initialization process will get the announcement of the “initialization_completed” synchronization point and start their initialization process (see Section 7.2.3).

Once synchronized, the federate waits for an ExCO update with the specified execution mode transition. The federate then goes to the procedure to process ExCO updates for mode transition to execution as described in Section 7.4.

7.2.3. Late Joiner

Any federate that joins into the federation execution after the Master Federate recognizes the last required federate and registers the “initialization_started” synchronization point is considered a Late Joiner (see Figure 7-4). Note that the “initialization_started” synchronization point is registered with only the currently joined federates as members of the synchronization set. As a result, Late Joiner federates will never receive the announcement of the “initialization_started” synchronization point. A Late Joiner federate will wait for the announcement of the “initialization_completed” synchronization point. This is a marker synchronization point. It will never be achieved by the Master Federate. Any Late Joiner federates that have come in during the initialization process will get the announcement of the “initialization_completed” synchronization point and start their initialization process (see Section 7.2.3). Upon getting the announcement, the Late Joiner federate can begin its initialization process.

The Late Joiner initialization process is much like the Early Joiner initialization process but without the intermediate synchronization with the Master, and there is no predefined multiphase initialization step. This makes the Late Joiner initialization process a shorter and simpler process flow, as shown in Figure 7-9.

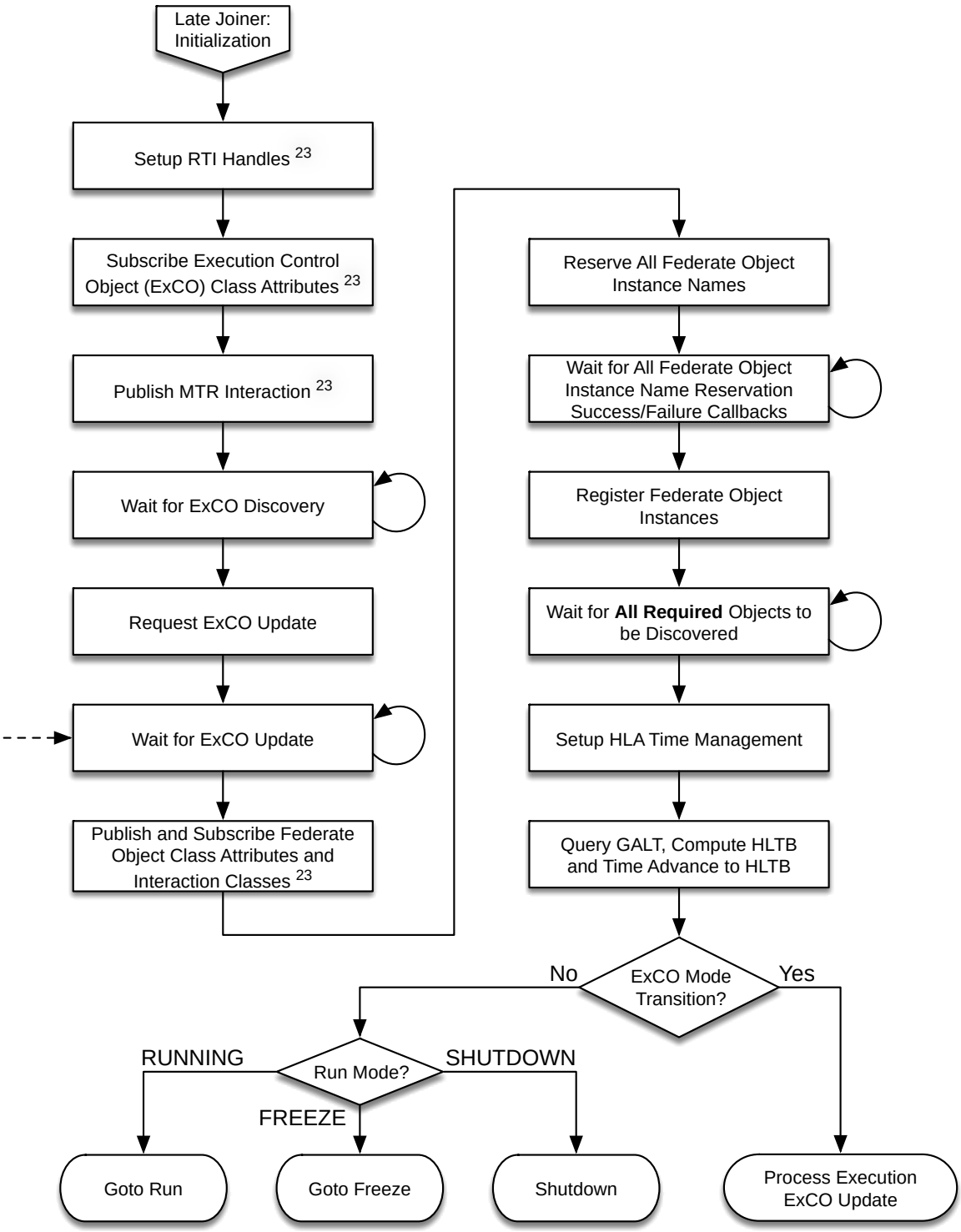

Figure 7-9: Late Joiner Initialization

The federate begins by setting up any needed RTI handles. The next step is to subscribe to the Execution Configuration object.[23] As described above (see Section 7.1.3), the Execution Configuration object plays an important role in the coordination and control of the federation execution. Only a single Execution Configuration object instance named “ExCO” shall ever exist in a given federation execution. The next step is to publish the MTR Interaction.[23] The federate then waits for the discovery of the ExCO object instance and then requests an update. The federate then waits for the update to the ExCO object instance. This will provide the federate with the current time, mode, root reference frame, and LCTS information.

The next step is to publish and subscribe any federate specific object classes.[23] This will be followed by the reservation of all federate specific object instance names, waiting for the instance name reservation success or failure callbacks. At this point, the federate will register the federation specific object instances. The federate now waits for all required object instances to be discovered.

The federate will then setup HLA time management. Then the federate will query for the federation execution’s GALT. The federate will then use the GALT along with the LCTS received in the ExCO update to compute the next common HLTB:

$$HLTB = (floor(GALT/LCTS) + 1) * LCTS$$

The federate will then issue a Time Advance Request to move to the HLTB, and wait until it receives a Time Advance Grant to that time.

At this point, the federate is ready to transition to an execution mode. If the current ExCO indicates that a mode transition is in progress, the federate goes to the procedure to process ExCO update for mode transition to execution as described in Section 7.4. If the current ExCO does not indicate a mode transition is in progress, then the federate proceeds to the current execution mode: EXEC_MODE_RUNNING, EXEC_MODE_FREEZE, or EXEC_MODE_SHUTDOWN.

7.3. Execution

Once initialization is completed, all federates transition into one of three possible execution modes: run, freeze, or shutdown. The general execution mode flow can be seen in Figure 7-10. Note that this is a general flow and that there will be differences in the execution flow of federates depending on their role in the federation execution and their capabilities. For instance, some federates might have CTE while others do not. However, if any federate requires CTE, then both the Master Federate and Pacing Federate shall have CTE. The Master Federate shall use CTE for ExCO mode management.

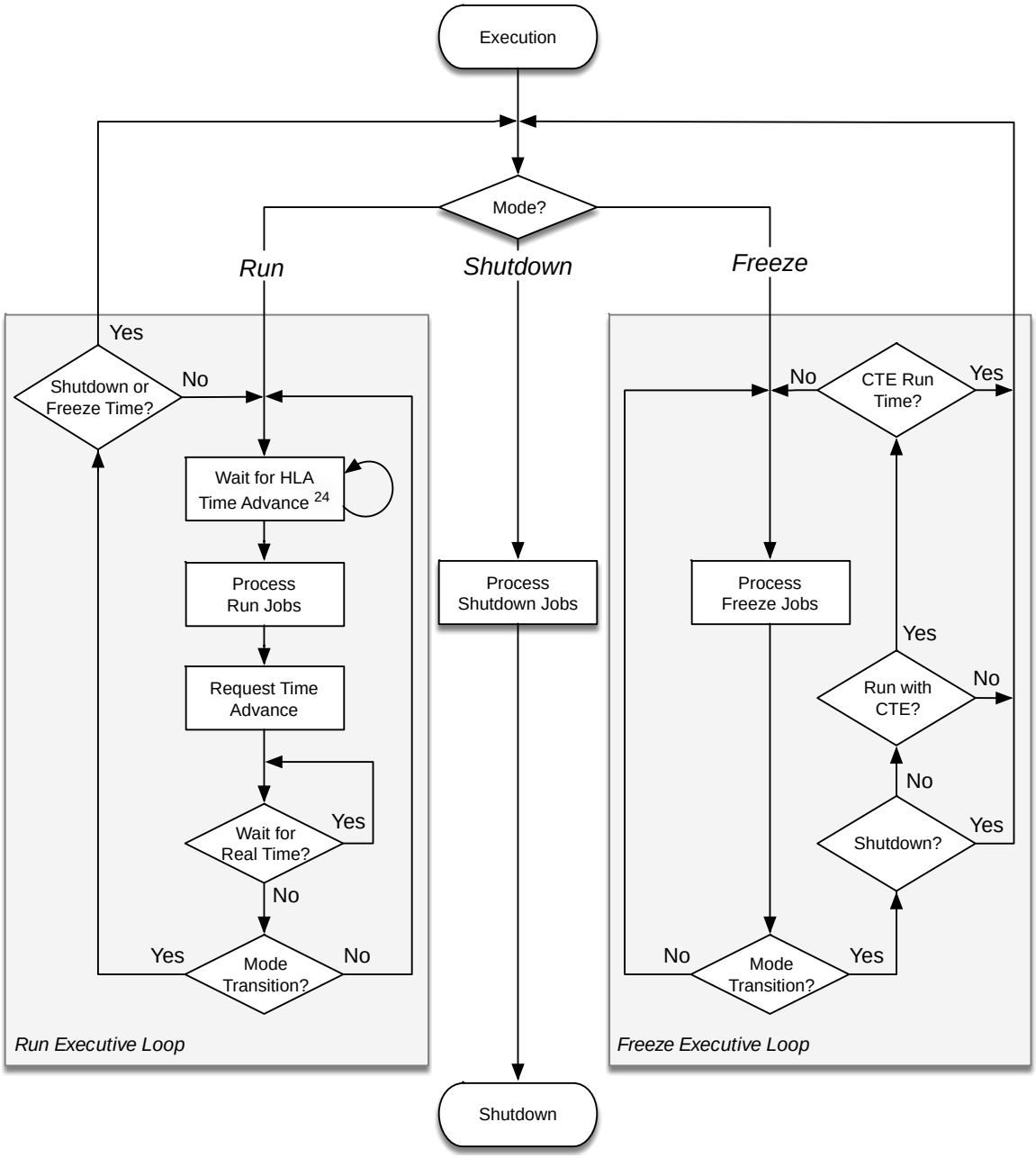

Figure 7-10: SpaceFOM Execution Overview

As with most simulation architectures, the SpaceFOM execution architecture is composed of executive looping constructs; in this case, two executive loops: a run executive loop and a freeze executive loop. A federate exits the execution phase by making a mode transition to shutdown.

A federate enters the run mode either directly from initialization or through a mode transition from freeze mode. Similarly, a federate enters a freeze mode directly from initialization or through a mode transition from run mode. A federate can transition to the shutdown mode from any execution mode.

Once a federate enters run mode, the federate enters into a run execution loop that starts with waiting for an HLA time advance (usually a Time Advance Grant (TAG) callback) to the last TAR to a specific (current) federation HLT that will correspond to the current SST.[24]

After receiving the TAG, the federate then performs any federate specific computations related to a running state. This is usually in the form of function calls (Run Jobs). Upon completion of the current SST computations, the federate makes a TAR to the HLT corresponding to the next SST. Refer to Section 4.4.2 for time step constraints. If the federate is tied to a real-time clock, like a Pacing federate, the federate waits until the CCT reaches the time corresponding to the requested SST. In this case, the CCT may be either the computer's clock or CTE.

Finally, the federate checks if an ExCO update has been received with a valid mode transition during this run loop or if a previously valid Freeze mode transition is pending. If not, the run executive loop returns to wait for the TAG to begin the next loop. If a Freeze mode transition is pending but the federate has not yet reached the appointed freeze time, the run executive loop returns to wait for the TAG to begin the next loop. If a Freeze mode transition is pending and the run executive has advanced to the appointed time for the freeze mode transition, the federate enters freeze mode. If a shutdown mode transition has been received, the federate enters shutdown mode.

Once a federate enters freeze mode, the federate enters into a freeze executive loop that starts by performing any federate specific computation related to a freeze state. The federate will remain in freeze mode until an ExCO update has been received and is processed to transition the federate to a different execution mode (run or shutdown).

Note that time does not advance when in freeze mode. However, the CCT will advance. This means that the CCT reference point with respect to the HLT and SST time lines will have to be reset. This applies to both internal computer clocks or CTE based clocks. It also means that, when a run mode transition is received and CTE is used, the freeze loop will continue until the CTE run time is reached (see Figure 7-10).

Once a federate enters shutdown mode, the federate performs any federate specific computations related to shutting down and then terminates. There are no mode transitions from shutdown mode.

7.4. Mode Transitions

As shown in Figure 7-1, a SpaceFOM-compliant federate's executive state can be characterized by two principal executive phases: initialization and execution. However, this characterization is a little too high level for a functional implementation. Fortunately, it does not require much more in the way of complexity. A SpaceFOM-compliant federate will exist in one of five (5) executive initialization or execution modes: uninitialized, initializing, running, freeze or shutdown. These modes and the transitions between them are defined and described in this section. The first two modes (uninitialized and initializing) are associated with the initialization section of Figure 7-1. The last three modes (running, freeze and shutdown) are associated with the execution section of Figure 7-1.

In general, the initialization mode transitions (uninitialized and initializing) are handled internally to each federate conditional to the role of the federate (Master, Early Joiner, or Late Joiner) and its progression through the initialization process. Specifically, the transition from uninitialized through initializing is gated by sync-points, ExCO updates, and the progression of the federation execution through the SpaceFOM initialization process described above in Section 7.2. In contrast, the execution mode transitions (running, freeze or shutdown) can be triggered through mode transition requests from any participating federate in the federation execution (see Figure 7-10).

[24] On first entry into the Run Executive Loop, the HLA time advance may come from initially enabling time regulation.

Mode transitions are controlled using two principal mechanisms: MTR interactions (see Section 7.1.4) and ExCO attribute updates (see Section 7.1.3). Any federate can request a mode transition by issuing an MTR interaction requesting a transition to another execution mode (running, freeze or shutdown). It is important to note that the Master Federate may ignore these mode transition requests if the federation execution state is not appropriate for the requested mode transition (see Table 7-1). A Master Federate is the only federate that can control mode transitions using the ExCO singleton object instance in the federation execution (see Rule 3-8 and Rule 3-9). The Master Federate will control the federation execution mode by updating the attribute values in the federation execution's ExCO indicating the appropriate mode transition.

In order to better understand the mode transition process, we should take a closer look at the definitions of the principal executive modes defined in the SpaceFOM.

7.4.1. Principal Executive Modes

The SpaceFOM defines the following 5 executive modes (2 initialization modes and 3 execution modes):

- **EXEC_MODE_UNINITIALIZED**: This is the mode in which all SpaceFOM-compliant federates will be in upon startup and prior to transitioning into their federate specific initialization. For the Master Federate and any Early Joiner federates, this will be up to the point they receive the initialization_started sync-point announcement. For a Late Joiner federate, this will be up to the point that they receive the initialization_completed sync-point announcement.
- **EXEC_MODE_INITIALIZING**: This is the mode in which all SpaceFOM-compliant federates will be until transitioning into one of the two principal executive execution loops (run or freeze) or to shutdown. For all federates, this occurs as the federate transitions from final initialization to the commanded execution mode. For Master and Early Joiner federates, see Figure 7-8. For Late Joiner federates, see Figure 7-9. From initialization mode, a federation execution may transition to run, freeze or shutdown modes.
- **EXEC_MODE_RUNNING**: This is the primary working executive execution mode for all SpaceFOM-compliant federates (see Figure 7-10). This is referred to as "run mode". SST and HLT are advanced during run mode and the states of simulation objects are propagated in coordination with SST. This is the default executive execution mode into which all federates will transition after initialization unless commanded to another mode by an ExCO mode transition update prior to leaving initialization. A federate remains in run mode until an ExCO mode update is received to transition into either freeze mode or shutdown. From run mode, a federation execution may transition to either freeze mode or shutdown mode.
- **EXEC_MODE_FREEZE**: This is the mode into which all SpaceFOM-compliant federates will enter when SST and HLT are not advancing but the federates are still executing. This is referred to as "freeze mode" or "pause mode". This is an executive loop like the run loop (see Figure 7-10) but in which neither SST is advanced nor federates states propagated. A federate remains in freeze mode until an ExCO mode update is received to transition into either run mode or shutdown. From freeze mode, a federation execution may transition to either run mode or shutdown mode.
- **EXEC_MODE_SHUTDOWN**: This is the termination mode for all SpaceFOM-compliant federates. This is referred to as "shutdown mode". A federate transitions to shutdown mode prior to withdrawing from the federation execution. There is no transition path from shutdown mode.

By default, a SpaceFOM-compliant federation execution will transition from initialization mode EXEC_MODE_INITIALIZING into execution mode EXEC_MODE_RUNNING or "run mode". See the run mode loop described in Figure 7-10 above. However, in some cases, the federation execution may need to suspend the advancement of SST to wait on some condition external to the federation execution. In this case, the federation execution will transition to execution mode EXEC_MODE_FREEZE or "freeze mode". Specifically, all federates in the federation execution will transition from their run executive loops to their freeze executive loops. Once the external condition is satisfied, the federation execution may transition back into run mode. Finally, the federation will not run forever and will need to transition to execution mode EXEC_MODE_SHUTDOWN or "shutdown mode" to terminate the federation execution.

7.4.2. Master Federate Mode Transitions

The Master Federate plays a number of important roles in the life and management of a SpaceFOM-compliant federation execution. One of these roles is the management of federation execution mode transitions. The Master Federate employs two HLA based mechanisms to do this: MTR interactions and ExCO attribute updates. While any federate in the federation execution may request an execution mode transition with an MTR interaction, only the Master Federate can issue federation execution mode transitions using attribute updates in the federation execution's singleton ExCO.

While the Master Federate will receive MTR interactions from participating federates, not all of these MTRs will be accepted. The Master Federate will filter mode transition requests based on two characteristics: relevance and precedence. Relevance is based on the current federation execution mode as compared to the requested federation execution mode transition. A table of valid mode transition requests based on current federation execution mode is presented in Table 7-1.

MTR Interaction Mode / ExCO Current Mode	MTR_GOTO_RUN	MTR_GOTO_FREEZE	MTR_GOTO_SHUTDOWN
EXEC_MODE_UNINITIALIZED	ignore	ignore	accept
EXEC_MODE_INITIALIZING	ignore [25]	accept	accept
EXEC_MODE_RUNNING	ignore	accept	accept
EXEC_MODE_FREEZE	accept	ignore	accept
EXEC_MODE_SHUTDOWN	ignore	ignore	ignore

Table 7-1: Master Mode Transition Request Validation Matrix

There is also a possibility that the Master Federate may receive another MTR while processing an MTR. When this occurs, the Master Federate relies on the following set of precedence rules:

1. MTR to EXEC_MODE_SHUTDOWN: In all cases, EXEC_MODE_SHUTDOWN has precedence over transitions to either EXEC_MODE_RUNNING or EXEC_MODE_FREEZE. If an MTR is received for EXEC_MODE_SHUTDOWN, the Master Federate will interrupt any processing for the current mode transition and process the transition to shutdown. This is true even if the mode transition attributes have been sent out in an ExCO attribute update and the federation is waiting on synchronization (see Rule 7-17).
2. All other MTRs when shutting down: In all cases, EXEC_MODE_SHUTDOWN has precedence over transitions to either EXEC_MODE_RUNNING or EXEC_MODE_FREEZE. If an MTR is received for either EXEC_MODE_RUNNING or EXEC_MODE_FREEZE while processing an EXEC_MODE_SHUTDOWN MTR, the Master Federate will ignore those MTRs.

[25] The default mode when entering execution is EXEC_MODE_RUNNING.

3. Duplicate MTRs: If an MTR is received with a request to a mode transition already being processed, the Master Federate will ignore those MTRs.
4. MTR to EXEC_MODE_FREEZE during initialization: The default execution mode coming out of initialization is EXEC_MODE_RUNNING. If an MTR to EXEC_MODE_FREEZE is received during the initialization process for the federation, the Master Federate will set the federation execution mode to EXEC_MODE_FREEZE for the ExCO mode transition update between initialization and execution (see Figure 7-8).

Figure 7-11 shows the general decision logic associated with the Master Federate's mode transition control when processing an MTR. The first step is to validate the MTR based on the validation criteria in Table 7-1 and the precedence rules. If the MTR is not valid, then the MTR is ignored and the Master Federate will remain in the current execution mode.

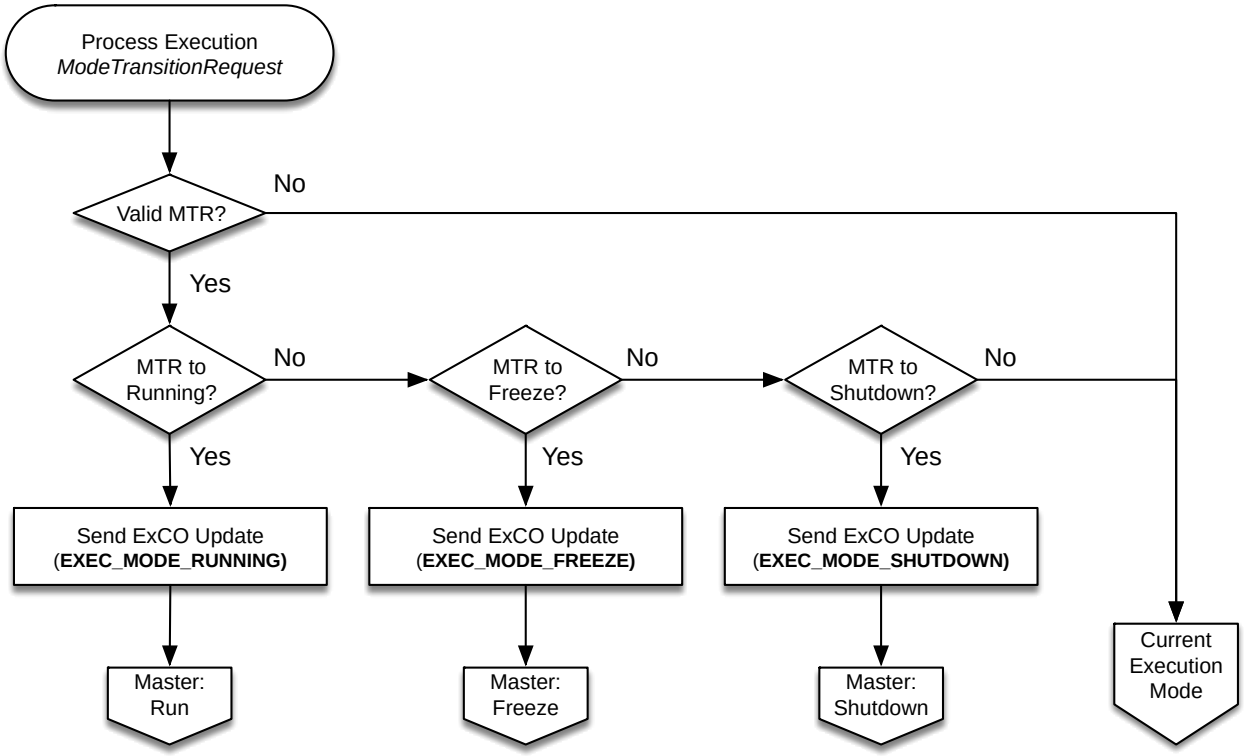

Figure 7-11: Master Federate Mode Transition Overview

If the MTR is valid, the requested mode transition is determined (EXEC_MODE_RUNNING, EXEC_MODE_FREEZE, or EXEC_MODE_SHUTDOWN). Once the requested mode transition is determined, an ExCO update is sent out with the appropriate mode. The Master Federate then transitions to the corresponding execution mode (see Sections 7.4.4, 7.4.5, or 7.4.6 respectively).

There is a special consideration for the EXEC_MODE_FREEZE mode transition. The Master Federate is required to compute a freeze time in the future for all federates to go to freeze. However, not all federates in a SpaceFOM-compliant federation execution have the same time step. In order for all federates in the federation execution to freeze on a natural time step for any federate, the freeze time will have to be on a common HLTB. This does not have to be the next available common HLTB but must be on a common HLTB in the future.

7.4.3. General Federate Mode Transitions

As stated previously, any federate can request a mode transition but only a Master Federate can actually command a mode transition with an appropriate update to the federation execution's singleton ExCO. All federates must honor an ExCO commanded mode transition.

In general, upon receiving an ExCO commanded mode transition, a federate will process the commanded mode transition in a manner similar to the decision logic diagram in Figure 7-12. The first step is to determine if the ExCO is commanding a mode change. If it is not a mode change, the next step is to check for an update to the CTE reference epoch. If it is not a CTE update or if the federate is not using CTE, the federate will ignore the ExCO mode transition. If the federate is using CTE and this is a CTE reference epoch update, the federate will register the epoch update used to release the run mode CTE update wait loop shown in Figure 7-13.

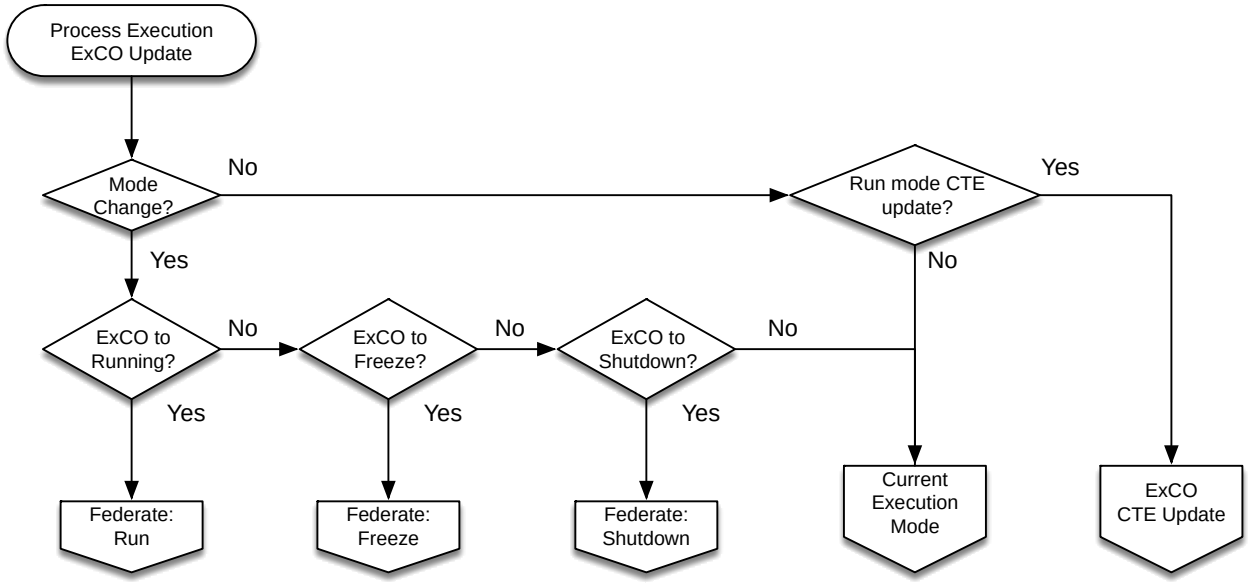

Figure 7-12: General Federate ExCO Mode Transition Overview

If the ExCO is commanding a mode transition, then the requested mode transition is determined (EXEC_MODE_RUNNING, EXEC_MODE_FREEZE, or EXEC_MODE_SHUTDOWN). Once the commanded mode transition is determined, the federate then transitions to the corresponding execution mode (see Sections 7.4.4, 7.4.5, or 7.4.6 respectively).

7.4.4. Transition to Run

The transition to run mode is probably the most complex of the execution mode transitions. In general, the transition to run mode is a coordinated activity managed by the Master Federate in coordination with all other federates in the federation execution. The Master Federate manages the transition with a combination of synchronization points, CTE epoch updates, and wait loops. The differing responsibilities in the run mode transition between the Master Federate and all other federates can be seen in Figure 7-13.

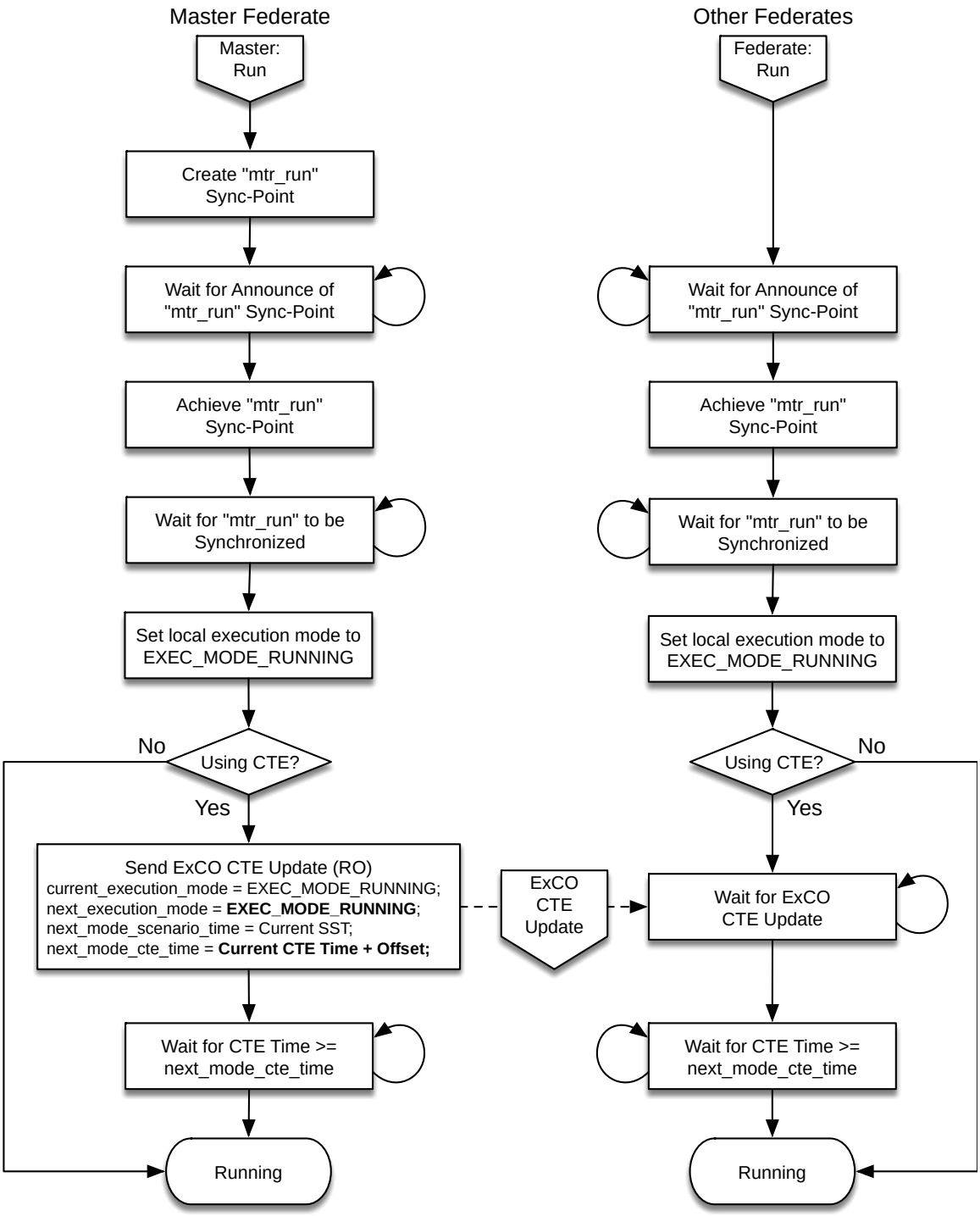

Figure 7-13: Run Mode Transition Overview

7.4.4.1. Master Federate Transition to Run

For the Master Federate, once a valid run MTR has been received and processed (see Section 7.4.2), the Master Federate will begin to process the run mode transition request. The first step is to create an “mtr_run” synchronization point. The Master Federate then waits, like all other federates, for the announcement of the “mtr_run” synchronization point. The Master Federate then achieves the synchronization point and enters a wait loop. The Master Federate waits in this loop until all other federates in the federation execution achieve the “mtr_run” synchronization point. At this point, the federation execution is synchronized on the “mtr_run” synchronization point and the wait loop is released. The Master Federate then marks its local current execution mode as EXEC_MODE_RUNNING and by association the federation execution current_execution_mode state to EXEC_MODE_RUNNING. If this federation execution is not using CTE, then the Master Federate immediately goes into its run loop.

In the special case where CTE is being used in the federation execution, the Master will have CTE (see Rule 7-4). In this case, the Master Federate will be responsible for determining the appropriate new CTE time epoch, formulating the appropriate ExCO parameters and publishing the corresponding update to the ExCO attributes. The Master Federate will then enter into a CTE time based wait loop that checks the current CTE time against the new CTE epoch. The Master Federate goes into its run loop at the point where the current CTE time equals or exceeds the new CTE epoch.

The Master Federate is now in run mode.

7.4.4.2. General Federate Transition to Run

Federation execution federates, other than the Master Federate, follow a process flow complementary to that of the Master Federate. Once a non-Master Federate receives a valid ExCO mode transition update to run, the federate will begin to process the mode transition command (see Figure 7-12). The first step is to wait for the announcement of the “mtr_run” synchronization point. The federate then achieves the synchronization point and enters a wait loop. The federate waits in this loop until all other federates in the federation execution achieve the “mtr_run” synchronization point. At this point, the federation execution is synchronized on the “mtr_run” synchronization point and the wait loop is released. The federate then marks its local current execution mode as EXEC_MODE_RUNNING to match the federation current_execution_mode state. If this federate is not using CTE, then the federate immediately goes into its run loop.

In the special case where this federate uses CTE, the federate will enter into a gated execution loop waiting on an ExCO CTE epoch update from the Master Federate. The federate will then enter into a CTE time based wait loop that checks the current CTE time against the new CTE epoch. The federate goes into its’ run loop at the point where the current CTE time equals or exceeds the new CTE epoch.

The federate is now in run mode.

7.4.5. Transition to Freeze

Like the previously described run mode transition, the transition to freeze mode is a coordinated activity managed by the Master Federate in coordination with all other federates in the federation execution. The Master Federate manages the transition with a combination of synchronization points and wait loops. The differing responsibilities in the freeze mode transition between the Master Federate and all other federates can be seen in Figure 7-14.

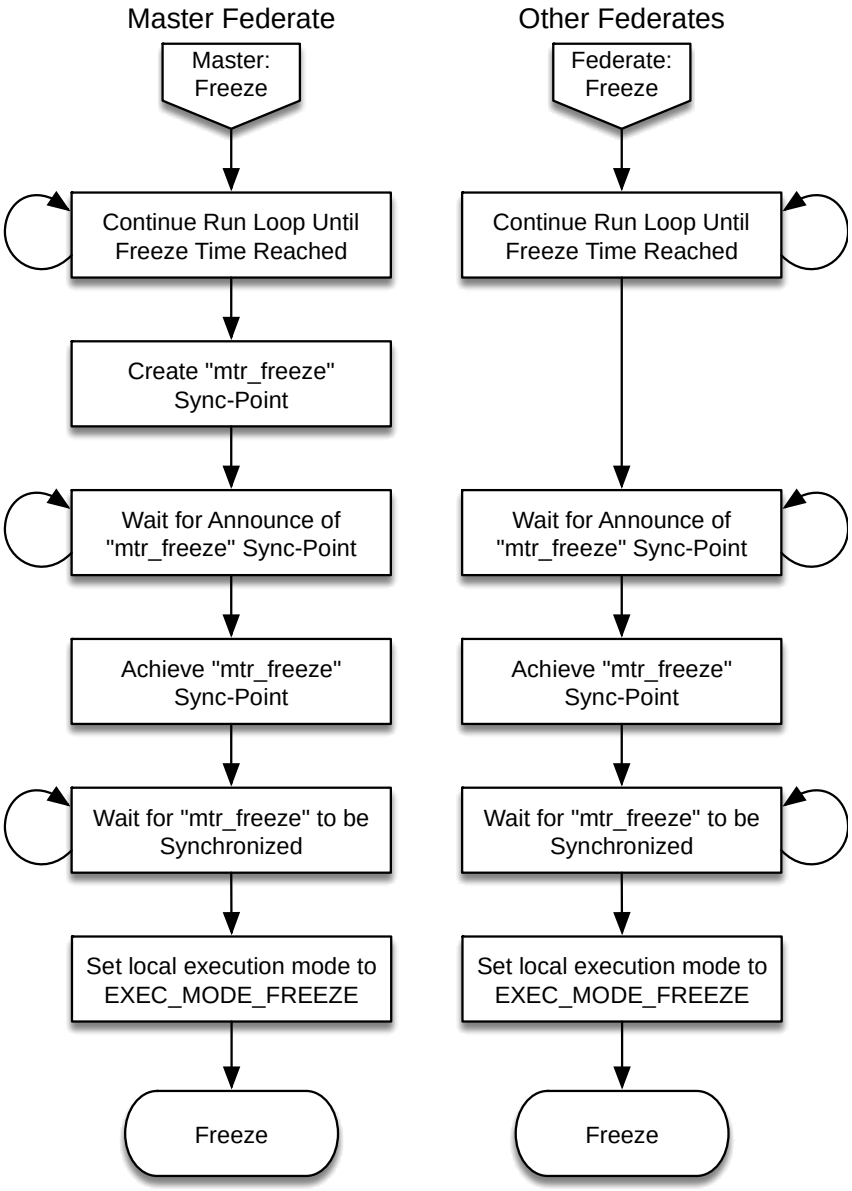

Figure 7-14: Freeze Mode Transition Overview

7.4.5.1. Master Federate Transition to Freeze

For the Master Federate, once a valid freeze MTR has been received, the Master Federate will begin to process the freeze mode transition request (see Figure 7-11). However, unlike either run mode or shutdown mode transitions, freeze mode transitions are timed transitions based on SST. Specifically, the ExCO update sent out by the Master Federate with a freeze mode transition will include an SST based time in the future for the federation to go to freeze. Therefore, it is important to note that the wait block in Figure 7-14 is not a blocking wait loop but really a state condition in the run loop that checks for a freeze transition time to switch execution from the Master Federate's run loop into this freeze process flow (see Figure 7-10). Freeze time is reached when the SST is greater than or equal to the SST of next mode transition (SST >= next_mode_scenario_time), in this case, the time to transition to freeze.

Once the Master Federate reaches the indicated freeze time, the Master Federate creates an “mtr_freeze” synchronization point. The Master Federate then waits, like all other federates, for the announcement of the “mtr_freeze” synchronization point. The Master Federate then achieves the synchronization point and enters a wait loop. The Master Federate waits in this loop until all other federates in the federation execution achieve the “mtr_freeze” synchronization point. At this point, the federation execution is synchronized on the “mtr_freeze” synchronization point and the wait loop is released. The Master Federate then marks its local current execution mode as EXEC_MODE_FREEZE and by association the federation execution current_execution_mode state to EXEC_MODE_FREEZE. The Master Federate immediately goes into its freeze loop.

The Master Federate is now in freeze mode.

7.4.5.2. General Federate Transition to Freeze

For a general federate, once a valid ExCO mode transition command has been received, the federate will begin to process the freeze mode transition command (see Figure 7-12). However, unlike either run mode or shutdown mode transitions, freeze mode transitions are timed transitions based on SST. Specifically, the ExCO update with a valid freeze mode transition will include an SST based time in the future for the federation to go to freeze. Therefore, it is important to note that the wait block in Figure 7-14 is not a blocking wait loop but really a state condition in the run loop that checks for a freeze transition time to switch execution from the federate’s run loop into this freeze process flow (see Figure 7-10). Freeze time is reached when the SST is greater than or equal to the SST of next mode transition (SST >= next_mode_scenario_time), in this case, the time to transition to freeze.

Once the federate reaches the indicated freeze time, the federate then waits for the announcement of the “mtr_freeze” synchronization point. The federate then achieves the synchronization point and enters a wait loop. The federate waits in this loop until all other federates in the federation execution achieve the “mtr_freeze” synchronization point. At this point, the federation execution is synchronized on the “mtr_freeze” synchronization point and the wait loop is released. The federate then marks its local current execution mode as EXEC_MODE_FREEZE to match the federation current_execution_mode state. The federate immediately goes into its freeze loop.

The federate is now in freeze mode.

7.4.6. Transition to Shutdown

Like the previously described run mode and freeze mode transitions, the Master Federate manages the transition to shutdown mode using the ExCO and the “mtr_shutdown” synchronization point. Unlike run mode and freeze mode transitions, the Master Federate does not coordinate the transition to shutdown mode with the “mtr_shutdown” synchronization point. Since shutdowns may be caused by any number of unforeseen events, not all federates can be counted on to achieve a synchronization point. This could result in a deadlock during shutdown. As a result, mode transitions to shutdown are immediate and uncoordinated. However, the “mtr_shutdown” synchronization point is used to avoid a possible race condition between when the last federate in a “shutdown” federation execution resigns and a possible new “late joiner” federate joins.

If a coordinated shutdown is desired, the federation execution can be transitioned to freeze mode first and then to shutdown after entering freeze mode.

The differing responsibilities in the freeze mode transition between the Master Federate and all other federates can be seen in Figure 7-15.

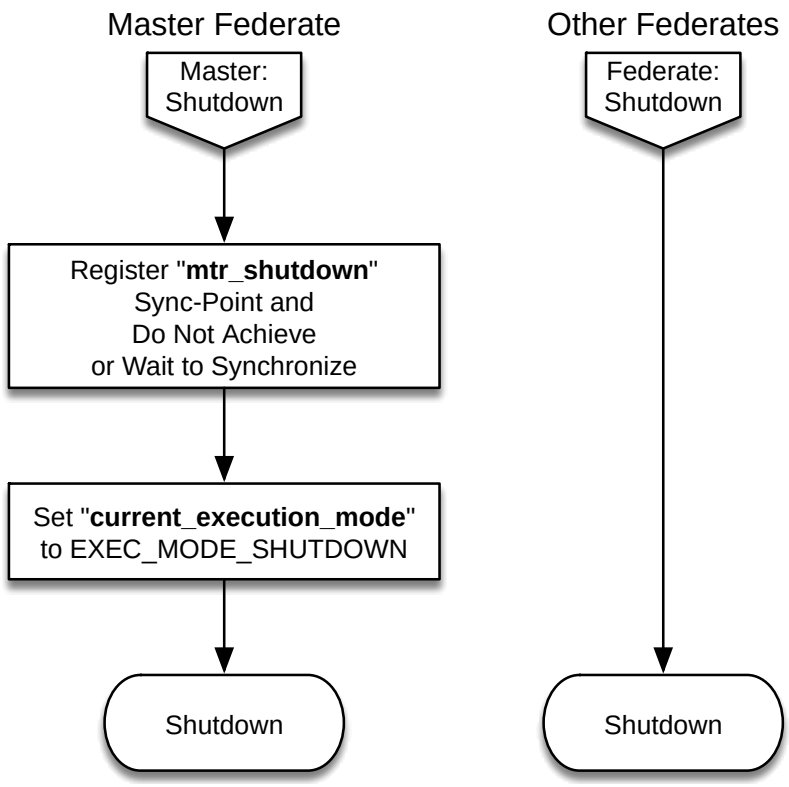

Figure 7-15: Shutdown Mode Transition Overview

7.4.6.1. Master Federate Transition to Shutdown

For the Master Federate, once a valid shutdown MTR has been received, the Master Federate will begin to process the shutdown mode transition request (see Figure 7-11). After the Master Federate sends out the ExCO update with EXEC_MODE_SHUTDOWN, the Master Federate will register an “mtr_shutdown” synchronization point, mark its local current execution mode as EXEC_MODE_SHUTDOWN, and immediately go into its shutdown process. Note that the “mtr_shutdown” synchronization point, like the “initialization_completed” synchronization point, is used as a marker and is not achieved by the Master Federate. In this case, the “mtr_shutdown” synchronization point should not be achieved by any federate.

The Master Federate is now in shutdown mode.

7.4.6.2. General Federate Transition to Shutdown

For a general federate, once a valid ExCO mode transition command has been received, the federate will begin to process the shutdown mode transition command (see Figure 7-12). Once the mode transition to EXEC_MODE_SHUTDOWN is confirmed, the federate then marks its local current execution mode as EXEC_MODE_SHUTDOWN to match the federation current_execution_mode state and immediately goes into its shutdown process.

The federate is now in shutdown mode.

7.5. The Execution Management FOM Module (SISO_SpaceFOM_management)

This SpaceFOM Execution Management module (SISO_SpaceFOM_management) provides the specifications for execution configuration objects, management objects, interactions, and synchronization points. These are used to convey federation mode transition information and to coordinate federation mode transition behavior. The entity types in this FOM module are presented in Appendix Section C.3 along with tables describing the types and their relationship to one another.

7.6. Rules

Based on the preceding discussion, we can infer some general and specific rules associated with SpaceFOM execution control. These rules are segregated into CTE rules, initialization rules, and mode transition rules.

7.6.1. CTE Rules

Rule 7-1: Document CTE Standard in FESFA

Requirement: When CTE is used, the CTE standard shall be documented in the FESFA.

Rationale: For compatibility, all time managed federates that use CTE for time synchronization need to use the same CTE standard. The FESFA should document the standard used.

Rule 7-2: Document Federate CTE Capabilities in FCD

Requirement: A SpaceFOM-compliant federate shall document the federate's CTE capabilities in its associated Federation Compliance Declaration.

Rationale: A federation designer needs to understand the CTE capabilities and possible requirements of a potential participating federate.

Rule 7-3: Use Compatible CTE Standard and Time Representation

Requirement: All federates in a federation execution requiring CTE based time synchronization shall use a compatible CTE standard and CTE time representation.

Rationale: For compatibility, all time managed federates that use CTE for time synchronization need to use a compatible CTE standard and CTE time representation.

Rule 7-4: Master and Pacing Federates Use CTE If Any Federate Requires CTE

Requirement: The Master Federate and Pacing Federate shall use CTE if any federate in the federation execution requires CTE time synchronization.

Rationale: In general, the Master Federate is responsible for managing the federation execution. In particular, the Master Federate controls mode transitions through the use of updates to the ExCO. The ExCO contains the *next_mode_cte_time* attribute associated with coordinating mode transition for CTE synchronized systems. The Master Federate can only provide this parameter if it is also using a compatible CTE system. To maintain a CTE synchronized system, the Pacing Federate must also use a compatible CTE system.

7.6.2. Initialization Rules

Rule 7-5: Master Federate Disables and Restores Auto-Provide During Initialization

Requirement: The Master Federate shall disable Auto-Provide at the start of initialization and restore the original value at the end of initialization.

Rationale: The SpaceFOM initialization process relies on a strict ordering on the delivery of ExCO object instance attribute updates from the Master Federate. The delivery of these updates is used to coordinate the initialization process. The HLA RTI switch Auto-Provide tells the RTI to automatically solicit updates from instance attribute owners when an object is discovered. Having Auto-Provide enabled during initialization will disrupt the initialization process, since ExCO updates will be solicited from the Master Federate whenever a new federate discovers the ExCO object instance. Therefore Auto-Provide needs to be disabled during the SpaceFOM Early Joiner initialization process. The original value of Auto-Provide should be restored once initialization is complete.

Rule 7-6: Required Federates Must Register with Unique Names

Requirement: Required federates shall register using unique names when joining the federation execution.

Rationale: Required federates are discovered by the Master Federate as part of the SpaceFOM initialization process. Required federates are recognized by federate name (see Rule 7-8). Therefore, required federates need to register their name when joining a SpaceFOM-compliant federation execution.

Rule 7-7: Document Required Federate Names In FESFA

Requirement: The unique names of required federates shall be documented in the FESFA.

Rationale: Required federates are discovered by the Master Federate as part of the SpaceFOM initialization process. Required federates are recognized by federate name (see Rule 7-6 and Rule 7-8). While HLA enforces the uniqueness of federate names, these names shall be recorded in the FESFA to mitigate the potential for name collisions.

Rule 7-8: Master Federate Discovers Required Federates By Name

Requirement: The Master Federate shall discover required federates by registered federate name.

Rationale: Required federates are discovered by the Master Federate as part of the SpaceFOM initialization process. Required federates must register a federation execution unique name when joining a federation execution. This name is available to the Master Federate to recognize when a required federate has joined the federation execution.

Rule 7-9: Required Object Instances Must Have Unique Names

Requirement: Required object instances shall have uniquely identifiable names when instantiated.

Rationale: Required object instances are registered and discovered early in the initialization process for the Master Federate and the Early Joiner federates. Specifically required object instances are determined by name. This can only be accomplished if all the required object instance names are unique across the federation execution when instantiated.

Rule 7-10: Required Object Instances Discovered By Reserved Object Instance Name

Requirement: Required object instances shall be discovered by a uniquely identifying reserved object instance name.

Rationale: The SpaceFOM initialization process provides for a set of required named object instances. This ensures that object instances required for a successful federation execution are present prior to leaving the initialization process. Reserved object instance names are available to SpaceFOM-compliant federates to satisfy this requirement.

Rule 7-11: Document Reserved Object Instance Names In FESFA

Requirement: The reserved names of all required object instances needed by a federation execution shall be documented in the Federation Execution Specific Federation Agreement.

Rationale: The object instance discovery process is a key component of the SpaceFOM initialization process. In order to ensure that all required object instances are accounted for in a federation execution, these names should be documented in the FESFA.

Rule 7-12: Document Reserved Object Instance Names in FCD

Requirement: The reserved names of all required object instances needed by a federate shall be documented in the federate's FCD.

Rationale: This is a means for a federate to convey to a federation execution designer that it requires specific object instances be provided as a prerequisite to participation. A federation execution designer can use the information in the FCD to determine the participating federates in a federation execution. The union of all the required object instances listed in the participating federates FCDs should provide the list needed for the FESFA (see Rule 7-11).

Rule 7-13: Document Multiphase Initialization Process In FESFA.

Requirement: The multiphase initialization process shall be documented in the Federation Execution Specific Federation Agreement.

Rationale: The SpaceFOM initialization process includes a section in which the Master Federate can manage the sequence-dependent initialization of federates. The satisfaction of these sequential dependencies can often be achieved through an iterative exchange of data between specified federates. Each of these iterations is called a phase. The entire process is called multiphase initialization. The multiphase initialization process is critical to the successful completion of a SpaceFOM-compliant initialization process. However, due to the indeterminate and potentially iterative nature of satisfying initialization dependencies, these dependencies and the associated process for resolving them need to be well documented in the FESFA.

Rule 7-14: Document Multiphase Initialization Synchronization Points in FESFA

Requirement: Multiphase initialization synchronization points shall be documented in the FESFA.

Rationale: The SpaceFOM multiphase initialization process uses synchronization points to coordinate the iterative process of resolving initialization data dependencies between participating federates. The Master Federate regulates the iteration process with these synchronization points. Other federates use them to acknowledge when their initialization data dependencies have been satisfied. Therefore, it is important that the multiphase initialization synchronization points be documented in the FESFA.

Rule 7-15: Master Federate Creates Named Multiphase Initialization Synchronization Points

Requirement: The Master Federate shall create a named synchronization point to regulate each loop in the multiphase initialization process.

Rationale: The Master Federate manages the SpaceFOM initialization process. This includes the multiphase initialization process. The multiphase initialization process consists of an iterative sequence of data exchanges between specified federates. The Master Federate keeps track of and regulates this iteration process through the use of named multiphase initialization synchronization points.

Rule 7-16: No ExCO Updates From Master Federate Prior To Initialization Object Discovery

Requirement: The Master Federate shall not send an ExCO update prior to achieving the "objects_discovered" synchronization point.

Rationale: Early in the initialization between the Master Federate and Early Joiner federates, the Early Joiner federates are in the process of subscribing to objects, including the ExCO. However, until the Early Joiner federates have successfully subscribed to the ExCO, any ExCO attribute updates will be lost. Restricting the Master Federate from publishing ExCO updates until after the "objects_discovered" synchronization point is achieved eliminates this issue.

Rule 7-17: Check for ExCO Mode Transitions in Wait Loops After Initialization Object Discovery

Requirement: After the federation is synchronized on the "objects_discovered" synchronization point, all federates shall check for ExCO mode transitions in any wait loops.

Rationale: There are natural wait loops for federates using the SpaceFOM execution control flow. An ExCO mode transition may come in while a federate is in one of these wait loops. If federates do not check for ExCO mode transitions in these wait loops, then a mode transition may be delayed or lost. Under some circumstances it could result in a deadlock condition. Therefore, a federate should always check for ExCO mode transitions in any wait loop.

Rule 7-18: Master Federate Can Only Mode Transition To Shutdown Prior To Initialization Complete

Requirement: The Master Federate shall not send an ExCO mode transition other than EXEC_MODE_SHUTDOWN prior to registration of “initialization_completed” synchronization point.

Rationale: Until initialization is complete, the federation execution is in a state of “initializing”. Since initialization is not complete, the federation execution cannot go to either run or freeze. The only other option is shutdown.

Rule 7-19: Master Federate Never Achieves “initialization_completed” Synchronization Point

Requirement: The Master Federate shall never achieve the “initialization_completed” synchronization point.

Rationale: The “initialization_completed” synchronization point is used as a marker synchronization point. It is intended to signal to all the federates that are joined, and will join, into a SpaceFOM-compliant federation execution that the Early Joiner initialization process is completed and that any Late Joiner federates can safely proceed with their initialization processes and join into the federation execution. The “initialization_completed” synchronization point will disappear if all federates achieve the synchronization point. However, as long as the Master Federate does not achieve the synchronization point it will remain even if other federates do achieve it.

Rule 7-20: Federates Do Not Achieve “mtr_shutdown” Synchronization Point

Requirement: No federate shall achieve the “mtr_shutdown” synchronization point.

Rationale: The “mtr_shutdown” synchronization point is used as a marker synchronization point. It is intended to signal to all the federates that are joined, and will join, into a SpaceFOM-compliant federation execution that the federation execution is being shut down. The “mtr_shutdown” synchronization point is used to prevent a possible race condition during shutdown; specifically, to prevent a Late Joiner federate from entering into a federation execution that is shutting down.

7.6.3. Mode Transition Rules

Rule 7-21: Only Master Federate Processes Mode Transition Requests.

Requirement: Only the Master Federate shall process MTR interactions.

Rationale: SpaceFOM-compliant federates participating in a SpaceFOM-compliant federation execution can issue requests to the Master Federate to change the execution mode of the federation execution. This is done with MTR interactions. Even though all federates could receive the MTR, only the Master Federate can command mode transitions using the ExCO; therefore, only the Master Federate can process an MTR.

Rule 7-22: All Federates Honor Mode Transitions.

Requirement: All federates shall honor ExCO mode transitions.

Rationale: Mode transitions are a key feature of a SpaceFOM-compliant federation execution. However, it requires that all participating federates comply with mode transitions sent out by the Master Federate in the ExCO.

Rule 7-23: Federates Start On Common HLA Logical Time Boundaries (HLTBs)

Requirement: Any late joining federate shall advance HLT to a common HLTB when first entering into a SpaceFOM-compliant federation execution.

Rationale: Time management and execution control requirements for SpaceFOM-compliant federation executions and the associated SpaceFOM-compliant federates require deterministically computable coordination points in the federation execution time line that can be achieved by all time constrained federates. Requiring SpaceFOM-compliant federates to start time advancement on HLTBs along with associated time step rules provides this.

The basic equation for computing the next common $HLTB$ is

$$HLTB = (floor(GALT/LCTS) + 1) * LCTS$$

where $GALT$ is the Greatest Available Logical Time.

Rule 7-24: Federation Execution Freezes On Common HLA Logical Time Boundaries.

Requirement: The freeze mode transition times computed by the Master Federate shall always be on a common HLTB in the future.

Rationale: Not all federates in a SpaceFOM-compliant federation execution have the same time step. In order for all federates in the federation execution to freeze on a natural time step for any federate, the freeze time will have to be on a common HLTB. This does not have to be the next available common HLTB but must be on a common HLTB in the future. Since the Master Federate sends out the ExCO with the Freeze mode transition, the Master Federate will determine the policy for computing the Freeze time at an HLTB in the future.

Rule 7-25: Document Master Federate Freeze Policy in FESFA.

Requirement: The policy used by the Master Federate to compute common HLTB freeze transitions shall be documented in the FESFA.

Rationale: It is important that all federates in a SpaceFOM-compliant federation execution understand the computational policy for computing freeze transition times (see Rule 7-24). There may be some latency in the delivery of freeze mode transitions and some federates may require numerous time steps to prepare to go to freeze. Therefore, the policy used to compute freeze mode transition times must be documented in the FESFA, implemented in the Master Federate, and understood by all participating federates.

7.7. Guidance

A SpaceFOM-compliant federation execution should only execute an HLA Save and/or Restore when the federation execution is in Freeze mode. Since all federation execution freezes occur on common HLTBs, all time managed federates will be in a time consistent state. In addition, executing the Save and/or Restore in freeze will avoid potential issues with advancing time during the Save or Restore operations.

8. General Agreements and Guidance

While the preceding chapters have defined much of the content of the SpaceFOM, there are still a few general topics that were not covered but are still prescriptive (normative) and descriptive (informative) to the SpaceFOM as a standard.

8.1. SpaceFOM Modules

The IEEE 1516-2010 HLA standard supports a modular model for FOM XML representations. Specifically, a FOM can be represented by a collection of FOM modules. This approach was introduced to provide a mechanism for extending and managing federation execution FOMs. The SpaceFOM makes use of this modular approach by defining 5 hierarchically related FOM modules: SISO_SpaceFOM_switches, SISO_SpaceFOM_datatypes, SISO_SpaceFOM_management, SISO_SpaceFOM_environment, and SISO_SpaceFOM_entity. The general relationship between these SpaceFOM modules is illustrated in Figure 8-1.

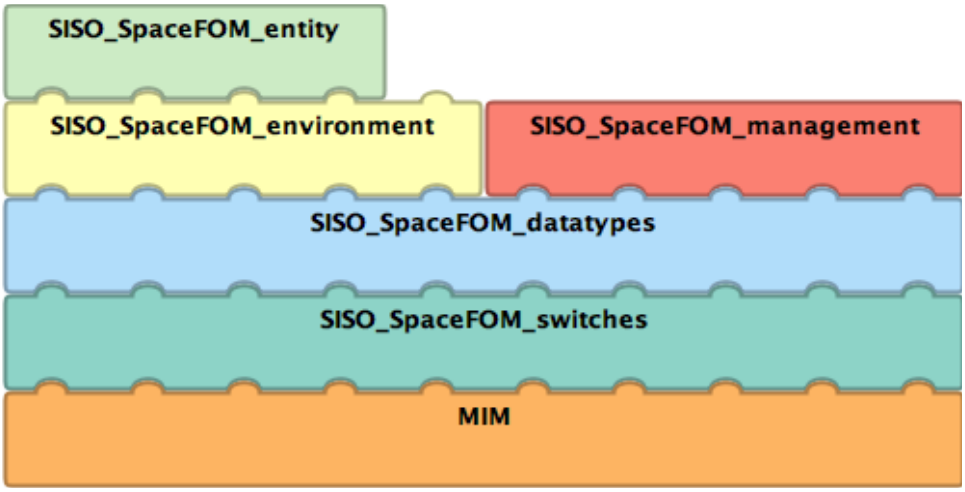

Figure 8-1: SpaceFOM Module Hierarchy

A brief description of the SpaceFOM modules and links to more detailed discussion is listed below:

- **SISO_SpaceFOM_switches**: Defines the switches table used in a SpaceFOM-compliant federation execution (see Section 8.10 and Appendix Section C.1).
- **SISO_SpaceFOM_datatypes**: Defines the common data types used by a SpaceFOM-compliant federate (see Section 3.3.1 and Appendix Section C.2).
- **SISO_SpaceFOM_management**: Defines the enumerations, object classes and interaction classes used to manage a SpaceFOM-compliant federation execution (see Section 7.5 and Appendix Section C.3).
- **SISO_SpaceFOM_environment**: Defines the environment object classes used as base data types for SpaceFOM-compliant federates (see Section 5.5 and Appendix Section C.4).
- **SISO_SpaceFOM_entity**: Defines the entity object classes used as base data types for SpaceFOM-compliant federates (see Section 6.3 and Appendix Section C.5).

The SpaceFOM modules along with their associated data types, enumerations, synchronization points, object classes, and interaction classes are discussed in detail in Appendix C.

8.2. Attribute Update Types

HLA provides a means for specifying the update type of each object attribute. The update type indicates the expectation of when an update of the attribute is provided by the publishing federate. For some update types, an additional update condition is specified to further identify specific conditions that result in an update. The SpaceFOM specifies an update type for all attributes and an update condition where appropriate. At a minimum, publishing federates shall update all required attributes and all provided optional attributes as specified by the update type and update condition. Publishing federates may provide updates more frequently, but this is not necessary.

8.3. Default Instance Attribute and Parameter Values

In many cases, federates may choose not to update attributes or send interaction parameters that have no meaning for that federate. The object and interaction class definitions provided in Appendix C of this document specify each attribute and parameter that can be treated in this manner. Subscribing federates shall assume default values for any attribute or parameter not provided by the updating/sending federate. Unless otherwise specified in the object or interaction class definition, default values shall be treated in the following manner:

- All integer and floating point numeric attributes and parameters default to zero;
- All Boolean attributes and parameters default to false;
- All enumerated attributes and parameters default to an enumerator indicating the information to remain unspecified, typically using the value 0, such as Other or No_Statement;
- All arrays and strings default to the empty string.

Attributes and parameters with data types other than those listed above do not have standard defaults. Modifications to the SpaceFOM should attempt to use the above default values whenever practical.

8.4. Synchronization Points

The SpaceFOM defines a collection of named synchronization points used for initialization control and execution mode transitions.

Name	SpaceFOM Module
initialization_started	C.3.4 SISO_SpaceFOM_management
initialization_completed	C.3.4 SISO_SpaceFOM_management
objects_discovered	C.3.4 SISO_SpaceFOM_management
root_frame_discovered	C.3.4 SISO_SpaceFOM_management
mtr_run	C.3.4 SISO_SpaceFOM_management
mtr_freeze	C.3.4 SISO_SpaceFOM_management
mtr_shutdown	C.3.4 SISO_SpaceFOM_management

Table 8-1: SpaceFOM Named Synchronization Points

As a general rule, as a default, all federates should achieve unrecognized synchronization points (see Rule 8-1).

8.5. Latency Compensation

Latency compensation is the process of adjusting a data value valid for a specified time, either with a time stamp or a time tag, to a different specified time. This can be a complex process with many options available to developers of a federate.[26] The possible need for latency compensation exists for any time stamped value but is explicitly an issue with any SpaceFOM object class instance that contains a time tag; which is any object class that contains a SpaceTimeCoodinate. This includes any object class based off of either the ReferenceFrame or PhysicalEnitity object classes.

In general, a SpaceFOM-compliant federate should be able to compensate for time offsets. Due to the time coherency requirements associated with the reference frame tree in a SpaceFOM-compliant federation execution, all SpaceFOM-compliant federates must be able to compensate a ReferenceFrame update to the federate's current SST.

8.6. Time Stamps and Time Tags

The SpaceFOM uses two fundamental representations of time primarily associated by time line: Time Stamp and Time Tag. The Time Stamp is used in conjunction with the HLT time line. While, the Time Tag is used in conjunction with the scenario time lines: FST and SST. These are discussed in detail in Section 4.3.2.

8.6.1. Time Stamps

A Time Stamp is defined as an instant in HLT associated with an event in the federation execution. In general, time stamps are the HLT values assigned to an HLA attribute update using the HLA time management interfaces. Latency compensation for federates which are not time-constrained require the transmission of time stamp information. In all cases, Time Tags are represented as HLAinteger64Time datatypes, which also correspond to HLAinteger64BE datatypes [3].

8.6.2. Time Tags

A Time Tag is defined as an instant in time associated with an event in the federation execution that corresponds to a known FST (a known SST for a specific federate). In general, time tags are used as the time values in SpaceFOM object class attributes for time. In all cases, Time Tags are represented as SpaceFOM Time data types (see Section 8.7.2.2).

8.7. Data Types and Encoding

The SpaceFOM modules described above define a collection of associated data types, enumerations, synchronization points, object classes, and interaction classes. In part, these types are defined to support the definition of the SpaceFOM object class hierarchy illustrated in Figure 8-2. However, the modules also proved a set of common data types and object classes used as base for extensibility and a priori interoperability.

[26] Latency compensation techniques are not discussed in this document but are covered in the broader HLA literature [32][33][34]

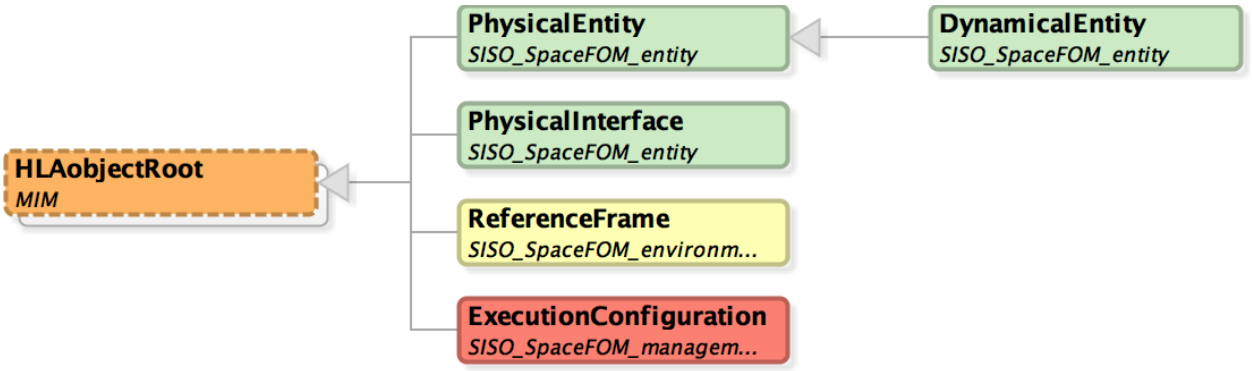

Figure 8-2: SpaceFOM Object Class Hierarchy

This section defines some of the details associated with the data types and encoding standards used by the SpaceFOM.

8.7.1. Data Type Naming Conventions

The SpaceFOM uses a Camel Case standard for naming data types. For example, the type name for a simple data type like mass rate is MassRate; an enumeration for execution mode state is ExecutionMode; an array type for a position vector is PostionVector. Enumerators for enumeration types are all caps with words separated by underscores. For example, the mode transition enumerator value for going to run mode is MTR_GOTO_RUN. Synchronization point names are all lower case separated by underscores. For example, the synchronization point for a mode transition request to go to run is “mtr_run”. This is applied across all data type specifications in all SpaceFOM modules.

8.7.2. SpaceFOM Specific Data Types

The SpaceFOM modules define a number of data types: basic HLA data representations, simple data types, enumerated data types, array data types, and fixed record data types. These are used throughout the SpaceFOM modules and provide a base for a priori interoperability with SpaceFOM-compliant federates.

8.7.2.1. Basic Data Representations

The SpaceFOM uses the following standard HLA data representations as underlying data representations for the SpaceFOM define datatypes, enumerations, object classes and interaction classes: HLAunicodeString, HLAinteger64Time, HLAinteger16LE, and HLAfloat64LE.

8.7.2.2. Simple Data Types

The SpaceFOM modules define the following simple data types:

Name	Representation	SpaceFOM Module
Angle	HLAfloat64LE	C.2.1 SISO_SpaceFOM_datatypes
Mass	HLAfloat64LE	C.2.1 SISO_SpaceFOM_datatypes
MassRate	HLAfloat64LE	C.2.1 SISO_SpaceFOM_datatypes
MassMomentOfInertia	HLAfloat64LE	C.2.1 SISO_SpaceFOM_datatypes
Length	HLAfloat64LE	C.2.1 SISO_SpaceFOM_datatypes
Velocity	HLAfloat64LE	C.2.1 SISO_SpaceFOM_datatypes
Acceleration	HLAfloat64LE	C.2.1 SISO_SpaceFOM_datatypes
Scalar	HLAfloat64LE	C.2.1 SISO_SpaceFOM_datatypes
AngularRate	HLAfloat64LE	C.2.1 SISO_SpaceFOM_datatypes

AngularAcceleration	HLAfloat64LE	C.2.1 SISO_SpaceFOM_datatypes
Time	HLAfloat64LE	C.2.1 SISO_SpaceFOM_datatypes
Energy	HLAfloat64LE	C.2.1 SISO_SpaceFOM_datatypes
Power	HLAfloat64LE	C.2.1 SISO_SpaceFOM_datatypes
SignalStrength	HLAfloat64LE	C.2.1 SISO_SpaceFOM_datatypes
Temperature	HLAfloat64LE	C.2.1 SISO_SpaceFOM_datatypes
TemperatureRate	HLAfloat64LE	C.2.1 SISO_SpaceFOM_datatypes
Force	HLAfloat64LE	C.2.1 SISO_SpaceFOM_datatypes
Torque	HLAfloat64LE	C.2.1 SISO_SpaceFOM_datatypes
Density	HLAfloat64LE	C.2.1 SISO_SpaceFOM_datatypes
MassMomentOfInertiaRate	HLAfloat64LE	C.2.1 SISO_SpaceFOM_datatypes

Table 8-2: SpaceFOM Simple Data Types

8.7.2.3. Enumerated Data Types

The SpaceFOM modules define the following enumerated data types:

Enumeration	SpaceFOM Module
ExecutionMode	C.3.3 SISO_SpaceFOM_management
MTRMode	C.3.3 SISO_SpaceFOM_management

Table 8-3: SpaceFOM Enumerated Data Types

8.7.2.4. Array Data Types

The SpaceFOM modules define the following array data types:

Name	SpaceFOM Module
PositionVector	C.2.2 SISO_SpaceFOM_datatypes
VelocityVector	C.2.2 SISO_SpaceFOM_datatypes
AccelerationVector	C.2.2 SISO_SpaceFOM_datatypes
AngularVelocityVector	C.2.2 SISO_SpaceFOM_datatypes
AngularAccelerationVector	C.2.2 SISO_SpaceFOM_datatypes
InertiaMatrix	C.2.2 SISO_SpaceFOM_datatypes
Vector	C.2.2 SISO_SpaceFOM_datatypes
Matrix	C.2.2 SISO_SpaceFOM_datatypes
ForceVector	C.2.2 SISO_SpaceFOM_datatypes
TorqueVector	C.2.2 SISO_SpaceFOM_datatypes
InertiaRateMatrix	C.2.2 SISO_SpaceFOM_datatypes

Table 8-4: SpaceFOM Array Data Types

8.7.2.5. Fixed Record Data Types

The SpaceFOM modules define the following fixed record data types:

Name	SpaceFOM Module
ReferenceFrameTranslation	C.2.3 SISO_SpaceFOM_datatypes
ReferenceFrameRotation	C.2.3 SISO_SpaceFOM_datatypes
AttitudeQuaternion	C.2.3 SISO_SpaceFOM_datatypes

SpaceTimeCoordinateState	C.2.3 SISO_SpaceFOM_datatypes

Table 8-5: SpaceFOM Fixed Record Data Types

8.7.2.6. Variant Record Data Types

The SpaceFOM does not currently define any variant record data types.

8.7.3. Endian Representation

To ensure interoperability among federates, federations have to agree on byte ordering conventions. The Little Endian byte order convention is used in the SpaceFOM with one notable exception. The standard representation for HLT values is HLAinteger64Time, which has an underlying HLAinteger64BE representation that is Big Endian. The motivation for choosing Little Endian is based on efficiency of encoding and decoding data for the majority of commodity microprocessor. The choice of Big Endian for the HLT values is solely based off of the HLA standard representation for integer time.

8.7.4. Word Alignment

The SpaceFOM uses the natural word alignment of the basic HLA datatypes and the standard HLA encodings for aggregate data types, object classes and object instances.

8.7.5. Basic Data Representations

The SpaceFOM utilizes a number of basic data representations that are defined by the HLA standards. Encoding Types. In IEEE 1516-2010, a datatype can be defined with an encoding. The encoding specifies how the datatype shall be encoded when provided to or received from the RTI.

8.8. Delivery Category

The HLA supports two different delivery categories – reliable and best effort. SpaceFOM federates are required to use reliable transportation for any data for which the delivery is managed using HLA Time Management. Data that is not HLA time managed may use best effort delivery if data loss is acceptable.

Atomic reflection of all attributes updated in a single RTI update is not guaranteed when using different delivery categories for different attributes in the update. That is, a set of attributes sent together might not be received together unless they use the same delivery category.

8.9. Switches

The SpaceFOM includes a FOM module named SISO_SpaceFOM_switches that contains the initial switches used by the RTI during a federation execution. This section describes their values in the FOM and during the Federation Execution.

The settings are as follows:

Switch	Value in SpaceFOM	Runtime Behavior
Auto-Provide	Disabled	Modified by Master Federate
Convey Region Designator Sets	Disabled	Always Disabled
Convey Producing Federate	Enabled	Always Enabled
Attribute Scope Advisory	Disabled	Always Disabled
Attribute Relevance Advisory	Disabled	Always Disabled
Object Class Relevance Advisory	Disabled	Always Disabled
Interaction Relevance Advisory	Disabled	Always Disabled
Service Reporting	Disabled	May be enabled if required
Exception Reporting	Disabled	May be enabled if required
Delay Subscription Evaluation	Disabled	Always Disabled
Automatic Resign Action	CancelThenDeleteThenDivest	Always CancelThenDeleteThenDivest

Table 8-6: SpaceFOM Compliant Switches Settings

8.9.1. Auto-Provide

This switch controls whether the RTI should automatically solicit updates from instance attribute owners when an object is discovered by a federate. The default setting for the Auto-Provide switch in the SpaceFOM is always Disabled. However, a federation execution can change the setting to Enabled. Regardless, the Master Federate will set the value of Auto-Provide to Disabled during initialization. Once initialization is complete, the Master Federate will restore the value of Auto-Provide to its original setting.

In order to support the default Auto-Provide setting in the RTI, all SpaceFOM-compliant federates should implement the Provide Attribute Value Update and Request Attribute Value Update service.

8.9.2. Convey Region Designator Sets

This switch controls whether the RTI should provide the optional Sent Region Set argument with invocations of Reflect Attribute Values and Receive Interaction. This switch shall be Disabled.

8.9.3. Convey Producing Federate

This switch controls whether the RTI should provide the optional Producing Federate with invocations of Discover Object Instances, Remove Object Instances, Reflect Attribute Values and Receive Interaction. This switch shall be set to Enabled.

8.9.4. Attribute Scope Advisory

This switch controls whether the RTI should advise federates, using a callback, when attributes of an object instance come into or go out of scope, i.e. any attribute updates, produced by the owning federate, will be delivered to this federate. This switch shall be Disabled.

8.9.5. Relevance Advisories

These switches controls whether federates are advised, using a callback, if any other federates will receive updates and interactions that the federate sends. These shall be Disabled.

Attribute Relevance Advisory: Whether the RTI should advise federates about whether they should provide attribute value updates for the value of an attribute of an object instance; the RTI bases this advisory on whether the value of the instance attribute is required by other federates.

Object Class Relevance Advisory: Whether the RTI should advise federates about whether they should register instances of an object class; the RTI bases this advisory on whether other federates have expressed an interest in attribute(s) of the object class.

Interaction Relevance Advisory: Whether the RTI should advise federates about whether they should send interactions of an interaction class; the RTI bases this advisory on whether other federates have expressed an interest in the interaction class.

8.9.6. Reporting Switches

These switches are used to report certain events. They are mainly used for debugging and performance analysis. Enabling these switches may result in extensive network traffic in the federation. They shall initially be set to Disabled. While they can temporarily be set to Enabled for debugging purposes, a federate shall not rely on them being Enabled. These switches are:

Service Reporting: Whether the RTI should report service invocations using MOM.

Exception Reporting: Whether the RTI should report exceptions using MOM.

8.9.7. Delay Subscription Evaluation Switch

This switch controls whether the RTI should filter messages as soon as possible to reflect the current known federate subscriptions, if disabled, or if enabled, defer subscription evaluation until only just before delivery to the federate. This switch shall be set to Disabled.

8.9.8. Automatic Resign Action Switch

This switch controls what resign action the RTI shall perform when it invokes the Resign Federation Execution service on behalf of a federate that was lost due to a fault. The value shall be CancelThenDeleteThenDivest.

8.10. The Switches FOM Module (SISO_SpaceFOM_switches)

The 1516-2010 HLA standard defined a set of switches that shall be set in the FOM. These switches regulate the behavior of some of the optional actions the RTI can perform on behalf of the federate, such as automatically requesting updates of an instance attribute when an object instance is discovered or advising the federates when certain events occur. To facilitate easy replacement of these settings, for the modular version of the HLA 1516-2010 SpaceFOM the switches have been confined to a single FOM module (SISO_SpaceFOM_switches). It is expected that federations might choose to update this module based on their federation agreement.

The SpaceFOM Switches Module is presented in Appendix Section C.1 along with the nominal values in Table 8-6.

8.11. Rules

Rule 8-1: Achieve Unknown Synchronization Points

Requirement: Federates shall achieve all unknown synchronization points.

Rationale: Synchronization points are used to implement key parts of the SpaceFOM execution control strategy. Some synchronization points need to be recognized by all SpaceFOM-compliant federates (e.g., initialization_started, initialization_completed, etc.). Other synchronization points are federation execution specific and will only be recognized by select federates. Regardless, all federates that are notified of a named synchronization point must achieve that synchronization point before the synchronization point can be cleared. Therefore, in order to unknowingly prevent a synchronization point from being cleared, a federate should achieve all unknown synchronization points.

Rule 8-2: Reference Frame Latency Compensation

Requirement: A SpaceFOM-compliant federate shall be able to compensate for latent reference frames states by propagating a reference frame state from a past time tag to the current federation scenario time.

Rationale: A reference frame tree is used to transform state representations from one reference frame to another. However, the reference frames in the tree need to have time homogeneous states to provide valid results. There are cases in which a federate may not have received all necessary reference frame states for the current time. In these cases, the federate needs to be able to propagate any latent reference frame states from their published time tag to the current time. This establishes a time consistent and useful reference frame tree.

Rule 8-3: Use of Nominal Switches Table Settings

Requirement: A SpaceFOM-compliant federation execution shall use the standard SpaceFOM Switches Table settings unless specifically stated otherwise in the FESFA.

Rationale: The Switches Table regulates the behavior of the RTI. All federates in a federation execution need to understand the expected behavior of the RTI. If a federation execution requires some specialized behavior from the RTI that differs from the nominal setting in the SpaceFOM Switched Table, this needs to be agreed upon in advance and documented in the FESFA.

Rule 8-4: Auto-Provide Disabled By Default

Requirement: The default Auto-Provide switches table value shall be Disabled but can be overridden.

Rationale: By setting the default switches table Auto-Provide value to Disabled, the behavior of the federation is well defined. While having Auto-Provide set to Enabled can be a convenience for some federates, it can also result in non-deterministic behavior and cause performance problems.

Rule 8-5: Document Auto-Provide Setting In FESFA

Requirement: The value of Auto-Provide in the switches table shall be documented in the FESFA, if the default SpaceFOM Auto-Provide setting is overridden.

Rationale: The behavior or the RTI changes significantly when Auto-Provide is Enabled. All potential federates wishing to participate in a federation execution will be able to see the Auto-Provide setting in the FESFA.

Rule 8-6: Federates Implement Provide Attribute Value Update Service Interface

Requirement: All federate shall implement the HLA Provide Attribute Value Update service interface.

Rationale: The default switches table Auto-Provide value is Disabled. To support a priori interoperability, a SpaceFOM-compliant federate needs to implement the HLA Provide Attribute Value Update service interface. This permits other federates to use the Request Attribute Value Update service to request updates when needed.

Rule 8-7: Late Joiner Federates Request Needed Attribute Updates

Requirement: All Late Joiner federates in a SpaceFOM-compliant federation execution shall explicitly request updates to needed attributes.

Rationale: The default switches table Auto-Provide value is Disabled. To support a priori interoperability, a SpaceFOM-compliant federate needs to explicitly request updates to needed attributes using the HLA Request Attribute Value Update service. This ensures that a late joining federate will get updates to needed attribute values.

8.12. Guidance

8.12.1. Synchronization Points

Federates that register special specific synchronization points should restrict the associated synchronization sets to include only those peer federates taking part in the specific synchronization.

Bibliography

This bibliography provides a listing of applicable Simulation Interoperability Standards Organization (SISO) reference material and a listing of general references cited throughout this document. The following references contain material that must be understood and used to implement the product.

Applicable SISO References

[1] Simulation Interoperability Standard Organization (SISO), "IEEE Standard for Modeling and Simulation (M&S) High Level Architecture (HLA) – Framework and Rules", IEEE Computer Society, New York, New York, IEEE Std 1516™-2010, 18 August 2010.

[2] Simulation Interoperability Standard Organization (SISO), "IEEE Standard for Modeling and Simulation (M&S) High Level Architecture (HLA) – Federate Interface Specification", IEEE Computer Society, New York, New York, IEEE Std™ 1516.1™-2010, 18 August 2010.

[3] Simulation Interoperability Standard Organization (SISO), "IEEE Standard for Modeling and Simulation (M&S) High Level Architecture (HLA) – Object Model Template (OMT) Specification", IEEE Computer Society, New York, New York, IEEE Std™ 1516.2™-2010, 18 August 2010.

[4] Simulation Interoperability Standard Organization (SISO), "Real-time Platform Reference Federation Object Model (RPR FOM)", SISO, Orlando, FL, SISO-STD-001.1-2015, Version 2.0, 10 August 2015.

[5] Simulation Interoperability Standard Organization, "Policies & Procedures", SISO, Orlando, FL, SISO-ADM-002-2017, 19 June 2017.

[6] Simulation Interoperability Standard Organization, "Policy for the Style and Format of SISO Documents", SISO, Orlando, FL, SISO-ADM-005-2011 13 June 2011.

[7] Simulation Interoperability Standard Organization (SISO), "Reference FOM Study Group - Final Report", SISO, Orlando, FL, SISO- REF-001-1998, Version 1.0, 9 March 1998.

[8] Simulation Interoperability Standard Organization (SISO), "Reference for Enumerations for Simulation Interoperability", SISO, Orlando, FL, SISO-REF-010-2016, Version 22, 10 May 2016.

[9] Simulation Interoperability Standard Organization (SISO), "IEEE Recommended Practice for Distributed Simulation Engineering and Execution Process (DSEEP)", IEEE Computer Society, New York, New York, IEEE Std™ 1730™-2010, 24 January 2011.

General References

[10] Simulation Exploration Experience (SEE), "SEE – Simulation Exploration Experience", Available at: http://www.exploresim.com , Web Page, Accessed: 16 July 2018.

[11] General Mission Analysis Tool (GMAT), "GMAT Wiki Home", Available at: http://gmatcentral.org , Web Page, Accessed: 16 July 2018.

[12] General Mission Analysis Tool (GMAT), "GMAT: CoordinateSystem Resources", Available at: http://gmat.sourceforge.net/docs/R2018a/html/CoordinateSystem.html , Web Page, Accessed: 16 July 2018.

[13] International Astronomical Union (IAU), "Naming of Astronomical Objects", Available at: https://www.iau.org/public/themes/naming/ , Web Page, Accessed: 16 July 2018.

[14] LoveToKnow Corporation, "Webster's New World College Dictionary", Available at http://websters.yourdictionary.com, Web Page, LoveToKnow Corporation, Burlingame, CA, Accessed: 12 March 2018.

[15] US Department of Defense, "Modeling and Simulation (M&S) Glossary", Available: https://www.msco.mil/MSReferences/Glossary/MSGlossary.aspx , Web Page, Accessed: 10 July 2017.

[16] Institute of Electrical and Electronics Engineers (IEEE), "IEEE Standards Dictionary Online", Available: https://www.ieee.org/publications_standards/publications/subscriptions/prod/standards_dictionary.html , Web Page, Accessed: 10 July 2017.

[17] World Wide Web Consortium (W3C), "Extensible Markup Language (XML)", Available: https://www.w3.org/XML/ , Web Page, Accessed: 14 July 2017.

[18] National Institute of Standards and Technology (NIST), "the NIST Reference on Constants, Units, and Uncertainty", Available: https://physics.nist.gov/cuu/Units/units.html , Web Page, Accessed: 13 July 2017.

[19] The Open Group, "The Open Group Base Specifications Issue 7", IEEE Std 1003.1-2008, 2016 Edition, Available: http://pubs.opengroup.org/onlinepubs/9699919799/basedefs/V1_chap04.html#tag_04_16 , Web Page, Accessed: 25 July 2017.

[20] J. Towers, J. Hines, "Highly Dynamic Vehicles in a Real/Simulated Virtual Environment (HyDy), Equations of Motion of the DIS 2.0.3 Dead Reckoning Algorithms," Advanced Research Projects Agency, February 7, 1994.

[21] G. Kaplan, "The IAU Resolutions on Astronomical Reference Systems, Time Scales, and Earth Rotation Models: Explanation and Implementation", United States Naval Observatory, Circular No. 179, October 20, 2005.

[22] N. Capitaine, "The Astronomical Reference Systems in the Framework of General Relativity", Observatorie de Paris / SYRTE, IAP 2010, Presentation.

[23] Luzum, et al., "The IAU 2009 system of astronomical constants: the report of the IAU working group on numerical standards for Fundamental Astronomy", Celestial Mechanics and Dynamical Astronomy, Special Report, August 2011, Volume 110, Issue 4, pp 293-304.

[24] International Astronomical Union (IAU), "IAU Division 1 Working Group Numerical Standards for Fundamental Astronomy IAU 2009 System of Astronomical Constants", Available: http://maia.usno.navy.mil/NSFA/IAU2009_consts.html , Web Page, Accessed: 10 July 2017.

[25] National Aeronautics and Space Administration (NASA), "Space Station Reference Coordinate Systems", SSP 30219 Revision J, International Space Station Program, NASA Johnson Space Center, Houston, Texas, 1 May 2008.

[26] National Aeronautics and Space Administration (NASA), "Constellation Program Level II Coordinate Systems", CxP 70138 Revision B, NASA Johnson Space Center, Houston, Texas, 2 December 2009.

[27] P. Heafner, "Fundamental Ephemeris Computations: For Use With JPL Data", Willmann-Bell, Inc., 1999.

[28] P. Sidelmann, "Explanatory Supplement to the Astronomical Almanac", U.S. Naval Observatory, Washington D.C., University Science Books, 2006.

[29] J. Kuipers, "Quaternions and Rotation Sequences", Princeton University Press, 2002.

[30] D. Eberly, "Rotation Representations and Performance Issues", http://www.geometictools.com , March 1, 2008.

[31] J.M.P. van Waveren, "From Quaternion to Matrix and Back", Id Software, Inc., 27 February 2005.

[32] G. Lauderdale, D. Snyder, E. Crues, D. Hasan, "Further Studies On The Feasibility Of A Distributed ISS and HTV Simulation", 2003 Fall Simulation Interoperability Workshop, Orlando, FL, September 14-19, 2003, 03F-SIW-010.

[33] R. Phillips and E. Crues, "Time Management Issues and Approaches for Real Time HLA Based Simulations", 2005 Fall Simulation Interoperability Workshop, Orlando, FL, September 18-23, 2005, 05F-SIW-058.

[34] R. Phillips, “An Analysis of Constraints on Real-Time Distributed Simulations”, 2005 Fall Simulation Interoperability Workshop, Orlando, FL, September 16-21, 2007, 07F-SIW-007.

Appendices

Appendix A. [Normative] Template for Federation Execution Specific Federation Agreement

The Federation Execution Specific Federation Agreement (FESFA) is a document that provides specific configuration data necessary to achieve interoperability based on the SpaceFOM. Several rules in the SpaceFOM put requirements on what data need to be recorded in the FESFA.

This appendix provides a template for a SpaceFOM-compliant FESFA. Versions of this template can be found on the SISO standards web site associated with the SpaceFOM. Alternately, a version of the template can be extracted from below.

The information that appears in red italicized text is provided as guidance and would not be visible in the template when viewed in Print Preview and/or when the document is printed. However, the text is visible here for instructional purposes. If this appendix is used to create a SpaceFOM-compliant FESFA, the author will want to "hide" or delete the red italicized informational text. The author may also choose to add or remove page breaks for readability.

Space Reference Federation Object Model (SpaceFOM)
Federation Execution Specific Federation Agreement (FESFA)
for the
<Federation Execution Title>

The information that appears in red italicized text is hidden text and is not seen when viewed in Print Preview and/or when the document is printed. The template is a Word Document and edited like any other Word Document.

The Space Reference Federation Object Model (SpaceFOM) Federation Execution Specific Federation Agreement (FESFA) is a document that provides specific configuration data necessary to achieve interoperability based on the SpaceFOM. Several rules in the SpaceFOM put requirements on what data need to be recorded in the FESFA. This template establishes the standard format and content so that all SpaceFOM FESFA products contain the same basic information and have the same basic look.

Purpose

This section of the FESFA template will provide the general purpose and description of this specific SpaceFOM-compliant federation execution. This should include intended scenarios and other information that describes the nature of the federates participating in a federation execution compliant with this FESFA.

Identification

This section of the FESFA template provides the general identifying information associates with the federation execution.

General name identifying the federation execution, this should match the <Federation Execution Title> in the title above but not necessarily the "HLA Federation Execution Name" below.

Federation Execution Title: _____

Information pertaining to the principal point of contact for this FESFA.

Point of Contact:

Name: _____

Phone: _____

Email: _____

Address: _____

Real world time frame (calendar dates) for proposed federation executions, not to be confused with federation execution scenario dates.

Planned Execution Time Frame: From: _____ To: _____

HLA federation execution name, not necessarily the identification name from above.

HLA Federation Execution Name: _____

Federation Composition

This section of the FESFA template provides the identifying information associates with the composition of the federation execution.

Information on the federate providing the role of the Master Federate for this federation execution.

Master Federate: _____

Information on the federate providing the role of the Pacing Federate for this federation execution.

Pacing Federate: _____

Information on the federate providing the role of the Root Reference Frame Publisher (RRFP) federate for this federation execution.

Root Reference Frame Publisher (RRFP): _____

List the names and descriptions of any additional required federates for this federation execution. Add additional lines as needed.

Additional required federates:

Name	Description

Time Management

This section of the FESFA template provides the general time management information associates with the federation execution. All participating time managed federates will need this information. Some of this information is published by the Master Federate in the Execution Control Object (ExCO).

The starting date for the federation execution federation scenario time (FST_0). This can be given as a calendar date and time but will ultimately have to be converted into the Terrestrial Time (TT) scale in Truncated Julian Date (TJD) format.

Epoch: _____ (TT scale in TJD format)

The federation execution's nominal HLA Logical Time (HLT) step in microseconds.

Federation HLT step: _____ (microseconds)

The federation execution's Least Common Time Step (LCTS) in microseconds. This is the least common value of all the federate time steps for the time regulating federates in a federation execution.

Federation LCTS: _____ (microseconds)

Identify the supported time management type for this federation execution.

Supported Time Management Types:

- No Pacing: _____ (yes/no)
- Scaled Pacing: _____ (yes/no)
- Real-time Pacing with Unlimited Overruns: _____ (yes/no)
- Real-time Pacing with Limited Overruns: _____ (yes/no)
- Strict/Conservative Real-time Pacing: _____ (yes/no)

If any of the real-time pacing options are supported, then include a section that describes limitations and how those overruns are handled.

Overrun handling: _____

Indication of the existence of Central Timing Equipment (CTE) to control the federation execution time advance for real-time hardware-in-the-loop (HwITL) simulations. This is a yes or no question.

CTE federates exists: _____ (yes/no)

References to any document(s) that describes the implementation and configuration details for any CTE. Add additional document references as necessary. Just mark (N/A) if no CTE is used.

CTE specification document(s):

1. <CTE reference document 1.>
2. <CTE reference document 2.>

Reference Frames

This section of the FESFA template provides the names and descriptions of the principal reference frames published during a federation execution. It should be sufficient to understand the principal topology of the federation execution's reference frame tree.

The name and brief description of the reference frame that represents the common base (root) reference frame for the federation execution's reference frame tree.

Root Reference Frame:

Name	Description
_____	_____

The name, parent, and brief description of any additional reference frame that play an important role in the federation execution's reference frame tree. Any reference frames published by or subscribed to by required federates should be listed here. This list should represent the union of all reference frames listed in the FCDs of the required federates. It may also include reference frames of other potential federates. Add additional lines as necessary.

Additional Reference Frames:

Name	Parent	Description

Object Management

This section of the FESFA template provides the general object management information associated with the federation execution. Most participating federates will need this information.

List the type strings associated with any PhysicalEntity object's "type" attribute used in this federation execution. List both the string (tag) values and a description of each tag.

Physical Entity Object Type Strings:

Type String (Tag)	Description

List the status strings associated with any PhysicalEntity object's "status" attribute used in this federation execution. List both the string (tag) values and a description of each tag.

Physical Entity Object Status Strings:

Status String (Tag)	Description

A brief description of the canonical naming convention used to distinguish PhysicalInterface object instances from one another. Add additional lines as necessary.

Physical Interface Instance Naming Convention: _____

The name, type, and brief description of any key object instances that play an important role in the federation execution. Add additional lines as necessary.

Key Object Instances:

Name	Object Class	Description

List the name and description of any additional FOM modules need by this federation execution. This should be the union of all FOM modules listed in the FCDs of the required federates. It may also include other FOM modules of other potential federates. Add additional lines as necessary.

Additional FOM Modules:

FOM Module Name	Description

Initialization

This section of the FESFA template provides the information associates with the initialization policy and approach use in the federation execution. It specifically focuses on the details of any multiphase initialization. All Early Joiner federates participating in the multiphase initialization process will need this information.

Indication for the use of multiphase initialization (MPI). This is a yes or no question.

MPI Used: _____ (yes/no)

The MPI specification can consist of an inline description of the MPI approach. Alternately, list references to any document(s) that describes the implementation and configuration details for any MPI used in the startup of the federation execution. Add additional document references as necessary. Just mark (N/A) if no MPI is used.

MPI Specification:

1. <MPI reference document 1.>
2. <MPI reference document 2.>

Additional Technical Information

This section of the FESFA template provides any additional technical information needed by federates participating in the federation execution. This section may be marked (N/A) or omitted if there is no additional technical information.

Specify any non-standard switches settings required to configure the RTI for this federation execution. For instance, this is where the behavior of the Auto-Provide switch would be documented if enabled. Add additional lines as necessary. Just mark (N/A) or omit if none.

Non-standard Switches Settings:

Switch	Value	Description

List or describe any additional data sources and/or databases required to support this federation execution. Add additional lines as necessary. Just mark (N/A) or omit if no additional data sources are needed.

Additional Common Data and/or Databases:

Data Source	Data Description

References to any additional technical document. Add additional document references as necessary. Just mark (N/A) or omit if none.

Additional Technical Documents:

1. <Technical reference document 1.>
2. <Technical reference document 2.>

Appendix B. [Normative] Template for Federate Compliance Declaration

The Federate Compliance Declaration (FCD) is a document that provides specific configuration data necessary to achieve interoperability based on the SpaceFOM. Several rules in the SpaceFOM put requirements on what data need to be recorded in the FCD. In general, The FCD describes which capabilities a federate has and which roles it can play in a SpaceFOM-compliant federation execution.

This appendix provides a template for a SpaceFOM-compliant FCD. Versions of this template can be found on the SISO standards web site associated with the SpaceFOM. Alternately, a version of the template can be extracted from below.

The information that appears in red italicized text is provided as guidance and would not be visible in the template when viewed in Print Preview and/or when the document is printed. However, the text is visible here for instructional purposes. If this appendix is used to create a SpaceFOM-compliant FCD, the author will want to "hide" or delete the red italicized informational text. The author may also choose to add or remove page breaks for readability.

Space Reference Federation Object Model (SpaceFOM)
Federate Compliance Declaration (FCD)
for the
<Federate Name>

The information that appears in red italicized text is hidden text and is not seen when viewed in Print Preview and/or when the document is printed. The template is a Word Document and edited like any other Word Document.

The Space Reference Federation Object Model (SpaceFOM) Federate Compliance Declaration (FCD) is a document that provides specific configuration data necessary to achieve interoperability based on the SpaceFOM. Several rules in the SpaceFOM put requirements on what data need to be recorded in the FCD. In general, The FCD describes which capabilities a federate has and which roles it can play in a SpaceFOM-compliant federation execution. This template establishes the standard format and content so that all SpaceFOM FCD products contain the same basic information and have the same basic look.

Purpose

This section of the FCD template will provide the general purpose and description of this specific SpaceFOM-compliant federate. This should include intended scenarios and other information that describes the nature of the federate's capabilities and compliance as a SpaceFOM-compliant federate.

Federate providers should provide a federate compliance declarations to facilitate the assessment of the suitability of a federate in a specific federation execution.

Identification

This section of the FCD template provides the general identifying information associates with this federate.

General name identifying this federate, this should match the <Federate Name> in the title above but not necessarily the name used when the federate joins an HLA federation execution (see "HLA Federation Execution Join Name" below).

Name: _____

Specify the federate version identification.

Version: _____

Information pertaining to the principal point of contact for this FCD.

Point of Contact:

Name: _____

Phone: _____

Email: _____

Address: _____

The HLA name used by this federate when joining an HLA federation execution, not necessarily the identification name from above. Specify the name here if it is fixed or indicate the means for setting if it can be configured at runtime.

HLA Federation Execution Join Name: _____

SpaceFOM Federate Roles Supported

This section of the FCD template provides information on the SpaceFOM roles that this federate can support.

Statement that this federate can fulfill the role of the Master Federate in a SpaceFOM-compliant federation execution. This is a yes or no question.

Can act as Master Federate: _____ (yes/no)

Statement that this federate can fulfill the role of the Pacing Federate in a SpaceFOM-compliant federation execution. This is a yes or no question.

Can act as Pacing Federate: _____ (yes/no)

Statement that this federate can fulfill the role of the Root Reference Frame Publisher (RRFP) federate in a SpaceFOM-compliant federation execution. This is a yes or no question.

Can act as Root Reference Frame Publisher: _____ (yes/no)

Time Management

This section of the FCD template provides the general time management information associates with this federate.

Specify the earliest and latest valid operating Simulation Scenario Time (SST) dates and times for this federate. This can be given as a calendar date and time but will ultimately have to be converted into the Terrestrial Time (TT) scale in Truncated Julian Date (TJD) format.

Valid Operating Time Frame:

Earliest: _____ (TT scale in TJD format)

Latest: _____ (TT scale in TJD format)

Specify the minimum, nominal, and maximum HLA Logical Time (HLT) step in microseconds supported by this federate. These time step capabilities will inform the Least Common Time Step (LCTS) calculation for any federation execution that this federate joins.

Time Step Support:

Minimum: _____ (microseconds)

Nominal: _____ (microseconds)

Maximum: _____ (microseconds)

Indication that this federate supports being an Early Joiner federate. This is a yes or no question.

Supports early joining: _____ (yes/no)

Indication that this federate supports being a Late Joiner federate. This is a yes or no question.

Supports late joining: _____ (yes/no)

Specify if this federate can or should be a time regulating federate.

Time Regulating: _____ (required/optional/no)

Specify if this federate can or should be a time constrained federate.

Time Constrained: _____ (required/optional/no)

Identify the supported time management type for this federate. These are yes or no questions.

Supported Time Management Types:

No Pacing: _____ (yes/no)

Scaled Pacing: _____ (yes/no)

Real-time Pacing with Unlimited Overruns: _____ (yes/no)

Real-time Pacing with Limited Overruns: _____ (yes/no)

Strict/Conservative Real-time Pacing: _____ (yes/no)

If any of the real-time pacing options are supported, then include a section that describes limitations and how those overruns are handled.

Overrun handling: _____

Indication that this federate requires support for Central Timing Equipment (CTE) to control its time advance. This is a yes or no question.

Requires CTE: _____ (yes/no)

References to any document(s) that describes the implementation and configuration details for any CTE. Add additional document references as necessary. Just mark (N/A) if CTE is not required.

CTE specification document(s):

1. <CTE reference document 1.>
2. <CTE reference document 2.>

Reference Frames

This section of the FCD template provides the names and descriptions of the principal reference frames published by or required by this federate.

If this federate can publish a root reference frame, then this is the name and brief description of the reference frame that represents the common base (root) reference frame for a SpaceFOM-compliant reference frame tree. If this federate cannot publish a root reference frame, then this entry should be omitted.

Root Reference Frame:

Name	Description

The name, parent, and brief description of any reference frames published by this federate. Add additional lines as necessary.

Published Reference Frames:

Name	Parent	Description

The name, parent, and brief description of any reference frames required by this federate. Add additional lines as necessary.

Required Reference Frames:

Name	Parent	Description

Object Management

This section of the FCD template provides the general object management information associated with the federate. The information in this section will inform the overall object management strategy for a federation execution in which this federate participates.

List the type strings associated with any PhysicalEntity object's "type" attribute used in this federate. List both the string (tag) values and a description of each tag.

Physical Entity Object Type Strings:

Type String (Tag)	Description

List the status strings associated with any PhysicalEntity object's "status" attribute used in this federate. List both the string (tag) values and a description of each tag.

Physical Entity Object Status Strings:

Status String (Tag)	Description

A brief description of the canonical naming convention used to distinguish PhysicalInterface object instances from one another. Add additional lines as necessary.

Physical Interface Instance Naming Convention: _____

_____

List the name and brief description of all PhysicalInterface instances. Add additional lines as necessary.

Physical Interface Instances:

Interface Instance Name	Description

List the name, type, and brief description of any published object instances. Add additional lines as necessary.

Published Object Instances:

Name	Type	Description

List the name, type, and brief description of any required object instances. Add additional lines as necessary.

Required Object Instances:

Name	Type	Description

List the name and description of any additional FOM modules need by this federate. Add additional lines as necessary.

Additional FOM Modules:

FOM Module Name	Description

Initialization

This section of the FCD template provides the information associates with the initialization policy and approach use by this federate. It specifically focuses on the details of its multiphase initialization process. The information in this section will inform the overall MPI strategy for a federation execution in which this federate participates.

Indication for the use of multiphase initialization (MPI). This is a yes or no question.

MPI Used: _____ (yes/no)

The MPI specification can consist of an inline description of the MPI approach. Alternately, list references to any document(s) that describes the implementation and configuration details for any MPI used in the startup of the federation execution. Add additional document references as necessary. Just mark (N/A) if no MPI is used.

MPI Specification:

1. <MPI reference document 1.>
2. <MPI reference document 2.>

Additional Technical Information

This section of the FCD template provides any additional technical information needed by this federate. This section may be marked (N/A) or omitted if there is no additional technical information.

List or describe any additional data sources and/or databases required to support this federate. Add additional lines as necessary. Just mark (N/A) or omit if no additional data sources are needed.

Additional Data and/or Databases:

Data Source	Data Description

References to any additional technical document. Add additional document references as necessary. Just mark (N/A) or omit if none.

Additional Technical Documents:

1. <Technical reference document 1.>
2. <Technical reference document 2.>

Compliance Statement

This is the general acknowledgement that this federate complies with the Space Reference FOM standard. This is a yes or no question.

This federate fulfils all relevant requirements in the SpaceFOM version 1.0: _____ (yes/no)

If the answer to the compliance question above is "no", then an explanatory note can be provided here.

Appendix C. [Informative] SpaceFOM Modules

The following sections provide details on the switches, data types, synchronization points, object classes, and interaction classes that make up the SpaceFOM modules. The information in this section is extracted from the SpaceFOM modules. While this section is informative, the source SpaceFOM XML files are normative.

C.1. The Switches FOM Module (SISO_SpaceFOM_switches)

The 1516-2010 HLA standard defined a set of switches that shall be set in the FOM. These switches regulate the behavior of some of the optional actions the RTI can perform on behalf of the federate, such as automatically requesting updates of an instance attribute when an object instance is discovered or advising the federates when certain events occur. To facilitate easy replacement of these settings, for the modular version of the HLA 1516-2010 SpaceFOM the switches have been confined to a single FOM module (SISO_SpaceFOM_switches). It is expected that federations might choose to update this module based on their federation agreement. More detail on these settings can be found in Section 8.9.

SpaceFOM Module Dependencies: <None>.

Switch	Setting
Auto-Provide	Disabled
Convey Region Designator Sets	Disabled
Convey Producing Federate	Disabled
Attribute Scope Advisory	Disabled
Attribute Relevance Advisory	Disabled
Object Class Relevance Advisory	Disabled
Interaction Relevance Advisory	Disabled
Service Reporting	Disabled
Exception Reporting	Disabled
Delay Subscription Evaluation	Disabled
Automatic Resign Action	CancelThenDeleteThenDivest

Table C-1: Standard SpaceFOM Switches Table

C.2. The Data Types FOM Module (SISO_SpaceFOM_datatypes)

This SpaceFOM module provides a number of essential data types. These are used for Object Attributes as well as Interaction Parameters. The data types use the International System of Units (SI) [18] wherever possible.

SpaceFOM Module Dependencies: SISO_SpaceFOM_switches.

C.2.1. Simple Data Types

For all data types defined in Table C-2, the underlying data representation is HLAfloat64LE.

Name	Units	Semantics
Angle	radian (r)	A scalar angular counterclockwise quantity.
Mass	kilogram (kg)	A measurement of mass.
MassRate	kilogram per second (kg/s)	A measurement of the rate of change of mass.
MassMomentOfInertia	kilogram meter squared (kg*m^2)	A scalar moment or coefficient of inertia. There are nine such scalars in a moment of inertia matrix.

Name	Units	Semantics
Length	meter (m)	A scalar length.
Velocity	meter per second (m/s)	A scalar translational velocity.
Acceleration	meter per second squared (m/s^2)	A scalar translational acceleration.
Scalar	NA	A unitless scalar value.
AngularRate	radian per second (r/s)	A scalar angular rate.
AngularAcceleration	radian per second squared (r/s^2)	A scalar angular acceleration.
Time	second (s)	A measurement of time.
Energy	Joule (J)	A measure of energy.
Power	Watt (W)	A measure of power.
SignalStrength	Decibel (dB)	A measure of signal strength.
Temperature	Kelvin (K)	A measure of absolute temperature.
TemperatureRate	Kelvin per second (K/s)	A measure of the time rate of change of temperature.
Force	Newton (N)	A scalar measurement of force.
Torque	Newton meter (N*m)	A scalar measurement of torque.
Density	kilograms per cubic meter (kg/m^3)	A measure of mass density.
MassMomentOfInertiaRate	kilogram meter squared per second (kg*m^2/s)	A measure of the time rate of change of a mass moment of inertia parameter.

Table C-2: SpaceFOM Simple Data Types

C.2.2. Array Data Types

Name	Element Data Type	Cardinality	Encoding	Semantics
PositionVector	Length	3	HLAfixedArray	A 3-vector that specifies the translational position of one point with respect to another. This data type does not specify which points are involved, nor does it specify the coordinate axes onto which the three components of the vector are projected.

Name	Element Data Type	Cardinality	Encoding	Semantics
VelocityVector	Velocity	3	HLAfixedArray	A 3-vector that specifies the time derivative of the vector position of some point with respect to another as seen by an observer fixed in some reference frame. This data type does not specify which points are involved, nor does it specify the observer frame of reference, nor does it specify the coordinate axes onto which the three components of the vector are projected.
AccelerationVector	Acceleration	3	HLAfixedArray	A 3-vector that specifies the time derivative of a vector velocity of some point with respect to another as seen by an observer fixed in some reference frame. This data type does not specify which points are involved, nor does it specify the observer frame of reference, nor does it specify the coordinate axes onto which the three components of the vector are projected.
AngularVelocityVector	AngularRate	3	HLAfixedArray	A 3-vector that specifies the time derivative of the orientation of one reference frame with respect to another. This data type does not specify the reference frames, nor does it specify the coordinate axes onto which the three components of the vector are projected.
AngularAccelerationVector	AngularAcceleration	3	HLAfixedArray	A 3-vector that specifies the time derivative of an angular velocity vector as seen by an observer fixed in some reference frame. This data type does not specify which angular velocity, nor does it specify the observer frame of reference, nor does it specify the coordinate axes onto which the three components of the vector are projected.

Name	Element Data Type	Cardinality	Encoding	Semantics
InertiaMatrix	MassMomentOfInertia	9	HLAfixedArray	A 3x3 matrix that specifies the mass inertia matrix of a body about some coordinate axes. The nine elements of the matrix are stored row-wise, namely: Ixx, Ixy, Ixz, Iyx, Iyy, Iyz, Izx, Izy, Izz. The off-diagonal components Iij (i!=j) are the so-called 'negative integrals'. This means that the elements in this inertia matrix satisfy the equation H=Iw. Where H is the angular momentum vector, I is the inertia matrix and w is the angular velocity vector. This data type does not specify the coordinate axes about which the moments are calculated.
Vector	Scalar	3	HLAfixedArray	A unitless 3-vector.
Matrix	Scalar	9	HLAfixedArray	A unitless 3x3 matrix. The nine elements of the matrix are stored row-wise, namely: m11, m12, m13, m21, m22, m23, m31, m32, m33, where the first index is the row index.
ForceVector	Force	3	HLAfixedArray	A 3-vector that specifies the vector force. This data type does not specify which points are involved, nor does it specify the coordinate axes onto which the three components of the vector are projected.
TorqueVector	Torque	3	HLAfixedArray	A 3-vector that specifies the vector torque. This data type does not specify which points are involved, nor does it specify the coordinate axes onto which the three components of the vector are projected.
InertiaRateMatrix	MassMomentOfInertiaRate	9	HLAfixedArray	A 3x3 matrix that specifies the time rate of change of the parameters in the InertiaMatrix. The elements in this matrix correspond directly to the elements in the InteriaMatrix.

Table C-3: SpaceFOM Array Data Types

C.2.3. Fixed Record Data Types

ReferenceFrameTranslation

This is the translational state of a subject reference frame with respect to a 'referent' frame. This data type does not specify the two reference frames.

Encoding: HLAfixedRecord

Name	Data Type	Semantics
position	PositionVector	Position of the subject frame origin with respect to the referent origin with components expressed in the referent coordinate axes.
velocity	VelocityVector	Velocity of the subject frame origin with respect to its referent origin with components expressed in the referent coordinate axes.

Table C-4: SpaceFOM ReferenceFrameTranslation Type

ReferenceFrameRotation

This is the rotational state of a reference frame with respect to a 'referent' frame.

Encoding: HLAfixedRecord

Name	Data Type	Semantics
attitude_quaternion	AttitudeQuaternion	Attitude quaternion that specifies the orientation of the subject frame with respect to the referent.
angular_velocity	AngularVelocityVector	Angular velocity of the subject frame with respect to the referent with components resolved onto the subject coordinate axes.

Table C-5: SpaceFOM ReferenceFrameRotation Type

AttitudeQuaternion

This is a quaternion quantifying the orientation of a 'subject' reference frame with respect to some other 'referent' frame. Quaternions consist of one scalar component and a 3-element vector component and can be denoted Q = (s, V), where s is a scalar and V is a vector (x, y, z). Confusion often arises regarding the meaning of the quaternion. (There are several similar but incompatible conventions.) This data type defines a so-called 'left unit transformation quaternion'. It may be used to transform the elements of a vector V_R = (xR, yR, zR) resolved in the referent frame's coordinate axes into the corresponding elements of a vector V_S = (xS, yS, zS) resolved in the subject frame's coordinate axes (i.e. determine the components of the same vector in another coordinate system). The quaternion transformation formula is (0, V_A) = Q . (0, V_B) . Q*, where '.' denotes quaternion multiplication and * denotes quaternion conjugation. The formula for multiplication of the quaternion Q1 = (s1, V1) where V1 is the vector (x1, y1, z1) by the quaternion Q2 = (s2, V2) where V2 is the vector (x2, y2, z2) is Q3 = Q1 . Q2 = (s3, V3) where s3 = s1 s2 - (x1 x2 + y1 y2 + z1 z2) and V3 = (x3, y3, z3) with x3 = s1 x2 + s2 x1 + (y1 z2 - z1 y2), y3 = s1 y2 + s2 y1 + (z1 x2 - x1 z2) and z3 = s1 z2 + s2 z1 + (x1 y2 - y1 x2). The formula for conjugation of the quaternion Q = (s, V), where V is the vector (x, y, z) is Q* = (s, -V) = (s, (-x, -y, -z)).

Encoding: HLAfixedRecord

Name	Data Type	Semantics
scalar	Scalar	The scalar component of the quaternion.
vector	Vector	The vector component of the quaternion.

Table C-6: SpaceFOM AttitudeQuaternion Type

SpaceTimeCoordinateState

A multi-dimensional representation of an observational coordinate frame and associated state. There are three spatial dimensions, three attitude dimensions and one time dimension. The spatial and attitude components define a right-handed orthogonal set of coordinate axes that constitute a reference frame. The time dimension specifies the 'position' of the coordinate with respect to the physical time scale (TT).

Encoding: HLAfixedRecord

Name	Data Type	Semantics
translational_state	ReferenceFrameTranslation	This is the reference frame's translational state with respect to its parent frame. If this frame has no parent, this attribute is meaningless.
rotational_state	ReferenceFrameRotation	This is the reference frame's rotational state with respect to its parent frame. If this frame has no parent, this attribute is meaningless.
time	Time	This specifies the simulated physical time (TT), which represents the time dimension associated with a reference frame state. It is the fourth component along with the three spatial dimensions that define a reference frame coordinate state.

Table C-7: SpaceFOM SpaceTimeCoordinate Type

C.3. The Execution Management FOM Module (SISO_SpaceFOM_management)

This SpaceFOM module provides the specifications for execution configuration and management objects, interactions and synchronization points. These are used to convey federation mode transition information and to coordinate federation mode transition behavior. These are discussed in detail in Chapter 7.

SpaceFOM Module Dependencies: SISO_SpaceFOM_switches, SISO_SpaceFOM_datatypes.

C.3.1. Object Classes

Figure C-1: Management Module Object Class Diagram

ExecutionConfiguration

SpaceFOM Class Hierarchy: *HLAobjectRoot.ExecutionConfiguration*

This is the federation Execution Configuration Object (ExCO). This object defines the base set of parameters necessary to coordinate federation and federate execution time lines and execution mode transitions in a SISO SpaceFOM-compliant federation execution.

Attribute	Datatype	Semantics
root_frame_name	HLAunicodeString	Specifies the name of the root coordinate frame in the federation execution's reference frame tree. This frame shall remain fixed throughout the federation execution.
scenario_time_epoch	Time	Federation execution scenario time epoch. This is the beginning epoch expressed in Terrestrial Time (TT), using as starting epoch that of the Truncated Julian Date (TJD)- 1968-05-24 00:00:00 UTC, that corresponds to HLA logical time 0. All joining federates shall use this time to coordinate the offset between their local simulation scenario times, their local simulation execution times and the HLA logical time.
current_execution_mode	ExecutionMode	Defines the current running state of the federation execution in terms of a finite set of states expressed in the ExecutionMode enumeration.

Attribute	Datatype	Semantics
next_execution_mode	ExecutionMode	Defines the next running state of the federation execution in terms of a finite set of states expressed in the ExecutionMode enumeration. This is used in conjunction with the next_mode_cte_time, next_mode_scenario_time and associated sync point mechanisms to coordinate federation execution mode transitions.
next_mode_scenario_time	Time	The time for the next federation execution mode change expressed as a federation scenario time reference. Note: This value is only meaningful for going into freeze; exiting freeze is coordinated through a sync point mechanism.
next_mode_cte_time	Time	The time for the next federation execution mode change expressed as a Central Timing Equipment (CTE) time reference. The standard for this reference shall be defined in the federation agreement when CTE is used.
least_common_time_step	HLAinteger64Time	A 64 bit integer time that represents microseconds for the least common value of all the time step values in the federation execution (LCTS). This value is set by the Master Federate and does not change during the federation execution. This is used in the computation to find the next HLA Logical Time Boundary (HLTB) available to all federates in the federation execution. The basic equation is HLTB = (floor(GALT/LCTS) + 1) * LCTS, where GALT is the greatest available logical time. This is used to synchronize the federates in a federation execution to be on a common logical time boundary.

Table C-8: SpaceFOM ExecutionConfiguration Object Class

Required Attribute Values: *root_frame_name*, *scenario_time_epoch*, *current_execution_mode*, *next_execution_mode*, *next_mode_scenario_time*, and *least_common_time_step*.

All the attributes values of an ExecutionControl object class instance are required except *next_mode_cte_time* if no CTE is being used. All attribute values are required when CTE is being used. There are special values for select attributes during specific phases of initialization and mode transition. For more detail on these attribute values, see Sections 7.1.3, 7.2, and 7.4.

C.3.2. Interaction Classes

Figure C-2: Management Module Interaction Class Diagram

ModeTransitionRequest

SpaceFOM Interaction Hierarchy: *HLAinteractionRoot.ModeTransitionRequest*

The MTR interaction is used by participating federates, that are not the Master Federate, to request a federation execution mode transition. An MTR can be sent at any time during initialization or execution but only certain MTR requests are valid at certain times.

Parameter	Data Type	Semantics
execution_mode	MTRMode	The run mode requested. There are only 3 valid Mode Transition Request (MTR) mode values: MTR_GOTO_RUN, MTR_GOTO_FREEZE, MTR_GOTO_SHUTDOWN. Of these three valid mode requests, only 7 combinations of current execution mode and requested mode are valid: 1. EXEC_MODE_UNINITIALIZED -> EXEC_MODE_SHUTDOWN 2. EXEC_MODE_INITIALIZED -> EXEC_MODE_FREEZE 3. EXEC_MODE_INITIALIZED -> EXEC_MODE_SHUTDOWN 4. EXEC_MODE_RUNNING -> EXEC_MODE_FREEZE 5. EXEC_MODE_RUNNING -> EXEC_MODE_SHUTDOWN 6. EXEC_MODE_FREEZE -> EXEC_MODE_RUNNING 7. EXEC_MODE_FREEZE -> EXEC_MODE_SHUTDOWN

Table C-9: SpaceFOM ModeTransitionRequest Interaction Class

Required Parameter Values: *execution_mode*.

There is only one parameter in the ModeTransitionRequest interaction and it is required. See the semantics section in Table C-9 and Section 7.1.4.

C.3.3. Data Types

ExecutionMode

Defines the mode for the running federation execution. This enumeration type is used for coordinating transitions between federation execution run states.

Representation: HLAinteger16LE

Enumerator	Value
EXEC_MODE_UNINITIALIZED	0
EXEC_MODE_INITIALIZING	1
EXEC_MODE_RUNNING	2
EXEC_MODE_FREEZE	3
EXEC_MODE_SHUTDOWN	4

Table C-10: SpaceFOM ExecutionMode Enumeration

MTRMode

MTR transition values. This enumeration is used to request a specific mode transition. However, not all mode transition requests are accepted for any given Run Mode. See mode transition validation table in the SpaceFOM documentation (Table 7-1).

Representation: HLAinteger16LE

Enumerator	Value
MTR_GOTO_RUN	2
MTR_GOTO_FREEZE	3
MTR_GOTO_SHUTDOWN	4

Table C-11: SpaceFOM MTRMode Enumeration

C.3.4. Synchronization Points

The synchronization points in the following table are used to coordinate execution control in a SpaceFOM-compliant federation execution. The first four are used for initialization control: *initialization_started*, *initialization_complete*, *objects_discovered*, and *root_frame_discovered*. The last three are used for MTR initiated execution mode control: *mtr_run*, *mtr_freeze*, and *mtr_shutdown*.

Label	Tag Data Type	Semantics
initialization_started	NA	Used to indicate that the initialization phase of a SpaceFOM-compliant federation execution has been started. This synchronization point (sync-point) is not created until all federates required by the Master Federate have joined the federation execution. Once this occurs, the Master Federate announces this sync-point along with the "startup" sync-point for any federates that have already joined the federation execution. All federates in the sync-point group must achieve this sync-point prior to proceeding with federate and federation execution initialization.
initialization_completed	NA	This synchronization point (sync-point) is registered by the federation execution Master Federate after all the early joining federates have achieved the "initialization_started" sync-point. This signals to any late joining federates that they can now proceed to the current run mode of the federation execution. This sync-point will never be achieved.
objects_discovered	NA	This synchronization point (sync-point) is used to mark the point at which all required objects have been discovered by all the federates taking part in the initialization process. This sync-point is used to ensure that all the necessary objects have been discovered prior to proceeding with the root reference frame discovery process and then multi-phase initialization.
root_frame_discovered	NA	This synchronization point (sync-point) is used to mark the point at which the root reference frame for this federation execution has been discovered by the Master Federate and all other federates participating in the initialization process. This is necessary prior to moving into the multi-phase initialization process.
mtr_run	NA	This is used to synchronize the mode transition to EXEC_MODE_RUNNING. This synchronization point (sync-point) is registered by the federation execution Master Federate upon receipt of a valid MTR interaction after sending out the associated ExCO update. Upon receiving the ExCO for the mode transition and at the associated transition time, all federates must achieve this sync-point prior to going into mode EXEC_MODE_RUNNING.

Label	Tag Data Type	Semantics
mtr_freeze	NA	This is used to synchronize the mode transition to EXEC_MODE_FREEZE. This synchronization point (sync-point) is registered by the federation execution Master Federate upon receipt of a valid MTR interaction after sending out the associated ExCO update. Upon receiving the ExCO for the mode transition and at the associated transition time, all federates must achieve this sync-point prior to going into mode EXEC_MODE_FREEZE.
mtr_shutdown	NA	This synchronization point (sync-point) is used as a marker for the mode transition to EXEC_MODE_SHUTDOWN. This sync-point is registered by the federation execution's Master Federate to "mark" the federation execution as shutting down. This marker sync-point is used in addition to the ExCO. This sync-point is never achieved and will remain for the life of the federation execution to inform any late joining federates of shutdown and that the federates should proceed directly to their shutdown processes.

Table C-12: SpaceFOM Synchronization Points

C.4. The Environment FOM Module (SISO_SpaceFOM_environment)

This SpaceFOM module provides the fundamental data types used to represent the basic physical environmental properties associated with space-based simulations. For instance, any position of an entity in a SpaceFOM simulation is related to a particular reference frame. Many different reference frames can be used in a federation execution. The Environment FOM Module specifies how these are represented and Chapter 5 of this document describes how to use them.

SpaceFOM Module Dependencies: SISO_SpaceFOM_switches, SISO_SpaceFOM_datatypes.

Figure C-3: Environment Module Object Class Diagram

C.4.1. Object Classes

ReferenceFrame

SpaceFOM Class Hierarchy: *HLAobjectRoot.ReferenceFrame*

This is an observational reference frame along with a companion right-handed orthogonal set of coordinate axes that are fixed in the frame.

Attribute	Data Type	Semantics
name	HLAunicodeString	A unique name for this reference frame instance. Reference frame names are essential in forming 'links' between parent/child reference frames.
parent_name	HLAunicodeString	The name of this frame's parent reference frame. If this frame has no parent (i.e., is a 'root' reference frame), then this string must be empty, otherwise the non-empty string must correspond to the name attribute of some other ReferenceFrame object instance in the simulation.

Attribute	Data Type	Semantics
state	SpaceTimeCoordinateState	A four dimensional representation of the reference frame with respect to its parent reference frame. If the parent fame is an empty string, then only the time dimension has meaning.

Table C-13: SpaceFOM ReferenceFrame Object Class

Required Attribute Values: *name*, *parent_name*, and *state*.

All the attributes values of a ReferenceFrame object class instance are required. There is a special case for the *parent_name* attribute of the root reference frame; it will be an empty string. See Section 5.4 for more detail.

C.5. The Entity FOM Module (SISO_SpaceFOM_entity)

This SpaceFOM module provides a generic description of physical entities with a position provided using a particular reference frame.

SpaceFOM Module Dependencies: SISO_SpaceFOM_switches, SISO_SpaceFOM_datatypes, SISO_SpaceFOM_environment.

Figure C-4: Entity Module Object Class Diagram

C.5.1. Object Classes

PhysicalEntity

SpaceFOM Class Hierarchy: *HLAobjectRoot.PhysicalEntity*

A PhysicalEntity is the highest-level object class in the SpaceFOM entity hierarchy. This object class provides attributes to describe an entity's location in time and space. It also contains attributes to uniquely identify it individually from all other physical entities in the federation execution.

Physical entities have two intrinsically associated reference frames: a 'structural frame' and a 'body frame'. These are not registered in the Federation Execution's reference frame tree (see Section 5.2.2) but are used to place and orient the entity in space with respect to a reference frame in that tree. The origin of the structural frame is located at some arbitrary but known point on the entity. The body frame origin is at the entity's center of mass. The body frame is located with respect to the entity's structural reference frame by a vector from the origin of the structural reference frame to the center of mass of the entity. This vector is expressed in the entity's structural reference frame. The orientation of the entity's body frame with respect to the entity's structural reference frame is defined by an attitude quaternion.

The position and attitude of an entity is therefore defined by the position and attitude of the entity's body frame with respect to the entity's parent_reference_frame, which must be a reference frame instance in the Federation Execution's Reference Frame Tree. This, along with time, the center_of_mass vector, and body_wrt_structural attitude quaternion, can be used to unambiguously locate the entity in time and space.

Attribute	Data Type	Semantics
name	HLAunicodeString	A non-empty string that identifies the entity. Each entity instance in the federation must have a unique name.

Attribute	Data Type	Semantics
type	HLAunicodeString	A non-empty string that identifies the entity type. It is not a mandatory field but it can be used to differentiate from a fuel tank and a space vehicle for example.
status	HLAunicodeString	An informative string that documents the current status of the entity (whatever that might be).
parent_reference_frame	HLAunicodeString	The non-empty string that identifies the reference frame with respect to which the kinematic state attributes of this entity are calculated. This string must exactly match the name of some ReferenceFrame instance in the federation.
state	SpaceTimeCoordinateState	A four dimensional representation of the entity's translational and rotational state with respect to its parent reference frame.
acceleration	AccelerationVector	A 3-vector that specifies the acceleration of the entity body frame origin (i.e., the entity's center of mass) with respect to the parent reference frame. This is the time derivative of the velocity vector as seen by an observer fixed in the parent frame. The components of this vector are resolved onto the coordinate axes of the parent frame.
rotational_acceleration	AngularAccelerationVector	A 3-vector that specifies the angular acceleration of the entity body frame with respect to the parent reference frame. This is the time derivative of the angular velocity vector as seen by an observer fixed in the parent frame. The components of this vector are resolved onto the coordinate axes of the entity body frame.
center_of_mass	PositionVector	A 3-vector that specifies the position of the entity center of mass (the body frame origin) with respect to the origin of the entity's structural frame. The components of this vector are resolved onto the coordinate axes of the structural frame.
body_wrt_structural	AttitudeQuaternion	An attitude quaternion that specifies the orientation of an entity's body frame with respect to the entity's structural frame. This attitude quaternion should never change. If not specified, an identity quaternion is assumed.

Table C-14: SpaceFOM PhysicalEntity Object Class

Required Attribute Values: *name*, *type*, *parent_reference_frame*, *state*, and *center_of_mass*.

Not all the attributes values of a PhysicalEntity object class instance are required. The attribute values for *status*, *acceleration*, *rotational_acceleration*, and *body_wrt_structural* are optional. In some cases, they are assumed to have special values when not present. For more detail, see Section 6.1.1.

DynamicalEntity

SpaceFOM Class Hierarchy: *HLAobjectRoot.PhysicalEntity.DynamicalEntity*

The DynamicalEntity object class extends the PhysicalEntity object class to provide additional attributes associated with an object subject to non-conservative dynamic forces and/or torques. Specifically, the DynamicalEntity provides additional force, torque, and mass property related parameters. These are usually associated with environmental effects and vehicle effector systems. These can be used for both visualization and to improve state propagation between updates.

Attribute	Data Type	Semantics
force	ForceVector	A 3-vector that specifies the total external force on the entity. Force is expressed and applied in the entity's structural reference frame.
torque	TorqueVector	A 3-vector that specifies the total external torque on the entity. It is expressed in the entity's structural reference frame.
mass	Mass	The mass of the DynamicalEntity.
mass_rate	MassRate	The time rate of change of the DynamicalEntity's mass.
inertia	InertiaMatrix	A 3x3 matrix that specifies the centroid moments and coefficients of inertia with respect to the coordinate axes of the DynamicalEntity's body frame
inertia_rate	InertiaRateMatrix	A 3x3 matrix that specifies the time rate of change of the parameters in the InertiaMatrix. The elements in this matrix correspond directly to the elements in the InteriaMatrix.
name	HLAunicodeString	A non-empty string that identifies the entity. Each entity instance in the federation must have a unique name.
type	HLAunicodeString	A non-empty string that identifies the entity type. It is not a mandatory field but it can be used to differentiate from a fuel tank and a space vehicle for example.
status	HLAunicodeString	An informative string that documents the current status of the entity (whatever that might be).
parent_reference_frame	HLAunicodeString	The non-empty string that identifies the reference frame with respect to which the kinematic state attributes of this entity are calculated. This string must exactly match the name of some ReferenceFrame instance in the federation.
state	SpaceTimeCoordinateState	A four dimensional representation of the entity's translational and rotational state with respect to its parent reference frame.

Attribute	Data Type	Semantics
acceleration	AccelerationVector	A 3-vector that specifies the acceleration of the entity body frame origin (i.e., the entity's center of mass) with respect to the parent reference frame. This is the time derivative of the velocity vector as seen by an observer fixed in the parent frame. The components of this vector are resolved onto the coordinate axes of the parent frame.
rotational_acceleration	AngularAccelerationVector	A 3-vector that specifies the angular acceleration of the entity body frame with respect to the parent reference frame. This is the time derivative of the angular velocity vector as seen by an observer fixed in the parent frame. The components of this vector are resolved onto the coordinate axes of the entity body frame.
center_of_mass	PositionVector	A 3-vector that specifies the position of the entity center of mass (the body frame origin) with respect to the origin of the entity's structural frame. The components of this vector are resolved onto the coordinate axes of the structural frame.
body_wrt_structural	AttitudeQuaternion	An attitude quaternion that specifies the orientation of an entity's body frame with respect to the entity's structural frame. This attitude quaternion should never change. If not specified, an identity quaternion is assumed.

Table C-15: SpaceFOM DynamicalEntity Object Class

Required Attribute Values: *name*, *type*, *parent_reference_frame*, *state*, *center_of_mass*, *mass*, and *inertia*.

Since a DynamicalEntity inherits from PhysicalEntity, like PhysicalEntity, not all the attributes values of a DynamicalEntity object class instance are required. However, all the attribute values that are required in a PhysicalEntity are also required for a DynamicalEntity, along with the additional *mass* and *inertia* attributes. The attribute values for *status*, *acceleration*, *rotational_acceleration*, and *body_wrt_structural* are optional. Also, the additional attribute values of *force*, *torque*, *mass_rate*, and *inertia_rate* are optional. In some cases, they are assumed to have special values when not present. For more detail, see Section 6.1.2.

PhysicalInterface

SpaceFOM Class Hierarchy: *HLAobjectRoot.PhysicalInterface*

Represents a location and orientation with respect to another frame. It is intended to act as a base representation for the position and orientation of an interface associated with either another PhysicalInterface instance or a PhysicalEntity instance. In either case, the position and orientation of the interface are specified with respect to the structural reference frame of the entity to which it is attached. This Object Class can be used as a common base for derived interfaces like grapple fixtures, docking ports, berthing interfaces, etc.

Attribute	Datatype	Semantics
name	HLAunicodeString	A non-empty string that identifies the interface. Each PhysicalInterface instance in the federation must have a unique name.

Attribute	Datatype	Semantics
parent_name	HLAunicodeString	The HLA Object Instance Name of the PhysicalEntity or PhysicalInterface to which this interface is attached.
position	PositionVector	A 3-vector that specifies the position of the interface reference frame origin with respect to the parent structural reference frame. The components of this vector are resolved onto the coordinate axes of the parent frame.
attitude	AttitudeQuaternion	An attitude quaternion of the interfaces reference frame ('subject frame') with respect to its parent structural reference frame ('referent frame').

Table C-16: SpaceFOM PhysicalInterface Object Class

Required Attribute Values: *name*, *type*, *parent_name*, *position*, and *attitude*.

All the attribute values for a PhysicalInterface object class instance are required. For more detail, see Section 6.2.

Appendix D. [Informative] Time Scales and Time Scale Relationships

D.1. Time Scales

A time scale is used to define absolute time. The Terrestrial Time (TT) time scale is used to represent absolute time for all SpaceFOM scenario physical time stamps. However, based on the technologies and needs of the historical time periods, a number of different time scales have been defined. These alternate time scales may also be important within federates in a federation execution. This appendix provides brief descriptions of common standard time scales.

D.1.1. Sidereal Time

Sidereal Time is the time between successive transits of a star over a particular meridian. This defines a sidereal day. In essence, sidereal time is the rotation of the Earth with respect to the celestial star field (see Figure D-1). By defining Sidereal Time as the hour angle from the vernal equinox to the local meridian, then the sidereal time associated with the Greenwich meridian is termed Greenwich Mean Sidereal Time, GMST, or θ_{GMST}. The sidereal time at a local meridian is called Local Sidereal Time, LST, or θ_{LST}. Note that GMST has recently been replaced by a measure of the Earth's rotation angle with respect to the International Celestial Reference Frame. This is called the Earth Rotation Angle (ERA).

D.1.2. Solar Time

Solar Time is the time between successive transits of the Sun over a particular meridian. This defines a solar day. There are two types of solar time: Apparent Solar Time (sundial time) and Mean Solar Time (clock time). The length of an Apparent Solar day varies by a small amount each day (see Table D-1). This is due to the eccentricity of the Earth's orbit around the Sun and the Earth's inclination to the orbit.

Date	Duration in mean solar time
February 11	24 hours
March 26	24 hours – 18.1 seconds
May 14	24 hours
June 19	24 hours + 13.1 seconds
July 26	24 hours
September 16	24 hours – 21.3
November 3	24 hours
December 22	24 hours + 29.9

Table D-1: Length of Apparent Solar Day (1998)

Having the length of a day vary isn't a convenient feature. The concept of Mean Solar Time was invented to address this problem. Mean Solar Time is based off the motion of a Fictitious Mean Sun that has uniform motion along the celestial equator and is obtained through measurement of the Earth's orientation.

D.1.3. Universal Time (UT)

Universal Time (UT) is defined as the Mean Solar Time at the Royal Observatory in Greenwich, England. There are three variations of Universal Time. UT0 is the observed mean solar time. UT1 is UT0 corrected for polar motion, the motion of the Earth's rotational axis over the surface of the Earth. UT1 is the most common and relevant to the following discussions. UT2 is corrected for seasonal variations and is considered obsolete.

D.1.4. Atomic Time (TAI)

International Atomic Time (TAI) (Temps Atomique International) was introduced in 1972 and is based on the SI second [28]. It is a highly accurate time system that is independent of the complexities and variability of the Earth rotation based time systems. This time scale is accurate enough to observe relativistic effects for clocks in motion or accelerated by a local gravity field. One advantage of using TAI is that it is a continuous uniform time scale. Specifically, the rate of time passage for TAI is constant unlike the Earth rotation based scales. This means that the Earth rotation based time scales diverge from TAI over time due to the variations in the Earth's rotation.

D.1.5. Coordinated Universal Time (UTC)

Coordinated Universal Time (UTC) is a time scale defined on the SI second but adjusted at irregular intervals to be with ±0.9 seconds of UT1. This means that the rate of time passage in UTC is the same as TAI. However, leap seconds are added by consensus when needed to track UT1. UTC is the de facto time scale for the world time keeping. However, due to the addition of leap seconds, it is not continuous. Also, conversion to and from UT1 requires a correction factor the must be observed or predicted.

D.1.6. GPS Time (GPST)

Global Positioning System (GPS) time is the uniform time scale based on the SI second with a starting epoch at midnight between Saturday January 5th and Sunday January 6th, 1980 (1980 January 6, 00:00:00 UTC). GPS Time counts in weeks and seconds of a week from this instant. The GPS week begins at the transition between Saturday and Sunday. The days of the week are numbered sequentially, with Sunday being 0, Monday 1, Tuesday 2, etc. The GPS time scale begins at the GPS starting epoch with GPS week 0. Within each week the time is usually denoted as the second of the week (SOW). This is a number between 0 and 604,800 (60 x 60 x 24x 7). Sometimes SOW is split into a day of week (DOW) between 0 and 6 and a second of day (SOD) between 0 and 86400.

While GPST is a uniform time scale, it does have rollover. To limit the size of the numbers used in the data and calculations, the GPS Week Number is a ten-bit count in the range 0-1023, repeating every 1024 weeks. As a result, the week number 'rolled over' from 1023 to 0 at 23:59:47 UTC on Saturday, 21st August 1999. This was before midnight UTC because every GPS week contains exactly 604,800 seconds, to keep the calculations consistent. The 13 intervening leap seconds had put UTC behind GPS system time. The last GPS week rollover occurred on 2019 April 06. GPS systems are currently being upgraded to prevent the need for rollover in the future.

D.1.7. Unix Time (UNXT)

Unix Time (UNXT) (also known as POSIX time or Epoch time) is a non-uniform time scale based on UTC and is used by many Unix based computer systems. UNXT is defined as the number of seconds that have elapsed since 00:00:00 Thursday, 1 January 1970 UTC not counting leap seconds. Specifically, when a leap second is inserted, Unix time numbers repeat themselves. This leads to an ambiguity with the time of the leap second and the time immediately after the leap second. Because it does not handle leap seconds, it is neither a linear representation of time nor a true representation of UTC. While POSIX compliant Unix systems provide both a command line and C language utilities to get a UTC time and date, UNXT is not a particularly good time scale for a model or simulation.

D.1.8. Terrestrial Time (TT)

Terrestrial Time (TT) is the time scale used to represent absolute time for all SpaceFOM scenario physical time stamps. Terrestrial Time is an astronomical or "dynamical" time scale used widely for geocentric and topocentric ephemerides. TT is defined to run at the same rate as TAI seconds but with an offset of 32.184 seconds. This offset is based on preserving continuity with other historical dynamic time scales.

D.1.9. Geocentric Coordinated Time (TCG)

Geocentric Coordinated Time (TCG) is a time scale primarily used for theoretical developments based on the Geocentric Celestial Reference System (GCRS). TCG is a relativistic time scale and advances at a rate 6.97×10^{-10} faster than SI seconds. TCG, TCB and TT are defined in a way that they have the same value on 1977 January 1, 00:00:00 TAI (JD 2443144.5 TAI).

D.1.10. Barycentric Coordinated Time (TCB)

Barycentric Coordinated Time (TCB) is a time scale primarily used for theoretical developments based on the Barycentric Celestial Reference System (BCRS). TCB is a relativistic time scale and advances at a rate 1.55×10^{-8} faster than SI seconds. TCG, TCB and TT are defined in a way that they have the same value on 1977 January 1, 00:00:00 TAI (JD 2443144.5 TAI).

D.2. Time Scale Relationships

This section provides equations that relate the various time scales discussed in Appendix Section D.1.

D.2.1. Apparent Solar Time vs. Mean Solar Time

The Equation of Time gives the difference between apparent and mean solar time:

$$EQ_{time} = -1.91466471sin(M_S) - 0.019994643sin(2M_S) + 2.466sin(2\lambda_{ecliptic}) - 0.0053sin(4\lambda_{ecliptic})$$

Where M_S is the mean anomaly of the Sun and $\lambda_{ecliptic}$ is the longitude of the ecliptic.

D.2.2. Mean Solar Time vs. Sidereal Time

The following figure is an exaggerated illustration of the difference between a sidereal day and a mean solar day:

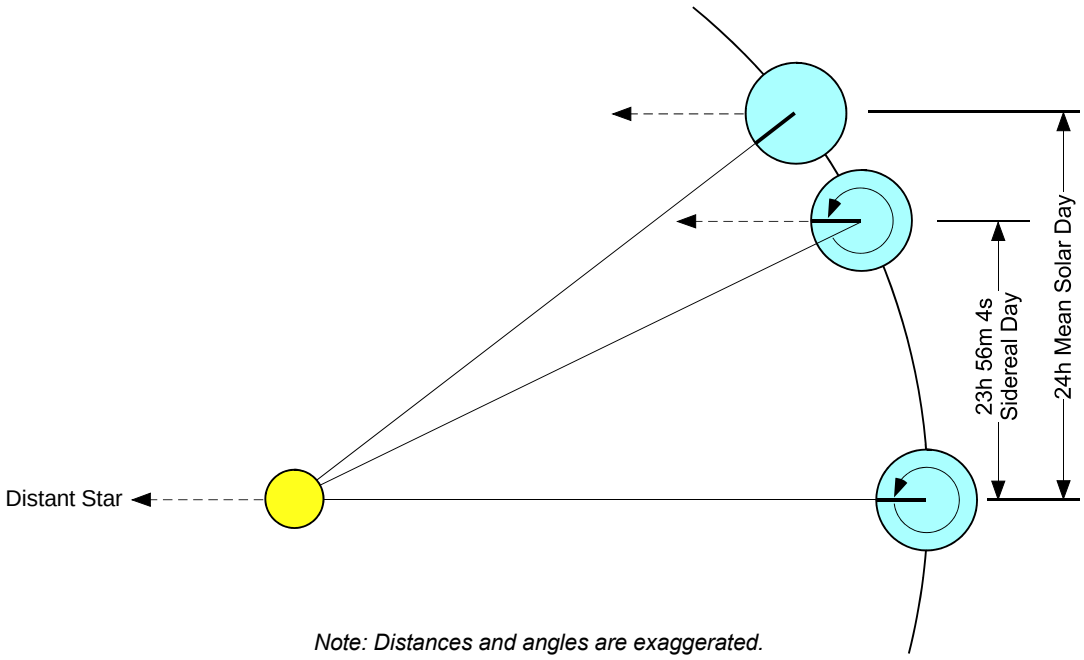

Figure D-1: Sidereal Day Compared To Mean Solar Day

Based on the orbital motion of the Earth around the Sun, the difference between a mean solar day and a sidereal day can be computed (see Table D-2).

1 solar day	=	1.002737909350795 sidereal days
1 sidereal day	=	0.997269566329084 solar days
1 solar day	=	24h 3m 56.5553678 s sidereal time
1 sidereal day	=	23h 56m 4.090524 s solar time

Table D-2: Solar Day / Sidereal Day Conversions

D.2.3. UT1 to UTC

The relationship between UT1 and UTC is expressed in the following equation:

$$UTC = UT1 - (UT1 - UTC) \approx UT1 - DUT1$$

where $DUT1$ is a correction factor that keeps the difference between UT1 and UTC within 0.9 seconds. Because UT1 varies irregularly due to variations in the Earth's rotation $DUT1$ varies continuously, and leap seconds must be periodically inserted to keep UT1 and UTC to within the ±0.9s difference. The value of $\Delta DUT1$ must be observed or predicted. $DUT1$ is a broadcast approximation of $UT1 - UTC$ and maintains an accuracy of ±0.1 seconds.

D.2.4. UT1 to TT

The relationship between UT1 and TT is expressed in the following equation:

$$TT = UT1 + \Delta T$$

where $\Delta T = 32.184 + \Delta AT - (UT1 - UTC)$. Values for ΔT, $(UT1 - UTC)$, and ΔAT are published.

D.2.5. UTC to TAI

The relationship between UTC and TAI is expressed in the following equation:

$$TAI = UTC + \Delta AT$$

where ΔAT are the integer leap seconds necessary to keep the UTC time scale to be within ±0.9seconds. Past values for ΔAT are published; future values must be predicted.

D.2.6. TAI to GPST

The relationship between TAI and GPST is expressed in the following equation:

$$GPST = TAI - 19$$

Note that GPST can be computed from UTC by using the expression for TAI in Section D.2.5.

D.2.7. TAI to TT

The relationship between TAI and TT is expressed in the following equation:

$$TT = TAI + 32.184$$

Note that TT and TAI only differ by a constant 32.184 second offset.

D.2.8. TCG to TT

The relationship between TCG and TT is expressed in the following equation:

$$\frac{dTT}{dTCG} = 1 - L_G$$

where L_G= 6.969290134×10^{-10} (exactly). This gives

$$TT = TCG - L\_G\ (TCG - t_0)$$

where t_0 is 1977 January 1, 00:00:00 TAI.

D.2.9. Time Scales from TT

Here are some common time scale conversion equations from TT:

$$UT1 = TT - \Delta T$$

$$UTC = TT + \Delta AT - 32.184$$

$$TAI = TT - 32.184$$

$$GPST = TT - 51.184$$

$$TCG = \frac{1}{(1 - L_G)}(TT - t_0)$$

The parameters ΔT, ΔAT, L_G, and t_0 are defined in the preceding sections dealing with corresponding time scale on the left hand side of the equations.

Appendix E. [Informative] Reference Frame Transformations

It was probably apparent from the content of the ReferenceFrameTranslation and ReferenceFrameRotation fixed records that the reference frame transformations would involve vector and quaternion equations. There are many good linear algebra textbooks that cover vectors and vector algebra; however, it may be more difficult to find a good reference on quaternions and quaternion algebra. The following sections will provide the nomenclature, brief description of quaternions, and basic vector transformations used for reference frame transformations.

E.1. Nomenclature

The first step in formulating the equations used for reference frame transformations is to define the nomenclature used in the equations:

A	A specific reference frame named A.
$\vec{\boldsymbol{a}}$	A generic acceleration vector ($\dot{\vec{\boldsymbol{v}}}$ - time derivative of a velocity vector).
$\vec{\boldsymbol{a}}_{\mathrm{A}}$	A generic acceleration vector expressed with respect to reference frame A.
$\boldsymbol{I}$	Identity matrix.
$\tilde{\boldsymbol{\imath}}$	Identity quaternion.
$\hat{\mathbf{i}}$	Orthogonal basis vector defining the x or 1 direction of a reference frame.
$\hat{\mathbf{j}}$	Orthogonal basis vector defining the y or 2 direction of a reference frame.
$\hat{\mathbf{k}}$	Orthogonal basis vector defining the z or 3 direction of a reference frame.
$\tilde{\boldsymbol{Q}}(\vec{\boldsymbol{x}})$	A quaternion rotation operator on the vector $\vec{\boldsymbol{x}}$ for the quaternion $\tilde{\boldsymbol{q}}$.
$\tilde{\boldsymbol{Q}}^{*}(\vec{\boldsymbol{x}})$	Conjugate quaternion rotation operator on the vector $\vec{\boldsymbol{x}}$ for the quaternion $\tilde{\boldsymbol{q}}$.
$\tilde{\boldsymbol{Q}}_{\mathrm{A}\to\mathrm{B}}(\vec{\boldsymbol{x}})$	A quaternion rotation operator on the vector $\vec{\boldsymbol{x}}$ for the quaternion $\tilde{\boldsymbol{q}}_{\mathrm{A}\to\mathrm{B}}$.
$\tilde{\boldsymbol{q}}$	A generic quaternion.
$\tilde{\boldsymbol{q}}^{*}$	Conjugate of a generic quaternion.
$\tilde{\boldsymbol{q}}_{\mathrm{A}\to\mathrm{B}}$	A quaternion for a frame rotation from frame A to frame B.
$\tilde{\boldsymbol{q}}_{0}$	A pure quaternion (a way of representing a vector $\vec{\boldsymbol{q}}$ as a quaternion).
$\vec{\boldsymbol{r}}$	A generic position vector.
$\vec{\boldsymbol{r}}_{\mathrm{A}}$	A generic position vector expressed with respect to reference frame A.
$\boldsymbol{T}$	A generic rotation matrix.
$\boldsymbol{T}^{\boldsymbol{T}}$	Transpose of a rotation matrix.
$\boldsymbol{T}_{\mathrm{A}\to\mathrm{B}}$	A rotation matrix for a transformation from frame A to frame B.
$\vec{\boldsymbol{u}}$	A unit vector.
$\vec{\boldsymbol{v}}$	A generic velocity vector ($\dot{\vec{\boldsymbol{r}}}$ - time derivative of a position vector).
$\vec{\boldsymbol{v}}_{\mathrm{A}}$	A generic velocity vector expressed with respect to reference frame A.
x	A scalar variable.
$\vec{\boldsymbol{x}}$	A vector variable.
$\vec{\boldsymbol{x}}_{\mathrm{A}}$	A vector expressed with respect to reference frame A.
$\dot{\vec{\boldsymbol{x}}}$	First time derivative of the vector $\vec{\boldsymbol{x}}$.
$\ddot{\vec{\boldsymbol{x}}}$	Second time derivative of the vector $\vec{\boldsymbol{x}}$.
α	Angular acceleration.
$\vec{\boldsymbol{\alpha}}$	Angular acceleration vector ($\dot{\vec{\boldsymbol{\omega}}}$ - time derivative of an angular velocity vector).
θ	Rotation angle
ω	Angular velocity
$\vec{\boldsymbol{\omega}}$	Angular velocity vector

The general form for a vector is:

$$\vec{\boldsymbol{x}} = \hat{\mathbf{i}}x_1 + \hat{\mathbf{j}}x_2 + \hat{\mathbf{k}}x_3$$

where $\hat{\mathbf{i}}$, $\hat{\mathbf{j}}$, and $\hat{\mathbf{k}}$ are the orthogonal basis vectors that define the reference frame in which the vector is expressed.

E.2. Quaternions

William Rowan Hamilton invented the concept of a quaternion in 1843. A quaternion is a hyper-complex number of rank 4. By comparison, scalars are rank 1 and complex numbers are rank 2. Quaternions can be used to represent vector rotations and frame rotations. Typically a quaternion is represented as a combination of a scalar part and a 3-vector part. We can define the quaternion $\tilde{\boldsymbol{q}}$ as follows:

$$\tilde{\boldsymbol{q}} = q_s + \hat{\mathbf{i}}q_1 + \hat{\mathbf{j}}q_2 + \hat{\mathbf{k}}q_3 = q_s + \vec{\boldsymbol{q}}$$

where q_s represents the scalar component and $\vec{\boldsymbol{q}}$ is the vector component.

Quaternions form a mathematical system called a *non-commutative division ring*. This means that quaternions satisfy the following condition:

1. Closed under addition and multiplication; specifically, the result is another quaternion:
$$\tilde{\boldsymbol{a}} + \tilde{\boldsymbol{b}} = \tilde{\boldsymbol{c}}$$
$$\tilde{\boldsymbol{a}}\tilde{\boldsymbol{b}} = \tilde{\boldsymbol{c}}$$
2. Addition and multiplication operations are associative:
$$(\tilde{\boldsymbol{a}} + \tilde{\boldsymbol{b}}) + \tilde{\boldsymbol{c}} = \tilde{\boldsymbol{a}} + (\tilde{\boldsymbol{b}} + \tilde{\boldsymbol{c}})$$
$$(\tilde{\boldsymbol{a}}\tilde{\boldsymbol{b}})\tilde{\boldsymbol{c}} = \tilde{\boldsymbol{a}}(\tilde{\boldsymbol{b}}\tilde{\boldsymbol{c}})$$
3. Addition operations are commutative:
$$\tilde{\boldsymbol{a}} + \tilde{\boldsymbol{b}} = \tilde{\boldsymbol{b}} + \tilde{\boldsymbol{a}}$$
4. Multiplication operations are NOT generally commutative (non-commutative): [27]
$$\tilde{\boldsymbol{a}}\tilde{\boldsymbol{b}} \neq \tilde{\boldsymbol{b}}\tilde{\boldsymbol{a}}$$
5. There is an identity for addition:
$$\tilde{\boldsymbol{a}} + 0 = \tilde{\boldsymbol{a}}$$
6. There is an identity for multiplication:
$$\tilde{\boldsymbol{a}}\tilde{\boldsymbol{\imath}} = \tilde{\boldsymbol{a}}$$
7. There is an inverse:
$$\tilde{\boldsymbol{a}}\tilde{\boldsymbol{a}}^{-1} = \tilde{\boldsymbol{\imath}} = 1$$
8. Multiplication is distributive over addition:
$$\tilde{\boldsymbol{a}}\left(\tilde{\boldsymbol{b}} + \tilde{\boldsymbol{c}}\right) = \tilde{\boldsymbol{a}}\tilde{\boldsymbol{b}} + \tilde{\boldsymbol{a}}\tilde{\boldsymbol{c}}$$

Furthermore, quaternion addition for two quaternions $\tilde{\boldsymbol{a}}$ and $\tilde{\boldsymbol{b}}$ is defined as:

$$\tilde{\boldsymbol{a}} + \tilde{\boldsymbol{b}} = (a_s + b_s) + \hat{\mathbf{i}}(a_1 + b_1) + \hat{\mathbf{j}}(a_2 + b_2) + \hat{\mathbf{k}}(a_3 + b_3)$$

Quaternion multiplication for two quaternions $\tilde{\boldsymbol{a}}$ and $\tilde{\boldsymbol{b}}$ is defined as:

$$\tilde{\boldsymbol{a}}\tilde{\boldsymbol{b}} = \tilde{\boldsymbol{c}} = c_s + \vec{\boldsymbol{c}} = c_s + \hat{\mathbf{i}}c_1 + \hat{\mathbf{j}}c_2 + \hat{\mathbf{k}}c_3$$

where

$$\begin{aligned} c_s &= a_s b_s - a_1 b_1 - a_2 b_2 - a_3 b_3 \\ c_1 &= a_s b_1 + a_1 b_s + a_2 b_3 - a_3 b_2 \\ c_2 &= a_s b_2 - a_1 b_3 + a_2 b_s + a_3 b_1 \\ c_3 &= a_s b_3 + a_1 b_2 - a_2 b_1 + a_3 b_s \end{aligned}$$

A 3-vector can be treated like a special form of quaternion called a *pure quaternion*:

$$\tilde{\boldsymbol{q}}_0 = 0 + \vec{\boldsymbol{q}} = \hat{\mathbf{i}}q_1 + \hat{\mathbf{j}}q_2 + \hat{\mathbf{k}}q_3$$

[27] This is a significant difference from scalars and complex numbers in that they do commute under multiplication. In fact no hyper complex numbers of rank 3 or more commute under multiplication.

By treating 3-vectors as pure quaternions, quaternion and vector addition and multiplication follow the quaternion definitions above.

The complex conjugate of a quaternion is:

$$\tilde{\boldsymbol{q}}^* = q_s - \vec{\boldsymbol{q}}$$

Two important characteristic of a quaternion complex conjugate are:

$$\left(\tilde{\boldsymbol{a}}\tilde{\boldsymbol{b}}\right)^* = \tilde{\boldsymbol{b}}^*\tilde{\boldsymbol{a}}^*$$

and

$$\tilde{\boldsymbol{q}} + \tilde{\boldsymbol{q}}^* = 2q_s$$

The norm operator for a quaternion is:

$$|\tilde{\boldsymbol{q}}| = \sqrt{\tilde{\boldsymbol{q}}^*\tilde{\boldsymbol{q}}} = \sqrt{q_0^2 + q_1^2 + q_2^2 + q_3^2}$$

The inverse of a quaternion is:

$$\tilde{\boldsymbol{q}}^{-1} = \frac{\tilde{\boldsymbol{q}}^*}{|\tilde{\boldsymbol{q}}|^2}$$

For the special case of a unit quaternion ($|\tilde{\boldsymbol{q}}| = 1$), the inverse is the complex conjugate:

$$\tilde{\boldsymbol{q}}^{-1} = \tilde{\boldsymbol{q}}^*$$

As stated above, a quaternion can be used to represent a vector or frame rotation. Without going into the details of why, a unit quaternion can be used to represent a single axis rotation for a vector or frame. If $\vec{\boldsymbol{u}}$ is a unit vector representing the axis of rotation and θ corresponds to the angle of rotation then the corresponding unit quaternion is defined by:

$$\tilde{\boldsymbol{q}} = \cos\frac{\theta}{2} + \vec{\boldsymbol{u}}\sin\frac{\theta}{2}$$

E.2.1. Right versus Left

It is this use of the quaternion as a rotation operator that fostered the terminology of a right versus a left quaternion operation (sometimes referred to as right-handed or left-handed quaternions). Something is a quaternion if it follows the rules described above. It is only right or left when considering the definition of what constitutes the positive rotational direction. For a right quaternion, the positive direction follows the right hand rule where a positive rotation is counter-clockwise about the rotation axis. For a left quaternion, the positive direction is clockwise about the rotation axis. The implication of this is that the exact same quaternion can be expressed as a rotation in two different ways.

$$\tilde{\boldsymbol{q}}_{right} = \cos\frac{\alpha}{2} + \vec{\boldsymbol{u}}\sin\frac{\alpha}{2}$$

where α is measured positive counter-clockwise (right handed) and

$$\tilde{\boldsymbol{q}}_{left} = \cos\frac{\beta}{2} + \vec{\boldsymbol{u}}\sin\frac{\beta}{2}$$

where β is measured positive clockwise (left handed).

From a geometric perspective, for these quaternions to represent the same rotation operation then $\alpha = -\beta$. Using the trigonometric identities $\cos(-\theta) = \cos\theta$, $\sin(-\theta) = -\sin\theta$ and the definition for the quaternion complex conjugate, gives the following:

$$\tilde{\boldsymbol{q}} = \tilde{\boldsymbol{q}}_{right} = \cos\frac{\alpha}{2} + \vec{\boldsymbol{u}}\sin\frac{\alpha}{2} = \cos\left(\frac{-\beta}{2}\right) + \vec{\boldsymbol{u}}\sin\left(\frac{-\beta}{2}\right) = \cos\frac{\beta}{2} - \vec{\boldsymbol{u}}\sin\frac{\beta}{2} = \tilde{\boldsymbol{q}}_{left}^*$$

This implies that the left quaternion representation is the complex conjugate of the right quaternion representation. This is a very useful relationship when converting between rotation conventions.

The question then is which quaternion rotation convention do we use, right or left? In general, it does not matter; either will work. The quaternion algebra is the same. However, it is important when computing the quaternion rotation operator, the corresponding rotation matrix, and any corresponding Euler angles.

E.2.2. Vector Rotation versus Frame Rotation

As stated earlier, a quaternion can be used to perform a vector rotation or a frame rotation (also known as a frame transformation). What is the difference and how does that affect the use of the quaternion?

We will define a vector or point rotation as the process of rotating a vector expressed in a given coordinate frame through an angle α about a fixed single axis $\vec{\boldsymbol{u}}$.

$$\tilde{\boldsymbol{q}}_{vector} = \cos\frac{\alpha}{2} + \vec{\boldsymbol{u}}\sin\frac{\alpha}{2}$$

We will define a frame rotation as the rotation of a coordinate frame through an angle β about a fixed single axis $\boldsymbol{u}$.

$$\tilde{\boldsymbol{q}}_{frame} = \cos\frac{\beta}{2} + \vec{\boldsymbol{u}}\sin\frac{\beta}{2}$$

The difference in the two cases is that in the first the vector moves with respect to a fixed frame and in the second a fixed vector is expressed in a rotated or transformed frame.

It can be shown that rotating a vector through an angle θ about a fixed single axis $\boldsymbol{u}$ is equivalent to rotating a frame through an angle $-\theta$ about the same fixed single axis $\vec{\boldsymbol{u}}$. In other words, $\alpha = -\beta$. Using the trigonometric identities $\cos(-\theta) = \cos\theta$, $\sin(-\theta) = -\sin\theta$ and the definition for the quaternion complex conjugate, gives the following:

$$\tilde{\boldsymbol{q}}_{vector} = \cos\frac{\alpha}{2} + \vec{\boldsymbol{u}}\sin\frac{\alpha}{2} = \cos\left(\frac{-\beta}{2}\right) + \vec{\boldsymbol{u}}\sin\left(\frac{-\beta}{2}\right) = \cos\frac{\beta}{2} - \vec{\boldsymbol{u}}\sin\frac{\beta}{2} = \tilde{\boldsymbol{q}}^{*}_{frame}$$

This implies that quaternion representation of a frame transformation is the complex conjugate of the quaternion representation of a vector rotation.

The question then is which operation should we use, vector rotation or frame transformation? Again, in general, it does not matter; either will work. The quaternion algebra is the same. However, like the rotation convention, it is important when computing the quaternion rotation operator, the corresponding rotation matrix and any corresponding Euler angles.

E.2.3. Quaternion Rotation Operator

Having now discussed quaternions in some detail, we now have some understanding on how they work. The question now is how do we use quaternions to perform these operations of vector rotation and frame transformation? The answer is through the use of a quaternion triple product called the *quaternion rotation operator*. For vector rotation using a right quaternion, the quaternion rotation operator is defined as:

$$\tilde{\boldsymbol{Q}}_{right\_vector}(\vec{\boldsymbol{x}}) = \tilde{\boldsymbol{q}}\vec{\boldsymbol{x}}\tilde{\boldsymbol{q}}^{*} = \vec{\boldsymbol{y}}$$

where $\tilde{\boldsymbol{Q}}(\,)$ is the quaternion rotation operator, $\tilde{\boldsymbol{q}}$ is the right quaternion representing the rotation operation, $\vec{\boldsymbol{x}}$ is the original vector, and $\vec{\boldsymbol{y}}$ is the resulting vector. Note, both $\vec{\boldsymbol{x}}$ and $\vec{\boldsymbol{y}}$ are expressed in the same frame.

However, quaternions can also be used for frame transformations where a fixed vector is expressed with respect to a rotated frame. For frame transformations using a right quaternion, the quaternion rotation operator is defined as:

$$\tilde{\boldsymbol{Q}}_{right\_frame}(\vec{\boldsymbol{x}}) = \tilde{\boldsymbol{q}}^{*}\vec{\boldsymbol{x}}\tilde{\boldsymbol{q}} = \vec{\boldsymbol{y}}$$

where $\tilde{\boldsymbol{Q}}(\,)$ is the quaternion rotation operator, $\tilde{\boldsymbol{q}}$ is the right quaternion representing the frame rotation, $\vec{\boldsymbol{x}}$ is the fixed vector expressed in the original frame, and $\vec{\boldsymbol{y}}$ is the fixed vector expressed in the rotated frame. Note here that vectors $\vec{\boldsymbol{x}}$ and $\vec{\boldsymbol{y}}$ are actually the same vector expressed in different frames.

However, we need to be careful here. There is also a difference between the quaternion rotation conventions for right and left. By applying the complex conjugate relationships to vector rotation and frame transformation, we get the following table of quaternion rotation operator conventions:

	Left	Right
Vector Rotation	$\vec{y} = \tilde{q}^* \vec{x} \tilde{q}$	$\vec{y} = \tilde{q} \vec{x} \tilde{q}^*$
Frame Transform	$\vec{y} = \tilde{q} \vec{x} \tilde{q}^*$	$\vec{y} = \tilde{q}^* \vec{x} \tilde{q}$

Table E-1: Quaternion Rotation Operator Conventions

Again, note that for vector rotations, $\tilde{q}$ is the quaternion representing the vector rotation, $\vec{x}$ is the vector to be rotated, and $\vec{y}$ is the resulting vector. Both $\vec{x}$ and $\vec{y}$ are expressed in the same frame. For frame transformations, $\tilde{q}$ is the quaternion representing the frame rotation, $\vec{x}$ is the fixed vector expressed in the original frame, and $\vec{y}$ is the fixed vector expressed in the rotated frame. For frame transformation the vectors $\vec{x}$ and $\vec{y}$ are actually the same vector expressed in different frames.

Quaternions specified in the SpaceFOM use the left rotation convention for quaternion representations. This is the convention used for the remainder of the document. In most cases, we will be using the quaternion to transform an instantaneously fixed point represented by a vector and expressed in one reference frame into a vector expressed in a different reference frame. Therefore, the quaternion rotation operator used to transform a vector expressed in one frame into a vector expressed in another frame is the one from the lower left corner of Table E-1:

$$\tilde{Q}(\vec{x}) = \tilde{q} \vec{x} \tilde{q}^* = \vec{y}$$

What about multiple quaternion rotation operations? Consider a sequence of two quaternion rotation operations using the quaternion $\tilde{a}$ and then the quaternion $\tilde{b}$. These result in the following nested quaternion rotation operators $\tilde{A}(\,)$ and then $\tilde{B}(\,)$:

$$\tilde{B}\left(\tilde{A}(\vec{x})\right) = \tilde{b}(\tilde{a} \vec{x} \tilde{a}^*) \tilde{b}^* = \tilde{b} \tilde{a} \vec{x} \tilde{a}^* \tilde{b}^* = \tilde{c} \vec{x} \tilde{c}^* = \tilde{C}(\vec{x})$$

where $\tilde{c} = \tilde{b} \tilde{a}$. This indicates that a sequential series of quaternion rotation operators can be replaced by a single quaternion rotation operator for the quaternion produced by the sequential multiplication of the constituent quaternions.

E.3. Quaternions and Rotation Matrices

It should be clear from the preceding discussions on quaternions and quaternion rotation operators that there is a direct association between those and the classic direction cosine rotation matrices used for vector transformation in classic vector-matrix based linear algebra. This can be seen from the following equation:

$$\vec{y} = \tilde{Q}(\vec{x}) = \tilde{q} \vec{x} \tilde{q}^* = \boldsymbol{T} \vec{x}$$

where $\boldsymbol{T}$ is the rotation matrix which transforms the vector $\vec{x}$ into the vector $\vec{y}$ in just the same way as the quaternion rotation operator $\tilde{Q}(\vec{x})$. In fact, the elements of transformation matrix $\boldsymbol{T}$ can be formulated in terms of the elements of the quaternion $\tilde{q}$ as follows:

$$\boldsymbol{T} = \begin{bmatrix} t_{11} & t_{12} & t_{13} \\ t_{21} & t_{22} & t_{23} \\ t_{31} & t_{32} & t_{33} \end{bmatrix} = \begin{bmatrix} 2q_s^2 - 1 + 2q_1^2 & 2q_1 q_2 - 2q_s q_3 & 2q_1 q_3 + 2q_s q_2 \\ 2q_1 q_2 + 2q_s q_3 & 2q_s^2 - 1 + 2q_2^2 & 2q_2 q_3 - 2q_s q_1 \\ 2q_1 q_3 - 2q_s q_2 & 2q_2 q_3 + 2q_s q_1 & 2q_s^2 - 1 + 2q_3^2 \end{bmatrix}$$

Just as the conjugate quaternion rotation operator $\tilde{Q}^*(\vec{y})$ transforms the vector $\vec{y}$ into the vector $\vec{x}$**,** the transpose of the transformation matrix $\boldsymbol{T}^T$ performs the same operation:

$$\boldsymbol{T}^T = \begin{bmatrix} t_{11} & t_{21} & t_{31} \\ t_{12} & t_{22} & t_{32} \\ t_{13} & t_{23} & t_{33} \end{bmatrix} = \begin{bmatrix} 2q_s^2 - 1 + 2q_1^2 & 2q_1 q_2 + 2q_s q_3 & 2q_1 q_3 - 2q_s q_2 \\ 2q_1 q_2 - 2q_s q_3 & 2q_s^2 - 1 + 2q_2^2 & 2q_2 q_3 + 2q_s q_1 \\ 2q_1 q_3 + 2q_s q_2 & 2q_2 q_3 - 2q_s q_1 & 2q_s^2 - 1 + 2q_3^2 \end{bmatrix}$$

An inverse relationship can also be derived to compute a quaternion $\tilde{q}$ from a given rotation matrix $\boldsymbol{T}$ as follows:

$$q_s = \left(\frac{1}{2}\right)\sqrt{t_{11} + t_{22} + t_{33} + 1}$$
$$q_1 = (t_{32} - t_{23})/(4q_s)$$
$$q_2 = (t_{13} - t_{31})/(4q_s)$$
$$q_3 = (t_{21} - t_{12})/(4q_s)$$

Note the potential for numerical issues when q_s is near zero (0). There are alternative formulations for computing a quaternion from a transformation matrix that have better numerical behavior [20][21].

E.4. Basic Vector Transformation

The reference frame tree provides the relationship between all reference frames in a SpaceFOM-compliant federation execution. With the exception of the root reference frame, every reference frame is defined relative to a parent frame using the ReferenceFrame object's translational and rotational state (SpaceTimeCoordinateState).

The offset between the origin of the child reference frame and the origin of its parent frame is provided by the position attribute and is a vector expressed in the parent reference frame. The attitude of the reference frame is provided by the attitude_quaternion attribute. This is a quaternion that can be used with the quaternion rotation operator to define the attitude of the child reference frame with respect to the parent reference frame. Together, these can be used to transform a given vector expressed in the child reference frame coordinates into a vector expressed in the parent reference frame coordinates.

This is done for the position of an entity expressed in a given child frame using the following vector and quaternion rotation operator relationships:

$$\vec{\boldsymbol{r}}_{parent} = \vec{\boldsymbol{r}}_{0\_parent} + \widetilde{\boldsymbol{Q}}(\vec{\boldsymbol{r}}_{child})$$

where $\vec{\boldsymbol{r}}_{child}$ is the position vector expressed in the child reference frame, $\widetilde{\boldsymbol{Q}}(\vec{\boldsymbol{r}}_{child})$ is the quaternion rotation operator associated with the attitude quaternion $\widetilde{\boldsymbol{q}}$ that defines the attitude of the child reference frame with respect to the parent reference frame, $\vec{\boldsymbol{r}}_{0\_parent}$ is the vector giving the position of child reference frame origin with respect to the parent reference frame origin expressed in parent reference frame coordinates, and $\vec{\boldsymbol{r}}_{parent}$ is the position vector of the entity expressed in parent reference frame coordinates.

The relative motion between a child reference frame and a parent reference frame is provided by the velocity and angular_velocity attributes. The following equation gives the velocity of an entity expressed in the parent reference frame given the velocity of the entity expressed in the child reference frame:

$$\vec{\boldsymbol{v}}_{parent} = \vec{\boldsymbol{v}}_{0\_parent} + \widetilde{\boldsymbol{Q}}\left(\vec{\boldsymbol{v}}_{child} + (\vec{\boldsymbol{\omega}}_{child} \times \vec{\boldsymbol{r}}_{child})\right)$$

where $\vec{\boldsymbol{v}}_{child}$ is the velocity vector of an entity expressed in the child reference frame, $\vec{\boldsymbol{\omega}}_{child}$ is the angular velocity vector of the child frame with respect to the parent frame and expressed in child frame coordinates, $\vec{\boldsymbol{v}}_{0\_parent}$ is the velocity of the child frame with respect to the parent frame expressed in parent frame coordinates, and $\vec{\boldsymbol{v}}_{parent}$ is the velocity of an entity expressed the parent reference frame.

In most cases, the position and velocity relationships above are sufficient. However, acceleration is sometimes needed and is included here for completeness. The following equation gives the acceleration of an entity expressed in the parent reference frame given the acceleration of the entity expressed in the child reference frame:

$$\vec{\boldsymbol{a}}_{parent} = \vec{\boldsymbol{a}}_{0\_parent} + \widetilde{\boldsymbol{Q}}\left(\vec{\boldsymbol{a}}_{child} + \left(\vec{\boldsymbol{\omega}}_{child} \times (\vec{\boldsymbol{\omega}}_{child} \times \vec{\boldsymbol{r}}_{child})\right) + (2\vec{\boldsymbol{\omega}}_{child} \times \vec{\boldsymbol{v}}_{child}) + (\vec{\boldsymbol{\alpha}}_{child} \times \vec{\boldsymbol{r}}_{child})\right)$$

where $\vec{\boldsymbol{a}}_{child}$ is the acceleration of an entity expressed in the child reference frame, $\vec{\boldsymbol{\alpha}}_{child}$ is the angular acceleration of the child frame with respect to the parent frame and expressed in child frame coordinates, $\vec{\boldsymbol{a}}_{0\_parent}$ is the acceleration of the child frame with respect to the parent frame expressed in parent frame coordinates, and $\vec{\boldsymbol{a}}_{parent}$ is the acceleration of an entity expressed in the parent reference frame.

E.4.1. Special Subcases

There are two interesting subcases to the general vector transformations expressed above. The first is the case when the child frame is co-located with the parent frame. This implies the following:

$$\vec{\boldsymbol{r}}_{0\_parent} = 0$$

$$\vec{\boldsymbol{v}}_{0\_parent} = 0$$

$$\vec{\boldsymbol{a}}_{0\_parent} = 0$$

This means that the parent and child reference frames share the same origin but their coordinate axes rotate with respect to one another. In other words, the relationship between the state of an entity expressed in the child frame and the state of an entity expressed in the parent frame is purely rotational:

$$\vec{\boldsymbol{r}}_{parent} = \widetilde{\boldsymbol{Q}}(\vec{\boldsymbol{r}}_{child})$$

$$\vec{\boldsymbol{v}}_{parent} = \widetilde{\boldsymbol{Q}}\left(\vec{\boldsymbol{v}}_{child} + (\vec{\boldsymbol{\omega}}_{child} \times \vec{\boldsymbol{r}}_{child})\right)$$

$$\vec{\boldsymbol{a}}_{parent} = \widetilde{\boldsymbol{Q}}\left(\vec{\boldsymbol{a}}_{child} + \left(\vec{\boldsymbol{\omega}}_{child} \times (\vec{\boldsymbol{\omega}}_{child} \times \vec{\boldsymbol{r}}_{child})\right) + (2\vec{\boldsymbol{\omega}}_{child} \times \vec{\boldsymbol{v}}_{child}) + (\vec{\boldsymbol{\alpha}}_{child} \times \vec{\boldsymbol{r}}_{child})\right)$$

The second interesting case is when the child and parent reference frames are co-aligned. This implies that the quaternion rotation θ is zero. This results in an identity quaternion and the following relationship:

$$\widetilde{\boldsymbol{Q}}(\vec{\boldsymbol{r}}_{child}) = \vec{\boldsymbol{r}}_{child}$$

This also implies angular velocity and angular acceleration are zero:

$$\vec{\boldsymbol{\omega}}_{child} = 0$$

$$\vec{\boldsymbol{\alpha}}_{child} = 0$$

This means that the parent and child reference frames coordinate axes are aligned but they do not share the same origin. In other words, the relationship between the state of an entity expressed in the child frame and the state of an entity expressed in the parent frame is purely translational:

$$\vec{\boldsymbol{r}}_{parent} = \vec{\boldsymbol{r}}_{0\_parent} + \vec{\boldsymbol{r}}_{child}$$

$$\vec{\boldsymbol{v}}_{parent} = \vec{\boldsymbol{v}}_{0\_parent} + \vec{\boldsymbol{v}}_{child}$$

$$\vec{\boldsymbol{a}}_{parent} = \vec{\boldsymbol{a}}_{0\_parent} + \vec{\boldsymbol{a}}_{child}$$

These two special cases will come up frequently for commonly used celestial reference frames.

E.4.2. Reverse Transformations

What happens if we are interested in the reverse transformation; specifically, transforming a position vector expressed in a parent frame into a position vector expressed in the child frame? Using the child to parent vector transformation equation above along with some vector and quaternion algebra, results in the following reverse transformation:

$$\vec{\boldsymbol{r}}_{child} = \widetilde{\boldsymbol{Q}}^{*}\left(\vec{\boldsymbol{r}}_{parent} - \vec{\boldsymbol{r}}_{0\_parent}\right) = -\vec{\boldsymbol{r}}_{0\_child} + \widetilde{\boldsymbol{Q}}^{*}\left(\vec{\boldsymbol{r}}_{parent}\right)$$

where $\widetilde{\boldsymbol{Q}}^{*}\left(\vec{\boldsymbol{r}}_{parent}\right)$ is the conjugate quaternion rotation operator associated with the attitude quaternion $\widetilde{\boldsymbol{q}}$ that defines the attitude of the child reference frame with respect to the parent reference frame, and $\vec{\boldsymbol{r}}_{0\_child}$ is the vector giving the position of child reference frame origin with respect to the parent reference frame origin expressed in child reference frame coordinates.

Similar relationships can be derived for velocity and acceleration:

$$\vec{\boldsymbol{v}}_{child} = \widetilde{\boldsymbol{Q}}^{*}\left(\vec{\boldsymbol{v}}_{parent} - \vec{\boldsymbol{v}}_{0\_parent}\right) - (\vec{\boldsymbol{\omega}}_{child} \times \vec{\boldsymbol{r}}_{child})$$

$$= -\vec{\boldsymbol{v}}_{0\_child} - (\vec{\boldsymbol{\omega}}_{child} \times \vec{\boldsymbol{r}}_{child}) + \tilde{\boldsymbol{Q}}^{*}(\vec{\boldsymbol{v}}_{parent})$$

$$\vec{\boldsymbol{a}}_{child} = \tilde{\boldsymbol{Q}}^{*}(\vec{\boldsymbol{a}}_{parent} - \vec{\boldsymbol{a}}_{0\_parent}) - (\vec{\boldsymbol{\omega}}_{child} \times (\vec{\boldsymbol{\omega}}_{child} \times \vec{\boldsymbol{r}}_{child})) - (2\vec{\boldsymbol{\omega}}_{child} \times \vec{\boldsymbol{v}}_{child}) - (\vec{\boldsymbol{\alpha}}_{child} \times \vec{\boldsymbol{r}}_{child})$$
$$= -\vec{\boldsymbol{a}}_{0\_child} - (\vec{\boldsymbol{\omega}}_{child} \times (\vec{\boldsymbol{\omega}}_{child} \times \vec{\boldsymbol{r}}_{child})) - (2\vec{\boldsymbol{\omega}}_{child} \times \vec{\boldsymbol{v}}_{child}) - (\vec{\boldsymbol{\alpha}}_{child} \times \vec{\boldsymbol{r}}_{child}) + \tilde{\boldsymbol{Q}}^{*}(\vec{\boldsymbol{a}}_{parent})$$

E.5. Sequential Reference Frame Tree Transformations

At this point in the discussion, a simple example might be helpful. We can start with the very simple reference frame tree composed of reference frames P, A1, A2, B1 and B2 (see Figure E-1). P is the root frame and parent to both A1 and A2. Reference frame A1 is the parent of B1 and reference frame A2 is the parent of B2. In addition, frames A1 and B1 are co-located but not co-aligned, as are A2 and B2. Frames P, A1 and A2 are co-aligned but not co-located. Finally, the position of an entity is given by a vector $\vec{\boldsymbol{r}}$ expressed in frame B1, denoted $\vec{\boldsymbol{r}}_{B1}$.

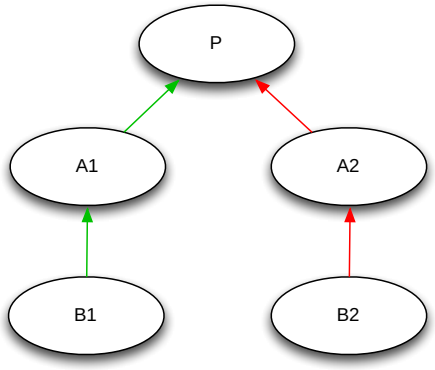

Figure E-1: Simple Reference Frame Tree Traversal

For this example, we are interested in determining the position of the entity expressed in the B2 frame instead of the B1 frame. Specifically, given the position vector of an entity expressed in reference frame B1 ($\vec{\boldsymbol{r}}_{B1}$), compute the position of the entity expressed in reference frame B2 ($\vec{\boldsymbol{r}}_{B2}$).

The first step is to establish the pathway between reference frames B2 and B1. The green and red arrows in Figure E-1 represent this pathway. Frame P is the closest common reference frame in the tree. The direct path from B1 to B2 goes through frame P. This gives the following chain of reference frames: B1 ➔ A1 ➔ P ➔ A2 ➔ B2. Note that the arrows between B2, A2, and P are red. This indicates that the desired transformation direction is reversed.

Let's start by determining the entity position expressed in frame A1 ($\vec{\boldsymbol{r}}_{A1}$). Since A1 and B1 are co-located, the relationship is a pure rotation ($\vec{\boldsymbol{r}}_{0\_B1} = 0$):

$$\vec{\boldsymbol{r}}_{A1} = \vec{\boldsymbol{r}}_{0\_B1} + \tilde{\boldsymbol{Q}}_{\boldsymbol{B1\rightarrow A1}}(\vec{\boldsymbol{r}}_{B1}) = \tilde{\boldsymbol{Q}}_{\boldsymbol{B1\rightarrow A1}}(\vec{\boldsymbol{r}}_{B1})$$

Now compute the entity position expressed in frame P ($\vec{\boldsymbol{r}}_{P}$). Since P and A1 are co-aligned the relationship is a pure translation ($\tilde{\boldsymbol{Q}}_{\boldsymbol{A1\rightarrow P}}(\vec{\boldsymbol{r}}_{A1}) = \vec{\boldsymbol{r}}_{A1}$):

$$\vec{\boldsymbol{r}}_{P} = \vec{\boldsymbol{r}}_{0\_A1} + \tilde{\boldsymbol{Q}}_{\boldsymbol{A1\rightarrow P}}(\vec{\boldsymbol{r}}_{A1}) = \vec{\boldsymbol{r}}_{0\_A1} + \vec{\boldsymbol{r}}_{A1}$$

The next step is to compute the entity position expressed in frame A2 ($\vec{\boldsymbol{r}}_{A2}$). Since P and A2 are co-aligned the relationship is also a pure translation ($\tilde{\boldsymbol{Q}}_{\boldsymbol{A2\rightarrow P}}(\vec{\boldsymbol{r}}) = \vec{\boldsymbol{r}}$). However, the available transformation information is for A2 expressed in frame P. So, we need to use the reverse transformation formula:

$$\vec{\boldsymbol{r}}_{A2} = \tilde{\boldsymbol{Q}}^{*}_{\boldsymbol{A2\rightarrow P}}(\vec{\boldsymbol{r}}_{P} - \vec{\boldsymbol{r}}_{0\_A2}) = \vec{\boldsymbol{r}}_{P} - \vec{\boldsymbol{r}}_{0\_A2}$$

The final step is to transform from the A2 frame into the B2 frame. Since A2 and B2 are co-located the relationship is a pure rotation ($\vec{r}_{0\_B2} = 0$). However, the available transformation information is for B2 expressed in frame A2. So, we need to use the reverse transformation formula:

$$\vec{r}_{B2} = \tilde{Q}^{*}_{B2 \to A2}\left(\vec{r}_{A2} - \vec{r}_{0\_B2}\right) = \tilde{Q}^{*}_{B2 \to A2}(\vec{r}_{A2})$$

This completes the transformation of the entity position expressed in frame B1 into and entity position expressed in frame B2.

So, how is this useful for SpaceFOM data?

Let's replace the names in the reference frame tree in Figure E-1 with common celestial corollaries.

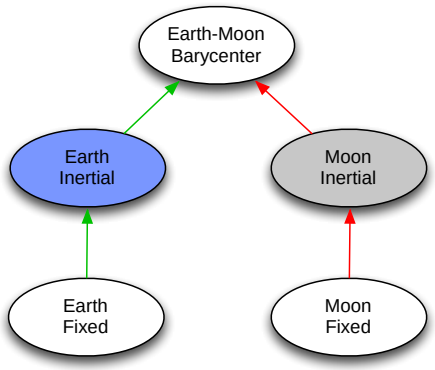

Figure E-2: Simple Earth-Moon Tree

Frame P becomes the Earth-Moon Barycenter frame (EMB), A1 becomes the Earth inertial frame (ECI), B1 becomes Earth fixed (ECF), A2 becomes Moon inertial (MCI), and B2 becomes Moon fixed (MCF). By making these substitutions, we get the following transformation sequence to go from ECF to MCF:

$$\vec{r}_{ECI} = \tilde{Q}_{ECF \to ECI}(\vec{r}_{ECF})$$
$$\vec{r}_{EMB} = \vec{r}_{0\_ECI} + \vec{r}_{ECI}$$
$$\vec{r}_{MCF} = \tilde{Q}^{*}_{MCF \to MCI}(\vec{r}_{MCI})$$
$$\vec{r}_{MCI} = \vec{r}_{EMB} - \vec{r}_{0\_MCI}$$

Using the reference frame transformation information in the SpaceFOM ReferenceFrame objects, we can compute the position of a communications site on the surface of the Earth with respect to Moon fixed coordinates. This will be useful in determining distance and line-of-sight computations for Earth-Moon communications.

A similar process can be done to derive the relationships for velocity:

$$\vec{v}_{ECI} = \tilde{Q}_{ECF \to ECI}\left(\vec{v}_{ECF} + (\vec{\omega}_{Earth} \times \vec{r}_{ECF})\right)$$
$$\vec{v}_{EMB} = \vec{v}_{0\_ECI} + \vec{v}_{ECI}$$
$$\vec{v}_{MCI} = \vec{v}_{EMB} - \vec{v}_{0\_MCI}$$
$$\vec{v}_{MCF} = \tilde{Q}^{*}_{MCF \to MCI}(\vec{v}_{MCI}) - (\vec{\omega}_{Moon} \times \vec{r}_{MCF})$$

where $\vec{\omega}_{Earth}$ is the angular velocity of the Earth expressed in the Earth fixed frame and $\vec{\omega}_{Moon}$ is the angular velocity of the Moon expressed in the Moon fixed frame.

Appendix F. [Informative] Standard Reference Frames and Axis

This appendix provides more detail on SpaceFOM-compliant standard (a.k.a., well known) reference frames and axis types.

F.1. Standard Reference Frames

General naming conventions are covered in section 5.3 but do not indicate specific reference frame names. In fact, the SpaceFOM does not predicate the use of any specific reference frames, it only prescribes that the reference frames comply with the reference frame tree structure specified in section 5.2 and comply with the reference frame naming convention specified in section 5.3. A SpaceFOM-compliant federation execution can create a reference frame tree and associated reference frames to satisfy its needs.

However, most space focused simulation occurs within our Sun-centered solar system. With that in mind, there are a number of standard or well know reference frames that can be useful as a basis of commonality between SpaceFOM federation executions. These are based on the Simulation Exploration Experience (SEE) and General Mission Analysis Tool (GMAT) projects and listed in Table F-1 along with their associated description and additional information in [10] and [11].

Name	Origin	Axes	Description	Additional Information
EarthMJ2000Eq	Earth	MJ2000Eq	An Earth mean-of-date equator inertial system defined at epoch J2000 and based on IAU-1976/FK5 theory with 1980 update to nutation.	A suggested parent Reference Frame is: SolarSystemBarycentricInertial
EarthMJ2000Ec	Earth	MJ2000Ec	An Earth mean-of-date ecliptic inertial system defined at epoch J2000 and based on IAU-1976/FK5 theory with 1980 update to nutation.	A suggested parent Reference Frame is: EarthMJ2000Eq
EarthFixed	Earth	ITRF	An Earth fixed system based on an International Terrestrial Reference Frame (ITRF) solution.	The Earth fixed system is used by the gravity model for full field modeling. A suggested parent Reference Frame is: EarthMJ2000Eq The International Earth Rotation and Reference Systems Service (IERS) maintains and updates ITRF solutions, releasing a new solution every few years. The federation agreement must identify which ITRF solution is used to define the EarthFixed frame.
EarthICRF	Earth	ICRF	An Earth mean-of-date equator inertial system defined at epoch J2000 and based on IAU-2000 theory with 2006 update to precession.	A suggested parent Reference Frame is: SolarSystemBarycentricInertial To provide a consistent transition to ICRF, the IAU defined ICRF to align with MJ2000Eq to the limits of the latter frame's precision. The more precise ICRF differs from MJ2000Eq by less than 0.1 arcseconds (~0.5 µrad). As a consequence, this reference frame is nominally based on ICRF but it can be substituted by MJ2000Eq if specified in the Federation Agreement.

Name	Origin	Axes	Description	Additional Information
*(CelestialBody)*Fixed [28] [29]	Celestial bodies other than Earth, Sun, and Moon	BodyFixed	The BodyFixed axis system is referenced to the body equator and the prime meridian of the body with the z-axis pointing North.	Fixed systems used by the gravity model for full field modeling at other bodies. A Celestial body's equator and prime meridian are defined by the IAU (see the "Report of the IAU Working Group on Cartographic Coordinates and Rotational Elements: 2009", DOI 10.1007/s10569-010-9320-4).
SolarSystemBarycentricInertial	Barycenter of the Solar System	ICRF	A pseudo-inertial reference frame that is positioned at the barycenter of the Solar System. It is the International Celestial Reference Frame (ICRF) adopted by International Astronomical Union (IAU).	The recommended *root* Reference Frame in the *reference frame tree* for general purpose space exploration simulations.
SunCentricInertial	Sun	BodyInertial	A pseudo-inertial reference frame that is positioned at the center of the Sun. This reference frame is a mean-of-date equatorial inertial system whose x-axis is the intersection of the mean-of-date equator with the mean-of-date ecliptic plane at epoch J2000. It is sometimes called Heliographic Coordinates (HGI) or Heliocentric Inertial Coordinates.	A suggested parent Reference Frame is: SolarSystemBarycentricInertial
MoonCentricInertial	Moon	BodyInertial	A pseudo-inertial reference frame that is positioned at the center of the Moon. This reference frame is a translation of the EarthICRF or EarthMJ2000Eq reference frame to the Moon's center.	A suggested parent Reference Frame is: EarthICRF or EarthMJ2000Eq

[28] See the EBNF rule for CelestialBody in Section 5.3.
[29] Please note that there is not a common standard on the definition of a generic *CelestialBodyInertial* reference frames; thus such a type of reference frames should be defined in the FESFA.

Name	Origin	Axes	Description	Additional Information
MoonCentricFixed	Moon	BodyFixed	Mean-Earth and Polar-Axis (aka Mean Earth and Rotation Axis). The x-axis is pointed in the mean direction of the Earth and marks the prime meridian of the Moon. The z-axis is along the axis of rotation. The y-axis completes the right-handed coordinate system.	A suggested parent Reference Frame is: MoonCentricInertial
MarsInertial	Mars	BodyInertial	Mars Mean Equator and IAU vector of J2000. The IAU-vector at Mars is the point on the mean equator of Mars where the equator ascends through the earth mean equator. This vector is the cross product of Earth mean north with Mars mean north.	A suggested parent Reference Frame is: SolarSystemBarycentricInertial

Table F-1: Well Known Reference Frames

F.2. A Reference Frame Tree example

At this point, it might be helpful to provide an example reference frame tree using well known reference frames from Table F-1 that is SpaceFOM-compliant in both structure and naming convention. This representative reference frame tree consists of the following reference frames:

I. SolarSystemBarycentricInertial
   A. SunCentricInertial
   B. EarthMoonBarycentricInertial
      1. EarthMoonBarycentricRotating
      2. EarthMoon L2Rotating
   C. EarthMJ2000Eq
      1. EarthFixed
      2. EarthMJ2000Ec
      3. MoonCentricInertial
         a) MoonCentricFixed
   D. MarsCentricInertial
      1. MarsCentricFixed

Note that the entries in black are well known reference frames from Table F-1 and the entries in red are additional reference frames.

While the additional reference frames are not “well know” reference frames, they do comply with the naming convention established in section 5.3. A graphical representation of this reference frame tree is shown in Figure F-1.

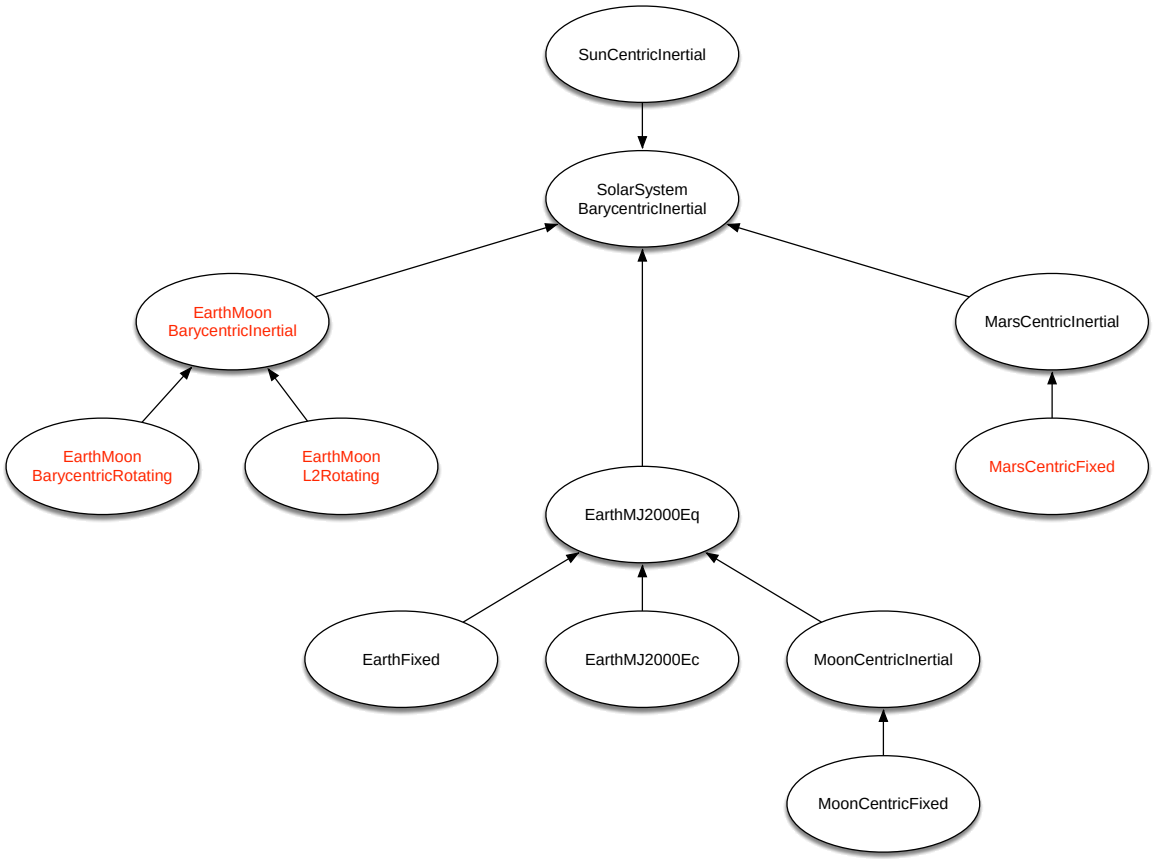

Figure F-1: An Example Know Reference Frame Tree

Observe that this reference frame's organizational structure is a SpaceFOM-compliant *reference frame tree* (see Section 5.2.2). It has only one *root reference frame*, a reference frame with no parent (e.g., SolarSystemBarycentricInertial). Each reference frame, with the exception of the *root reference frame*, has a single parent (e.g.,MoonCentricFixed). Also, some reference frames are parents to more than one child frame (e.g., EarthMJ200Eq).

Note that specialized reference frame trees can be constructed by "pruning" the reference frame tree above. For example, a SpaceFOM-compliant federation execution that is only concerned with modeling activities within the Earth-Moon system might choose to reduce the reference frame tree as shown in Figure F-2.

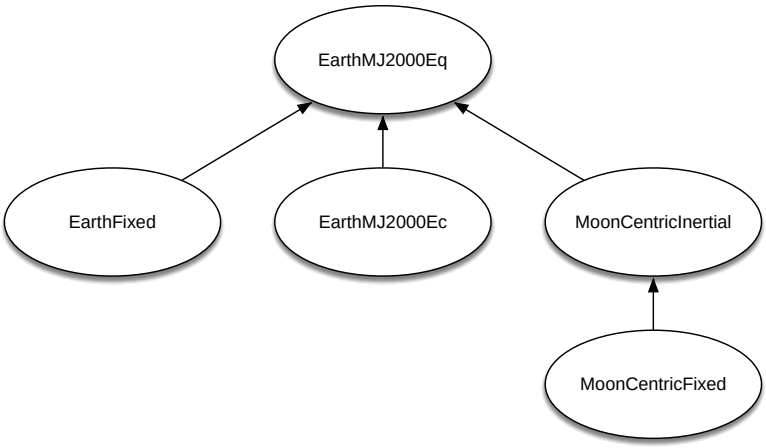

Figure F-2: Reduced Earth-Moon Reference Frame Tree

F.3. Standard Axis Types

Note that the naming conventions defined in 5.3 and the well-known reference frames in Table F-1 both require an Axes specification. Description of standard Axes Types are listed in Table F-2 and were extracted from [12].

Axes Name	Origin Limitations	Base Type	Description
MJ2000Eq	None	IAU-1976 FK5	An inertial coordinate system. The nominal x-axis points along the line formed by the intersection of the Earth's mean equatorial plane and the mean ecliptic plane (at the J2000 epoch), in the direction of Aries. The z-axis is normal to the Earth's mean equator at the J2000 epoch and the y-axis completes the right-handed system. The mean planes of the ecliptic and equator, at the J2000 epoch, are computed using IAU-1976/FK5 theory with 1980 update for nutation.
MJ2000Ec	None	IAU-1976 FK5	An inertial coordinate system. The x-axis points along the line formed by the intersection of the Earth's mean equator and the mean ecliptic plane at the J2000 epoch. The z-axis is normal to the mean equatorial plane at the J2000 Epoch and the y-axis completes the right-handed set. This system is computed using IAU-1976/FK5 theory with 1980 update for nutation.
ICRF	None	IAU-2000	An inertial coordinate system. The axes are close to the mean Earth equator and pole at the J2000 epoch, and at the Earth's surface, the RSS difference between vectors expressed in MJ2000Eq and ICRF is less than 1 m. Note that since MJ2000Eq and ICRF are imperfect realizations of inertial systems, the transformation between them is time varying. This axis system is computed using IAU-2000A theory with 2006 update for precession.
ITRF	None		An Earth-fixed system. This system is computed using IAU-2000A theory with 2006 update for precession.
MODEq	None	IAU-1976 FK5	A quasi-inertial coordinate system referenced to Earth's mean equator at the current epoch. The current epoch is defined by the context of use and should be documented in the FESFA (see Rule 4-3 for the definition of the Federation Scenario Time epoch). This system is computed using IAU-1976/FK5 theory with 1980 update for nutation.
MODEc	None	IAU-1976 FK5	A quasi-inertial coordinate system referenced to the mean ecliptic at the current epoch. The current epoch is defined by the context of use and should be documented in the FESFA (see Rule 4-3 for the definition of the Federation Scenario Time epoch). This system is computed using IAU-1976/FK5 theory with 1980 update for nutation.
TODEq	None	IAU-1976 FK5	A quasi-inertial coordinate system referenced to Earth's true equator at the current epoch. The current epoch is defined by the context of use and should be documented in the FESFA (see Rule 4-3 for the definition of the Federation Scenario Time epoch). This system is computed using IAU-1976/FK5 theory with 1980 update for nutation.
TODEc	None	IAU-1976 FK5	A quasi-inertial coordinate system referenced to Earth's true ecliptic at the current epoch. The current epoch is defined by the context of use and should be documented in the FESFA (see Rule 4-3 for the definition of the Federation Scenario Time epoch). This system is computed using IAU-1976/FK5 theory with 1980 update for nutation.

Axes Name	Origin Limitations	Base Type	Description
MOEEq	None	IAU-1976 FK5	A quasi-inertial coordinate system referenced to Earth's mean equator at the reference epoch (see Rule 4-3 for the definition of the Federation Scenario Time epoch). This system is computed using IAU-1976/FK5 theory with 1980 update for nutation.
MOEEc	None	IAU-1976 FK5	A quasi-inertial coordinate system referenced to the mean ecliptic at the reference epoch (see Rule 4-3 for the definition of the Federation Scenario Time epoch). This system is computed using IAU-1976/FK5 theory with 1980 update for nutation.
TOEEq	None	IAU-1976 FK5	A quasi-inertial coordinate system referenced to Earth's true equator at the reference epoch (see Rule 4-3 for the definition of the Federation Scenario Time epoch). This system is computed using IAU-1976/FK5 theory with 1980 update for nutation.
TOEEc	None	IAU-1976 FK5	A quasi-inertial coordinate system referenced to the true ecliptic at the reference epoch (see Rule 4-3 for the definition of the Federation Scenario Time epoch). This system is computed using IAU-1976/FK5 theory with 1980 update for nutation.
Equator	Celestial Body		A true of date equator axis system for the celestial body selected as the origin. The Equator system is defined by the body's equatorial plane and its intersection with the ecliptic plane, at the current epoch. The current epoch is defined by the context of use and should be defined in the FESFA.
BodyFixed	Celestial Body		The BodyFixed axis system is referenced to the body equator and the prime meridian of the body.
BodyInertial	Celestial Body		An inertial system referenced to the equator (at the J2000 epoch) of the celestial body selected as the origin of the coordinate system. Because the accuracy of available models varies for different solar system bodies, the BodyInertial axis system uses different theories for different celestial bodies. For Earth, the BodyInertial axis system is identical to the MJ2000Eq system. For all other celestial bodies, the BodyInertial axis system is based upon the IAU report entitled "Report of the IAU/IAG Working Group on Cartographic Coordinates and Rotational Elements of the Planets and Satellites: 2000". Because the BodyInertial axis system uses different theories for different bodies, the following definitions describe only the nominal axis configurations. The x-axis points along the line formed by the intersection of the body's equator and earth's mean equator at J2000. The z-axis points along the body's spin axis direction at the J2000 epoch. The y-axis completes the right-handed set.
GSE	None		The Geocentric Solar Ecliptic system. The x-axis points from Earth to the Sun. The z-axis is defined as the cross product RxV of the earth's position and velocity with respect to the sun. The y-axis completes the right-handed set. The GSE axes are computed using the relative motion of the Earth and Sun even if the origin is not Earth.

Axes Name	Origin Limitations	Base Type	Description
GSM	None		The Geocentric Solar Magnetic system. The x-axis points from Earth to the Sun. The z-axis is defined to be orthogonal to the x-axis and lies in the plane of the x-axis and Earth's magnetic dipole vector. The y-axis completes the right-handed set. The GSM axes are computed using the relative motion of the Earth and Sun even if the origin is not Earth.
Topocentric [30]	Celestial Body		The Topocentric coordinate system is a ground-station based coordinate system. The y-axis points to East and the z-axis is normal to the local horizon. The x-axis completes the right handed set.
BodySpinSun	Celestial Body		The BodySpinSun is a celestial body spin-axis-referenced system. The x-axis points from the celestial body to the Sun. The y-axis is computed as the cross product of the x-axis and the body's spin axis. The z-axis completes the right-handed set.
Topodetic	Celestial Body or Solar System Reference in proximity to a Celestial Body		The topodetic coordinate system is a coordinate system defined relative to a celestial body. One axis points from the origin to the nadir, one axis points East, and the third axis points North. The origin is often a human-made body operating in proximity to a celestial body; however, it could also be a point fixed to the surface of a celestial body.

Table F-2: Standard Axes Types

[30] Space communities use different definitions of 'topocentric.' The one here is based on celestial observation.

Appendix G. [Informative] Rules Versus Roles Mappings

This appendix presents a general mapping matrix between the SpaceFOM Rules defined in this document and their association with the principal SpaceFOM documents (FESFA and FCD) and principal SpaceFOM federate roles described in this document. The rules are listed in the left column and the documents/roles are listed in the top heading rows. An association between a specific rule and a specific document/role is indicated if the corresponding row/column cell is marked with an **X**.

These association are general in the sense that there exists some relevant relationship between the given rule and the given document/role. In electronic versions of this document, the rules should be linked back to the original definition of the rule in preceding SpaceFOM text. The association should be apparent when reviewing the rule in context with the associated document/role.

Rule	FESFA	FCD	Federates					
			Master	Pacing	RRFP	Early Joiner	Late Joiner	Any
Rule 1-1: Federation Execution Compliance with HLA Rules	X		X	X	X	X	X	X
Rule 1-2: Federate Compliance with HLA Rules		X	X	X	X	X	X	X
Rule 1-3: Federate Compliance with the HLA OMT Specification.			X	X	X	X	X	X
Rule 1-4: Do Not Modify SpaceFOM Modules	X	X						
Rule 1-5: Extend SpaceFOM With Added FOM Modules	X	X						
Rule 3-1: The Federation Execution Specific Federation Agreement (FESFA)	X							
Rule 3-2: The Federate Compliance Declaration (FCD)		X	X	X	X	X	X	X
Rule 3-3: Documentation of Required Federates in FESFA	X		X	X	X	X		
Rule 3-4: Specification of Federate Role Capabilities in FCD		X	X	X	X	X	X	X
Rule 3-5: Specify Master Federate in FESFA	X		X					
Rule 3-6: Specify Pacing Federate in FESFA	X			X				
Rule 3-7: Specify Root Reference Frame Publishing Federate in FESFA	X				X			
Rule 3-8: Only the Master Federate Creates and Publishes the ExCO			X					
Rule 3-9: Only the Master Federate Manages Mode Transitions			X					
Rule 3-10: Only One Master Federate in a Federation Execution	X		X					

Rule	FESFA	FCD	Federates					
			Master	Pacing	RRFP	Early Joiner	Late Joiner	Any
Rule 3-11: The Pacing Federate Regulates HLA Time Management				X				
Rule 3-12: Only One Pacing Federate in a Federation Execution	X			X				
Rule 3-13: Only One Root Reference Frame Publisher Federate in a Federation Execution	X				X			
Rule 3-14: Use of Common Data Types			X	X	X	X	X	X
Rule 4-1: HLA Logical Time Starts at Zero (0)			X	X	X	X	X	X
Rule 4-2: Use HLA Logical Time and Federation Scenario Time Epoch to Coordinate Time Lines			X	X	X	X	X	X
Rule 4-3: Specify Federation Scenario Time Epoch in FESFA	X		X					
Rule 4-4: Document Federates Time Frame Dependencies in FCD		X	X	X	X	X	X	X
Rule 4-5: Fixed Federation Scenario Time Epoch			X					
Rule 4-6: Master Federate Publishes Federation Simulation Time Epoch	X		X					
Rule 4-7: Use Federation Scenario Time Epoch to Coordinate Simulation Scenario Time			X	X	X	X	X	X
Rule 4-8: Use of Time Stamps when Exchanging Data			X	X	X	X	X	X
Rule 4-9: Time Stamp Format			X	X	X	X	X	X
Rule 4-10: Scenario Time Expressed in the Terrestrial Time (TT) Scale	X		X	X	X	X	X	X
Rule 4-11: Federate Declaration of Supported Time Management Types in FCD		X	X	X	X	X	X	X
Rule 4-12: Federation Execution Specification of Supported Time Management Type in FESFA	X			X				
Rule 4-13: Federation Execution Contains Only Federates That Support Time Management Types	X	X	X	X	X	X	X	X
Rule 4-14: Definition of the Federation Time Step in FESFA	X			X				
Rule 4-15: Pacing Federate Uses Federation Time Step as Federate Time Step				X				
Rule 4-16: Constant Time Steps				X	X	X	X	X

Rule	FESFA	FCD	Federates					
			Master	Pacing	RRFP	Early Joiner	Late Joiner	Any
Rule 4-17: Definition and Publication of Least Common Time Step (LCTS)	X		X					
Rule 4-18: Relationship Between Federate Time Step and Simulation Time Step			X	X	X	X	X	X
Rule 4-19: Relationship Between Federate Time Step and Federation Time Step			X	X	X	X	X	X
Rule 4-20: Federate Time Management Lookahead Matches Federate Time Step			X	X	X	X	X	X
Rule 4-21: Synchronized Federates Use HLA Time Management TAR and TAG Services			X	X	X	X	X	X
Rule 4-22: Time Regulating Pacing Federate				X				
Rule 4-23: Time Constrained Pacing Federate				X				
Rule 4-24: Publication of Time Stamp Ordered Data			X	X	X	X	X	X
Rule 4-25: Subscription to Time Stamp Ordered Data			X	X	X	X	X	X
Rule 4-26: Time Stamp Order Delivery for Updates and Interactions			X	X	X	X	X	X
Rule 4-27: Receive Order Delivery for Updates and Interactions			X	X	X	X	X	X
Rule 4-28: All Federates Enable Asynchronous Delivery			X	X	X	X	X	X
Rule 4-29: When to Call Time Advance Request			X	X	X	X	X	X
Rule 5-1: Unique ReferenceFrame Names.					X	X	X	X
Rule 5-2: Root ReferenceFrame Parent Name					X			
Rule 5-3: Only One Root Reference Frame					X			
Rule 5-4: Fixed Root Reference Frame					X			
Rule 5-5: ReferenceFrame Parent Name Existence					X	X	X	X
Rule 5-6: All Parent ReferenceFrames Must Exist					X	X	X	X
Rule 5-7: Document All Federation Reference Frames in FESFA	X							
Rule 5-8: Coherent ReferenceFrame Updates					X	X	X	X
Rule 6-1: Unique PhysicalEntity Names						X	X	X

Rule	FESFA	FCD	Federates					
			Master	Pacing	RRFP	Early Joiner	Late Joiner	Any
Rule 6-2: PhysicalEntity Types Described in FESFA	X							
Rule 6-3: PhysicalEntity Type Set at Initialization						X	X	X
Rule 6-4: PhysicalEntity Status Defined and Described in FESFA	X							
Rule 6-5: Handling Unrecognized PhysicalEntity Status Values						X	X	X
Rule 6-6: Existence of PhysicalEntity Parent Reference Frame						X	X	X
Rule 6-7: Default PhysicalEntity Body Frame Specification						X	X	X
Rule 6-8: Coherent PhysicalEntity Updates						X	X	X
Rule 6-9: Specifying DynamicalEntity Force and Acceleration Together						X	X	X
Rule 6-10: Specifying DynamicalEntity Torque and Rotational Acceleration Together						X	X	X
Rule 6-11: Coherent DynamicalEntity Updates						X	X	X
Rule 6-12: Unique PhysicalInterface Names						X	X	X
Rule 6-13: PhysicalInterface Naming Convention Specified in the FESFA	X							
Rule 6-14: Existence of PhysicalInterface Parent Object Instance						X	X	X
Rule 6-15: Coherent PhysicalInterface Updates						X	X	X
Rule 7-1: Document CTE Standard in FESFA	X							
Rule 7-2: Document Federate CTE Capabilities in FCD		X						
Rule 7-3: Use Compatible CTE Standard and Time Representation				X		X	X	X
Rule 7-4: Master and Pacing Federates Use CTE If Any Federate Requires CTE			X	X				
Rule 7-5: Master Federate Disables and Restores Auto-Provide During Initialization			X					
Rule 7-6: Required Federates Must Register with Unique Names			X	X	X	X		
Rule 7-7: Document Required Federate Names In FESFA	X		X	X	X	X		

Rule	FESFA	FCD	Federates					
			Master	Pacing	RRFP	Early Joiner	Late Joiner	Any
Rule 7-8: Master Federate Discovers Required Federates By Name			X					
Rule 7-9: Required Object Instances Must Have Unique Names			X	X	X	X	X	X
Rule 7-10: Required Object Instances Discovered By Reserved Object Instance Name			X	X	X	X	X	X
Rule 7-11: Document Reserved Object Instance Names In FESFA	X							
Rule 7-12: Document Reserved Object Instance Names in FCD		X						
Rule 7-13: Document Multiphase Initialization Process In FESFA.	X							
Rule 7-14: Document Multiphase Initialization Synchronization Points in FESFA	X							
Rule 7-15: Master Federate Creates Named Multiphase Initialization Synchronization Points			X					
Rule 7-16: No ExCO Updates From Master Federate Prior To Initialization Object Discovery			X					
Rule 7-17: Check for ExCO Mode Transitions in Wait Loops After Initialization Object Discovery			X	X	X	X	X	X
Rule 7-18: Master Federate Can Only Mode Transition To Shutdown Prior To Initialization Complete			X					
Rule 7-19: Master Federate Never Achieves “initialization_completed” Synchronization Point			X					
Rule 7-20: Federates Do Not Achieve “mtr_shutdown” Synchronization Point			X	X	X	X	X	X
Rule 7-21: Only Master Federate Processes Mode Transition Requests.			X					
Rule 7-22: All Federates Honor Mode Transitions.				X	X	X	X	X
Rule 7-23: Federates Start On Common HLA Logical Time Boundaries (HLTBs)							X	X
Rule 7-24: Federation Execution Freezes On Common HLA Logical Time Boundaries.	X	X	X	X	X	X	X	X
Rule 7-25: Document Master Federate Freeze Policy in FESFA.	X		X					

Rule	FESFA	FCD	Federates					
			Master	Pacing	RRFP	Early Joiner	Late Joiner	Any
Rule 8-1: Achieve Unknown Synchronization Points			X	X	X	X	X	X
Rule 8-2: Reference Frame Latency Compensation						X	X	X
Rule 8-3: Use of Nominal Switches Table Settings	X		X					
Rule 8-4: Auto-Provide Disabled By Default			X					
Rule 8-5: Document Auto-Provide Setting In FESFA	X							
Rule 8-6: Federates Implement Provide Attribute Value Update Service Interface			X	X	X	X	X	X
Rule 8-7: Late Joiner Federates Request Needed Attribute Updates							X	

Table G-1: Rules Versus Roles Mappings

Appendix H. [Normative] The Normative Nature of SpaceFOM Figures

There are numerous Rules in this SpaceFOM document, all of which are considered normative specifications. However, there are also a number of figures in the SpaceFOM that provide supportive information to the textual content and description. Some of these figures are provided as illustrative or clarifying information to the reader. However, some of the figures provide normative specification for relationships and processes. Some are both normative and informative.

This appendix provides the following table to help the reader distinguish between the normative and informative nature of the figures in the SpaceFOM.

Figure	Normative	Informative	Comment
Figure 1-1: Main Concepts of the HLA Standard		X	
Figure 1-2: Adding Extensions to the SpaceFOM		X	
Figure 4-1: Bidirectional Infinite Time Line		X	
Figure 4-2: Unidirectional Infinite Time Line		X	
Figure 4-3: Finite Time Line		X	
Figure 4-4: Relationships Between Time Lines	X	X	Generally informative but provides normative direction on time line associations.
Figure 4-5: Time Lines, Epochs, and Time Stamps Example		X	
Figure 4-6: Time Steps in a Federation		X	
Figure 4-7: Sample Use of Time Management		X	
Figure 5-1: Directed Rooted Labeled Tree	X	X	Generally informative but provides normative directions on reference frame tree topology.
Figure 5-2: Traversing A Tree	X	X	Generally informative but provides normative directions on reference frame tree traversal.
Figure 5-3: Example Reference Frame Tree		X	
Figure 7-1: Simplified SpaceFOM Executive Flow		X	

Figure	Normative	Informative	Comment
Figure 7-2: SpaceFOM Initialization Overview	X	X	Generally informative but provides a normative overview of the SpaceFOM initialization process.
Figure 7-3: Join Federation Process	X		
Figure 7-4: Role Determination Process	X		
Figure 7-5: Master and Early Joiner HLA Initialization	X		Generally normative; however, some of the step can occur earlier in the process flow.
Figure 7-6: Epoch and Root Reference Frame Discovery	X		
Figure 7-7: Multiphase Initialization Process	X		
Figure 7-8: HLA Time Management, Synchronization, and Transition to Execution	X		
Figure 7-9: Late Joiner Initialization	X		Generally normative; however, some of the step can occur earlier in the process flow.
Figure 7-10: SpaceFOM Execution Overview	X		On first entry into the Run Executive Loop, the HLA time advance may come from initially enabling time regulation.
Figure 7-11: Master Federate Mode Transition Overview	X		
Figure 7-12: General Federate ExCO Mode Transition Overview	X		
Figure 7-13: Run Mode Transition Overview	X		
Figure 7-14: Freeze Mode Transition Overview	X		
Figure 7-15: Shutdown Mode Transition Overview	X		
Figure 8-1: SpaceFOM Module Hierarchy	X	X	Generally informative but provides normative overview of module dependencies.

Figure	Normative	Informative	Comment
Figure 8-2: SpaceFOM Object Class Hierarchy	X	X	Generally informative but provides normative overview of object class hierarchy.
Figure C-1: Management Module Object Class Diagram		X	
Figure C-2: Management Module Interaction Class Diagram		X	
Figure C-3: Environment Module Object Class Diagram		X	
Figure C-4: Entity Module Object Class Diagram		X	
Figure D-1: Sidereal Day Compared To Mean Solar Day		X	
Figure E-1: Simple Reference Frame Tree Traversal		X	
Figure E-2: Simple Earth-Moon Tree		X	

Table H-1: Normative Nature of SpaceFOM Diagrams