IEEE Standard for Modeling and Simulation (M&S) High Level Architecture (HLA)—Federate Interface Specification

Sponsor

Simulation Interoperability Standards Committee
of the
IEEE Computer Society

Approved 21 September 2000

IEEE-SA Standards Board

Abstract: The high level architecture (HLA) has been developed to provide a common architecture for distributed modeling and simulation. The HLA defines an integrated approach that provides a common framework for the interconnection of interacting simulations. This document, the second in a family of three related HLA documents, defines the standard services of and interfaces to the HLA Runtime Infrastructure (RTI). These services are used by the interacting simulations to achieve a coordinated exchange of information when they participate in a distributed federation. The standards contained in this architecture are interrelated and need to be considered as a product set, when changes are made. They each have value independently.
Keywords: architecture, class attribute, data distribution management, federate, federation, federation execution, federation object model, high level architecture (HLA), instance attribute, instance attribute ownership, interaction class, object class, runtime infrastructure (RTI), simulation object model, time-constrained, time-regulating

---

The Institute of Electrical and Electronics Engineers, Inc.
3 Park Avenue, New York, NY 10016-5997, USA

Print: ISBN 0-7381-2621-7 SH94883
PDF: ISBN 0-7381-2622-5 SS94883

IEEE Standards documents are developed within the IEEE Societies and the Standards Coordinating Committees of the IEEE Standards Association (IEEE-SA) Standards Board. The IEEE develops its standards through a consensus development process, approved by the American National Standards Institute, which brings together volunteers representing varied viewpoints and interests to achieve the final product. Volunteers are not necessarily members of the Institute and serve without compensation. While the IEEE administers the process and establishes rules to promote fairness in the consensus development process, the IEEE does not independently evaluate, test, or verify the accuracy of any of the information contained in its standards.

Use of an IEEE Standard is wholly voluntary. The IEEE disclaims liability for any personal injury, property or other damage, of any nature whatsoever, whether special, indirect, consequential, or compensatory, directly or indirectly resulting from the publication, use of, or reliance upon this, or any other IEEE Standard document.

The IEEE does not warrant or represent the accuracy or content of the material contained herein, and expressly disclaims any express or implied warranty, including any implied warranty of merchantability or fitness for a specific purpose, or that the use of the material contained herein is free from patent infringement. IEEE Standards documents are supplied "**AS IS**."

The existence of an IEEE Standard does not imply that there are no other ways to produce, test, measure, purchase, market, or provide other goods and services related to the scope of the IEEE Standard. Furthermore, the viewpoint expressed at the time a standard is approved and issued is subject to change brought about through developments in the state of the art and comments received from users of the standard. Every IEEE Standard is subjected to review at least every five years for revision or reaffirmation. When a document is more than five years old and has not been reaffirmed, it is reasonable to conclude that its contents, although still of some value, do not wholly reflect the present state of the art. Users are cautioned to check to determine that they have the latest edition of any IEEE Standard.

In publishing and making this document available, the IEEE is not suggesting or rendering professional or other services for, or on behalf of, any person or entity. Nor is the IEEE undertaking to perform any duty owed by any other person or entity to another. Any person utilizing this, and any other IEEE Standards document, should rely upon the advice of a competent professional in determining the exercise of reasonable care in any given circumstances.

Interpretations: Occasionally questions may arise regarding the meaning of portions of standards as they relate to specific applications. When the need for interpretations is brought to the attention of IEEE, the Institute will initiate action to prepare appropriate responses. Since IEEE Standards represent a consensus of concerned interests, it is important to ensure that any interpretation has also received the concurrence of a balance of interests. For this reason, IEEE and the members of its societies and Standards Coordinating Committees are not able to provide an instant response to interpretation requests except in those cases where the matter has previously received formal consideration.

Comments for revision of IEEE Standards are welcome from any interested party, regardless of membership affiliation with IEEE. Suggestions for changes in documents should be in the form of a proposed change of text, together with appropriate supporting comments. Comments on standards and requests for interpretations should be addressed to:

Secretary, IEEE-SA Standards Board
445 Hoes Lane
P.O. Box 1331
Piscataway, NJ 08855-1331
USA

Note: Attention is called to the possibility that implementation of this standard may require use of subject matter covered by patent rights. By publication of this standard, no position is taken with respect to the existence or validity of any patent rights in connection therewith. The IEEE shall not be responsible for identifying patents for which a license may be required by an IEEE standard or for conducting inquiries into the legal validity or scope of those patents that are brought to its attention.

IEEE is the sole entity that may authorize the use of certification marks, trademarks, or other designations to indicate compliance with the materials set forth herein.

Introduction

(This introduction is not part of the IEEE Std 1516.1-2000, IEEE Standard for Modeling and Simulation (M&S) High Level Architecture (HLA)—Federate Interface Specification.)

This document has been developed to record an international standard for the M&S HLA. It serves as one of three related standards for the HLA. It defines the services and interfaces to be used by federates when participating in a federation execution.

Participants

This standard was prepared by the HLA Working Group, and sponsored by the Simulation Interoperability Standards Committee (SISC) of the IEEE Computer Society. When the working group approved this standard, it had the following membership:

Susan F. Symington, *Chair*
Katherine L. Morse, *Vice Chai*r
Mikel Petty, *Secretary*

Stephen Bachinsky
Joe Batman
Brian Beebe
Keith Briggs
Jim Calvin
Richard M. Fujimoto
Deborah Fullford
Allison Griffin
James H. Hammond
James Ivers
Andreas Kemkes
John F. Kramer
Frederick S. Kuhl
Gary M. Lightner
Reed Little
Rodney Long
Margaret Loper
Robert R. Lutz
Jeffery M. Nielsen
Marnie R. Salisbury
Randy Saunders
Roy Scrudder
David W. Seidel
Robert C. Turrell
Richard Weatherly
Annette Wilson

The working group acknowledges the following individuals who also contributed to the preparation of this standard:

Phillippe Annic
Tobin Bergen-Hill
Dominque Canazzi
Judith S. Dahmann
Peter Hoare
Mikael Karlsson
Michael Mazurek
J. Russell Noseworthy
Graham Shanks

The following members of the balloting committee voted on this standard:

Dale E. Anglin
Stephen Bachinsky
Michael R. Bachmann
David L. Barton
Robert P. Bennett
Christina L. Bouwens
Keith Briggs
Richard Briggs
Danny Cohen
Glenn Conrad
Mark Crooks
Dannie Cutts
Judith S. Dahmann
Steven Drake
Laura E. Feinerman
Jeff E. Fischer
Richard M. Fujimoto
Deborah Fullford
Brian F. Goldiez
Victor M. Gonzalez
James H. Hammond
Jack G. Harrington
Robert V. Head, Jr.
Callie M. Hill
Ronald C. Hofer
James W. Hollenbach
Margaret M. Horst
Ken Hunt
Kyle Isakson
James Ivers
Andreas Kemkes
John F. Kramer
Frederick S. Kuhl
Tom Lake
Gary M. Lightner
Paul R. Little
Reed Little
Margaret Loper
Robert R. Lutz
Jayne Lyons
Theodore F. Marz
Mark McAuliffe
David R. McKeeby
Katherine L. Morse
Thomas H. Mullins
Jeffrey M. Nielsen
Michael J. O'Connor
John Peng
Paul Perkinson
Hung Q. Phan
David R. Pratt
Paul M. Reeves
David Roberts
Marnie R. Salisbury
Randy Saunders
David W. Seidel
Richard J. Severinghaus
Joseph F. Seward
Sean Sharp
Steven M. Sheasby
John W. Sheppard
Allen H. Skees
Col. Mark Smith
Susan D. Solick
Sandra Swearingen
Stephen J. Swenson
Susan F. Symington
Donald Theune
William V. Tucker
Grant R. Tudor
John A. Tufarolo
Robert C. Turrell
Andrew J. Ventre
William F. Waite
Charles E. Walters
Richard Weatherly
Chris C. Wertman
Annette Wilson
Douglas D. Wood
Philomena M. Zimmerman

When the IEEE-SA Standards Board approved this standard on 21 September 2000, it had the following membership:

Donald N. Heirman, *Chair*
James T. Carlo, *Vice Chair*
Judith Gorman, *Secretary*

Satish K. Aggarwal
Mark D. Bowman
Gary R. Engmann
Harold E. Epstein
H. Landis Floyd
Jay Forster*
Howard M. Frazier
Ruben D. Garzon
James H. Gurney
Richard J. Holleman
Lowell G. Johnson
Robert J. Kennelly
Joseph L. Koepfinger*
Peter H. Lips
L. Bruce McClung
Daleep C. Mohla
James W. Moore
Robert F. Munzner
Ronald C. Petersen
Gerald H. Peterson
John B. Posey
Gary S. Robinson
Akio Tojo
Donald W. Zipse

*Member Emeritus

Also included is the following nonvoting IEEE-SA Standards Board liaison:

Alan Cookson, *NIST Representative*
Donald R. Volzka, *TAB Representative*

Jennifer McClain Longman
IEEE Standards Project Editor

Contents

IEEE Standard for Modeling and Simulation (M&S) High Level Architecture (HLA)—Federate Interface Specification

1. Overview

1.1 Scope

This document defines the interface between federates (simulations, supporting utilities, or interfaces to live systems) and the underlying software services that support interfederate communication in a distributed simulation domain.

1.2 Purpose

The high level architecture (HLA) has been developed to provide a common architecture for distributed modeling and simulation. To facilitate interfederate communications, HLA federates interact with an underlying software infrastructure. This specification defines the standard services and interfaces to be used by the federates to support efficient information exchange when participating in a distributed federation execution. (A federation execution occurs when sets of federates are brought together to support an objective.)

1.3 Introduction

This document is being developed to define the HLA interface specification. The HLA requires a language independent specification (LIS) and multiple language bindings to support intersimulation communication in a distributed simulation domain.

The formal definition of the modeling and simulation (M&S) high level architecture comprises three main components: the HLA rules, the HLA federate interface specification, and the HLA object model template (OMT). This document shall provide a complete description of the essential elements of the second component of the HLA, the interface specification. The other two components of the HLA formal definition are described in the documents listed in Clause 2.

The high level architecture is an integrated architecture that has been developed to provide a common architecture for M&S. The HLA requires that interfederate communications use a standard application programmer's interface (API). This specification defines the standard services and interfaces to be used by the federates in order to support efficient information exchange when participating in a distributed federation execution. Additionally, the capability for reuse of individual federates that adhere to these standard services and interfaces is increased.

This document provides a specification for the HLA functional interfaces between federates and the runtime infrastructure (RTI). The RTI provides services to federates in a way that is analogous to how a distributed operating system provides services to applications. These interfaces are arranged into seven basic service groups as follows:

a) Federation management
b) Declaration management
c) Object management
d) Ownership management
e) Time management
f) Data distribution management
g) Support services

The seven service groups describe the interface between the federates and the RTI, and the software services provided by the RTI for use by HLA federates. The RTI requires a set of services from the federate that are referred to as "RTI initiated" and are denoted with a † (printer's dagger).

Any software acting as an HLA RTI shall implement all of the federate initiated services in accordance with this specification. Likewise, any software acting as an HLA federate shall implement all of the RTI initiated federate services in accordance with this specification.

1.4 Background

1.4.1 HLA federation object model framework

A concise and rigorous description of the object model framework is essential to the specification of the interface between federates and the RTI and of the HLA services. The rules and terminology used to describe a federation object model (FOM) are described in IEEE Std 1516.2-2000. A simulation object model (SOM) describes salient characteristics of a federate to aid in its reuse and other activities focused on the details of its internal operation. As such, a SOM is not the concern of the RTI and its services. A FOM, on the other hand, deals with interfederate issues and is relevant to the use of the RTI. FOMs describe the set of object classes and interaction classes chosen for a planned federation along with the attributes of the object classes and parameters of the interaction classes.

Every HLA **object instance** is an instance of an object class found in the FOM. Object classes are chosen by the object model designer to facilitate an appropriate organizational scheme. Each object class has a set of attributes associated with it. An attribute is a distinct, identifiable portion of the object state. In this discussion, "attribute designator" refers to the attribute and "attribute value" refers to its contents. From the federation perspective, the set of all attribute values for an object instance shall completely define the state of that instance. Federates may associate additional state information with an object instance that is not communicated between federates, but this is outside the purview of the HLA federation object model.

Federates use the state of the object instances as one of the primary means of communication. At any point during a federation execution, at most one federate shall be responsible for modeling a given object instance attribute. That federate provides instance attribute values to the other federates in the federation execution

via HLA services. The federate providing the instance attribute values is said to be **updating** that instance attribute. Federates receiving those values are said to be **reflecting** that instance attribute.

The privilege to update a value for an instance attribute shall be uniquely held by a single joined federate at any given point during a federation execution. A joined federate that has the privilege to update values for an instance attribute is said to **own** that instance attribute. The RTI provides services that allow joined federates to exchange ownership of object instance attributes.

Each object instance shall have a designator. The value of an object instance designator shall be unique for each federation execution. Object instance designators shall be dynamically generated by the RTI.

The FOM framework also allows for interaction classes for each object model. The types of interactions possible and their parameters are specified within the FOM.

A **federation** is the combination of a particular FOM, a particular set of federates, and the HLA services. A federation is designed for a specific purpose using a commonly understood federation object model and a set of federates that may associate their individual semantics with that object model. A federation execution is an instance of the *Create Federation Execution* service invocation and entails executing the federation with a specific FOM and an RTI, and using various execution details.

1.4.2 Relationship of HLA and object-oriented concepts

Although the HLA OMT is the standardized documentation structure for HLA object models, FOMs and SOMs do not completely correspond to common definitions of object models in object-oriented (OO) analysis and design (OOAD) techniques. In the OOAD literature, an object model is described as an abstraction of a system developed for the purpose of fully understanding the system. To achieve this understanding, most OO techniques recommend defining several views of the system. For HLA object models, the intended scope of the system description is much narrower, focusing specifically on requirements and capabilities for federate information exchange. For SOMs, the intent is to describe the public interface of the federate in terms of an identified set of supported HLA object classes and interaction classes. A more complete description of how a federate is designed and functions internally (for example, a traditional OO object model) should be provided via documentation resources other than the SOM. For FOMs, the intent is to describe information exchange that happens during a federation execution.

Differences between HLA and OOAD principles and concepts also appear at the individual object level. In the OOAD literature, objects are defined as software encapsulations of data and operations (methods). In the HLA, objects are defined entirely by the identifying characteristics (attributes), values of which are exchanged between federates during a federation execution. Any OO-related behaviors and operations that affect the values of HLA object attributes are kept resident in the federates. Although HLA class structures are driven by subscription requirements and FOM growth concerns, class structures in OO systems are typically driven by efficiency and maintainability concerns.

As discussed above, HLA object classes are described by the attributes that are defined for them. These are, in OO parlance, data members of the class. These attributes, which are abstract properties of HLA object classes, are referred to as class attributes. HLA object instances are spawned via an HLA service using an HLA object class as a template. Each attribute contained by an HLA object instance is called an instance attribute.

OO objects interact via message passing, in which one OO object invokes an operation provided by another OO object. HLA objects do not directly interact. It is the federates that interact, via HLA services, by updating instance attribute values or sending interactions. Also, responsibility for updating the instance attributes associated with an HLA object instance can be distributed among different federates in a federation (effectively distributing responsibility for maintaining the HLA object instance's state across the

federation), whereas OO objects encapsulate state locally and associate update responsibilities with operations that are closely tied to the object's implementation in an OO programming language.

In addition to the stated semantic variations in shared terminology, other differences may also exist. Precise definitions of all HLA terms can be found in Clause 3.

1.4.3 General nomenclature and conventions

There are various entities (object classes, attributes, interaction classes, parameters, dimensions, regions, federates, object instances, etc.) referenced in this document that may have the following different "views:"

— Name: human readable or for communication between federates.

— Handle: capable of being manipulated by a computer or for communication between a federate and the RTI. Originated by the RTI, federation execution-wide unique, and unpredictable. The handle for a class attribute that is inherited shall be the same as the handle that is assigned to the class attribute in the object class in which it is declared. The handle for a parameter that is inherited shall be the same as the handle that is assigned to the parameter in the interaction class in which it is declared.

The arguments to the services described in this document will use different views of the entities depending on a particular RTI implementation. For clarity, the first 11 clauses of this document refer to only a generic view, known as a "designator," when referring to these entities, and the API annexes refer to the actual implementation-specific arguments. Clause 12 ties the two concepts together.

Some of the arguments to the services described in this document are in the form of a group that represents some association. These groupings can take several forms (depending on the requirements of the respective service), each of which is described below. The groupings may be implemented slightly differently in each API, but all implementations shall support the same semantics. These groupings are pairs, sets, collections, constrained sets of pairs, and collections of pairs. Collections are sets that permit duplication of elements. They are used in cases in which duplication is unlikely and not really an error. As such, using a collection rather than a set retains the same effects, but without the expensive checks to eliminate duplicates that using a set would require. In these cases, it is preferred not to penalize the majority of users with the less efficient set just to catch errors that are anticipated to be infrequent.

The following data shall be needed for the implementation of a running RTI and federation executions:

— FOM Document Data (FDD): FOM information (class, attribute, parameter names, etc.) used by the RTI at runtime. Each federation execution needs an FDD. In the abstract, creation of a federation execution is simply the binding of a federation execution name to an FDD.

The following data may be needed for the implementation of a running RTI and federation executions:

— RTI initialization data (RID): RTI vendor-specific information needed to run an RTI. If required, a RID is supplied when an RTI is initialized.

For all joined federate-initiated services in this specification, except 4.2, *Create Federation Execution*, 4.3, *Destroy Federation Execution*, and 4.4, *Join Federation Execution*, an implied supplied argument that is a joined federate's connection to a federation execution. For all RTI-initiated services, an implied supplied argument is also a joined federate's connection to a federation execution. The manner in which these arguments are actually provided to the services is dependent on the RTI implementation and the particular API programming language, and therefore it is not shown in the service descriptions. Also, for the RTI-initiated services, some implicit preconditions are not stated explicitly because the RTI is assumed to be well behaved.

As mentioned earlier, all RTI-initiated services are denoted with a † (printer's dagger) after the service name.

1.4.4 Organization of this document

The seven HLA service groups are specified in Clause 4 through Clause 10. Each service is described using several components:

— Name and description: service name and narrative describing the functionality of the service.

— Supplied arguments: arguments returned by the service.

— Preconditions: conditions that shall exist for the service to execute correctly.

— Postconditions: conditions that shall exist once the service has executed correctly.

— Exceptions: notifications of any irregularity that may occur during service invocation. Wherever possible (excepting situations like RTI internal error), exceptions are side-effect free.

— Reference state charts: reference to state chart(s) that are germane to the service.

After the clauses describing each of the service groups is a clause describing the management object model (MOM) and then a clause that ties the service narratives with the APIs.

Annex A through Annex C contain HLA application programmer's interfaces (APIs) for the following languages:

— Ada 95

— Java

— C++

Annex D and Annex E contain a bibliography and rationale, respectively.

2. References

This standard shall be used in conjunction with the following publications. When the following specifications are superseded by an approved revision, the revision shall apply.

IEEE Std 1516-2000, IEEE Standard for Modeling and Simulation (M&S) High Level Architecture (HLA)—Framework and Rules.[1]

IEEE Std 1516.2-2000, IEEE Standard for Modeling and Simulation (M&S) High Level Architecture (HLA)—Object Model Template (OMT) Specification.

The Unicode Consortium, Ed., The Unicode Standard, Version 3.0. Reading, MA: Addison-Wesley Developers Press, 2000.

[1]IEEE publications are available from the Institute of Electrical and Electronics Engineers, 445 Hoes Lane, P.O. Box 1331, Piscataway, NJ 08855-1331, USA (http://standards.ieee.org/).

3. Definitions, abbreviations, and acronyms

3.1 Definitions

For the purposes of this standard, the following terms and definitions apply. *The Authoritative Dictionary of IEEE Standards Terms* [B6][2] should be referenced for terms not defined in this clause.

3.1.1 accuracy: The measure of the maximum deviation of an attribute or a parameter value in the simulation or federation from reality or some other chosen standard or referent.

3.1.2 active subscription: A request to the runtime infrastructure (RTI) for the kinds of data (class attributes as well as interactions) that the joined federate is currently interested in receiving. The RTI uses this data along with publication data received from other joined federates to support services, such as

a) Start/Stop Registration
b) Turn On/Off Updates
c) Turn On/Off Interactions

The RTI also uses the subscription (and publication) data to determine how to route data to the joined federates that require that data.

3.1.3 attribute: A named characteristic of an object class or object instance. *See also*: **class attribute** and **instance attribute**.

3.1.4 attribute ownership: The property of an instance attribute that gives a joined federate the capability to supply values for that instance attribute to its federation execution. *See also*: **instance attribute**.

3.1.5 available attributes: The set of declared attributes of an object class in union with the set of inherited attributes of that object class.

3.1.6 available dimensions:

a) Pertaining to an attribute: the dimensions associated with the class attribute in the Federation Object Model (FOM). The available dimensions of an instance attribute are the available dimensions of the corresponding class attribute.
b) Pertaining to an interaction class: the dimensions associated with that interaction class in the FOM. The available dimensions of a sent interaction are the available dimensions of the interaction class specified in the *Send Interaction With Regions* service invocation.

NOTE—See 9.1 in IEEE Std 1516.1-2000.

3.1.7 available parameters: The set of declared parameters of an interaction class in union with the set of inherited parameters of that interaction class.

3.1.8 candidate discovery class: The registered class of an object instance, if subscribed. If the registered class of an object instance is not subscribed, the closest superclass of the registered class of the object instance to which the joined federate is subscribed. The term candidate discovery class pertains to object instances only.

[2]The numbers in brackets correspond to those of the bibliography in Annex E.

3.1.9 candidate received class: The sent class of an interaction, if subscribed. If the sent class of an interaction is not subscribed, the closest superclass of the sent class of the interaction to which the joined federate is subscribed. The term candidate received class pertains to interactions only.

3.1.10 class: A description of a group of items with similar properties, common behavior, common relationships, and common semantics.

3.1.11 class attribute: A named characteristic of an object class denoted by a pair composed of an object class designator and an attribute designator.

3.1.12 class hierarchy: A specification of a class–subclass or "is–a" relationship between classes in a given domain.

3.1.13 coadunate: To attach an object instance handle (and possibly an object instance name) to an object instance.

3.1.14 collection: A set in which an element may occur multiple times (this corresponds to the mathematical notion of a bag).

3.1.15 collection of pairs: A group of pairs in which multiple pairs may have the same first component and a given pair may occur multiple times.

3.1.16 compliant object model: A High Level Architecture (HLA) Federation Object Model (FOM) or Simulation Object Model (SOM) that fully conforms with all of the rules and constraints specified in Object Model Template (OMT).

NOTE—See IEEE Std 1516.2-2000.

3.1.17 constrained set of pairs: A group of pairs in which no two pairs have the same first component (this corresponds to the mathematical notion of a function). An example would be a group of instance attribute and value pairs; each instance attribute may have at most one value associated with it.

3.1.18 corresponding class attribute of an instance attribute: The class attribute that, from the perspective of a given joined federate, is the class attribute of the joined federate's known class for the object instance containing the instance attribute that has the same attribute designator as the instance attribute.

3.1.19 corresponding instance attributes of a class attribute: The instance attributes that, from the perspective of a given joined federate, are

a) Unowned instance attributes of object instances that have a known class at the joined federate equal to the object class of the class attribute and that have the same attribute designator as the class attribute, or

b) Instance attributes owned by the joined federate that belong to object instances that have a known class at the owning federate equal to the object class of the class attribute and that have the same attribute designator as the class attribute.

3.1.20 datatype: A representation convention for a data element establishing its format, resolution, cardinality, and ordinality.

3.1.21 declared attributes: The set of class attributes of a particular object class that are listed in the Federation Object Model (FOM) as being associated with that object class in the object class hierarchy tree.

3.1.22 declared parameters: The set of parameters of a particular interaction class that are listed in the Federation Object Model (FOM) as being associated with that interaction class in the interaction class hierarchy tree.

3.1.23 default range: A range lower bound and a range upper bound, defined in the Federation Object Model Document Data (FDD) and specified in terms of [0, the dimension's upper bound), for a dimension.

3.1.24 default region: A multidimensional region provided by the runtime infrastructure (RTI) that is composed of one range for each dimension found in the Federation Object Model Document Data (FDD). The bounds of each of these ranges are [0, the range's dimension's upper bound). There is no way for a federate to refer to the default region.

NOTE—See 9.1 in IEEE Std 1516.1-2000.

3.1.25 designator: Some arguments (mostly identifiers) to services may have different views (implementations), depending on a particular programming language, application programmer's interface (API). For clarity, service descriptions refer to a generic view of the arguments, known as a *designator*.

3.1.26 dimension: A named interval. The interval is defined by an ordered pair of values, the first being the dimension lower bound and the second being the dimension upper bound. A runtime infrastructure (RTI) provides a predefined interval, whose lower and upper bounds are fixed as [0, the dimension's upperbound as specified in the Federation Object Model Document Data (FDD)]. This interval provides a single basis for communication of all dimension-related data with an RTI. All normalized intervals communicated to the RTI will be subsets of this interval.

NOTE—See 9.1 of IEEE Std 1516.1-2000.

3.1.27 discover: To receive an invocation of the *Discover Object Instance †*[3] service for a particular object instance.

NOTE—See 6.5 of IEEE Std 1516.1-2000.

3.1.28 discovered class: The class that was an object instance's candidate discovery class at a joined federate when that object instance was discovered by that joined federate. *See also*: **candidate discovery class.**

NOTE—See 5.1.2 in IEEE Std 1516.1-2000.

3.1.29 exception: Notification of any irregularity that may occur during a service invocation.

3.1.30 federate: An application that may be or is currently coupled with other software applications under a Federation Object Model Document Data (FDD) and a runtime infrastructure (RTI). *See also*: **federate application** and **joined federate.**

3.1.31 federate application: An application that supports the High Level Architecture (HLA) interface to a runtime infrastructure (RTI) and that is capable of joining a federation execution. A federate application may join the same federation execution multiple times or may join multiple federation executions. However, each time a federate application joins a federation execution, it is creating a new joined federate. *See also*: **joined federate**.

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

3.1.33 federation execution: The actual operation, over time, of a set of joined federates that are interconnected by a runtime infrastructure (RTI).

[3]All RTI initiated services are denoted with a † (printer's dagger) after the service name.

3.1.34 Federation Object Model (FOM): 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.

3.1.35 Federation Object Model (FOM) Document Data (FDD): The data and information in an FOM document that is used by the *Create Federation Execution* service to initialize a newly created federation execution

NOTE—See IEEE Std 1516.2-2000.

3.1.36 handle: An identifier originated/created by the runtime infrastructure (RTI) that is federation execution-wide unique and unpredictable.

3.1.37 High Level Architecture (HLA) time axis: A totally ordered sequence of values in which each value typically represents an HLA instant of time in the physical system being modeled. For any two points T1 and T2 on the time axis, if T1 < T2, T1 represents an instant of time that occurs before the instant represented by T2.

3.1.38 in scope: Of or pertaining to an instance attribute of an object instance for which

a) The object instance is known to the joined federate

b) The instance attribute is owned by another joined federate, and

c) Either the instance attribute's corresponding class attribute is a

   1) Subscribed attribute of the known class of the object instance, or

   2) Subscribed attribute of the known class of the object instance with regions, and the update region set of the instance attribute at the owning joined federate overlaps the subscription region set of the instance attribute's corresponding class attribute at the known class of the instance attribute at the subscribing joined federate

      NOTE—See 6.1 of IEEE Std 1516.1-2000.

3.1.39 inherited attribute: A class attribute of an object class that was declared in a superclass of that object class in the object class hierarchy tree defined in the Federation Object Model (FOM).

3.1.40 inherited parameter: An interaction parameter that was declared in a superclass of that interaction class in the interaction class hierarchy tree defined in the Federation Object Model (FOM).

3.1.41 instance attribute: A named characteristic of an object instance denoted by a pair composed of the object instance designator and the attribute designator.

3.1.42 interaction: An explicit action taken by a federate that may have some effect or impact on another federate within a federation execution.

3.1.43 interaction class: A template for a set of characteristics that is common to a group of interactions. These characteristics correspond to the parameters that individual federates may associate with interactions.

3.1.44 interaction parameters: The information associated with an interaction that a federate potentially affected by the interaction may receive to calculate the effects of that interaction on its current state.

3.1.45 joined federate: A member of a federation execution, actualized by a federate application invoking the *Join Federation Execution* service as prescribed in IEEE Std 1516.1-2000. *See also*: **federate application**.

3.1.46 known class:

a) An object instance's registered class if the joined federate knows about the object instance as a result of having registered it, or

b) An object instance's discovered class if the joined federate knows about the object instance as a result of having discovered it.

3.1.47 known object instance: An object instance that a given joined federate has either registered or discovered, and one that the joined federate has not subsequently deleted (globally or locally) or was notified to remove. *See also*: **register** and **discover**.

3.1.48 logical time: A federate's current point on the High Level Architecture (HLA) time axis. Federates making use of the time management services follow restrictions on what time stamps can be sent in time stamp order (TSO) messages (relative to their logical time) to ensure that federates receiving those messages receive them in TSO.

3.1.49 lookahead: Lookahead is a nonnegative value that establishes a lower value on the time stamps that can be sent in time stamp order (TSO) messages by time-regulating joined federates. Once established, a joined federate's lookahead value may only be changed using the *Modify Lookahead* service. Each time-regulating joined federate must provide a lookahead value when becoming time-regulating.

3.1.50 Management Object Model (MOM): A group of predefined High Level Architecture (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 runtime infrastructure (RTI), and

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

3.1.51 message: A change of object instance attribute value, an interaction, or a deletion of an existing object instance, often associated with a particular point on the High Level Architecture (HLA) time axis, as denoted by the associated time stamp.

3.1.52 object class: A fundamental element of a conceptual representation for a federate that reflects the real world at levels of abstraction and resolution appropriate for federate interoperability. A template for a set of characteristics that is common to a group of object instances. These characteristics correspond to the class attributes that individual federates may publish and to which other federates may subscribe.

3.1.53 object instance: A unique instantiation of an object class that is independent of all other instances of that object class. At any point during a federation execution, the state of a High Level Architecture (HLA) object instance is defined as the collection of the values of all its instance attributes.

3.1.54 object model: A system specification defined primarily by class characteristics and relationships. The High Level Architecture (HLA) idea of an object model is similar in many ways, but not identical, to the common idea of an object model in object-oriented literature.

3.1.55 object model framework: The rules and terminology used to describe High Level Architecture (HLA) object models.

3.1.56 order type: A runtime infrastructure (RTI) provided means of ordering messages originating from multiple joined federates that are delivered to a single joined federate. Different categories of service are defined with different characteristics regarding whether and how an RTI orders messages that are to be delivered to a joined federate.

3.1.57 out of scope: Of or pertaining to an instance attribute of an object instance for which one or more of the following is not true:

a) The object instance is known to the joined federate

b) The instance attribute is owned by another joined federate, and

c) Either the instance attribute's corresponding class attribute is a

   1) Subscribed attribute of the known class of the object instance, or

   2) Subscribed attribute of the known class of the object instance with regions, and the update region set of the instance attribute at the owning joined federate overlaps the subscription region set of the instance attribute's corresponding class attribute at the known class of the instance attribute at the subscribing joined federate.

      NOTE—See 6.1 of IEEE Std 1516.1-2000.

3.1.58 overlap:

a) Pertaining to region sets: Two region sets overlap if there is a region in each set, such that the two regions overlap.

b) Pertaining to regions: If two regions have at least one dimension in common, they overlap if all ranges of dimensions that are contained in both regions overlap pairwise. If two regions do not have any dimensions in common, they do not overlap.

c) Pertaining to ranges: Two ranges A = $[a_{lower}, a_{upper})$ and B = $[b_{lower}, b_{upper})$ overlap, if and only if either $a_{lower} = b_{lower}$ or $(a_{lower} < b_{lower} < a_{upper})$.

   NOTE—See 9.1 of IEEE Std 1516.1-2000.

3.1.59 owned: Pertaining to the relationship between an instance attribute and the joined federate that has the unique right to update that instance attribute's value.

3.1.60 owned instance attribute: An instance attribute that is explicitly modeled by the owning joined federate. A joined federate that owns an instance attribute has the unique responsibility to provide values for that instance attribute to the federation, through the runtime infrastructure (RTI), as documented in the Federation Object Model Document Data (FDD).

3.1.61 pair: A grouping of two related elements (a first component and a second component), the combination of which is treated as an entity. An example of a pair would be an instance attribute grouped with its current value.

3.1.62 parameter: A named characteristic of an interaction.

3.1.63 passel: A group of attribute handle/value pairs from an *Update Attribute Values* service invocation that are delivered together via a *Reflect Attribute Values †* service invocation. All pairs within the passel have the same user-supplied tag, sent message order type, transportation type, receive message order type, time stamp (if present), and set of sent region designators (if present). A passel is a message.

3.1.64 passive subscription: A request to the runtime infrastructure (RTI) for the kinds of data (object classes and attributes as well as interactions) that the joined federate is currently interested in receiving, but unlike an active subscription, this information is not used by the RTI to arrange for data to be delivered, nor is it used to tell publishing joined federates that another joined federate is subscribing to that data (by way of *Start/Stop Registration*, *Turn On/Off Updates,* or *Turn On/Off Interactions* service invocations). This form of subscription is provided to support certain types of logger joined federates.

3.1.65 promoted: Pertaining to an object instance, as known by a particular joined federate, that has a discovered class that is a superclass of its registered class.

NOTE—See 5.1.3 of IEEE Std 1516.1-2000.

3.1.66 published:

a) Pertaining to an object class such that, from the perspective of a given joined federate, there is at least one available attribute of the object class that was an argument to a *Publish Object Class Attributes* service invocation that was not subsequently unpublished via the *Unpublish Object Class Attributes* service.

NOTE—See 5.1.2 of IEEE Std 1516.1-2000.

b) Pertaining to an interaction class that, from the perspective of a given joined federate, was an argument to a *Publish Interaction Class* service invocation that was not subsequently followed by an *Unpublish Interaction Class* service invocation for that interaction class.

NOTE—See 5.1.3 of IEEE Std 1516.1-2000.

3.1.67 published attributes of an object class: The class attributes that have been arguments to *Publish Object Class Attributes* service invocations by a given joined federate for that object class that have not subsequently been unpublished (either individually or by unpublishing the whole object class), and possibly the **HLAprivilegeToDeleteObject** attribute for that object class.

NOTE—See 5.1.2 of IEEE Std 1516.1-2000.

3.1.68 range: A subset of a dimension, defined by an ordered pair of values, the first being the range lower bound and the second being the range upper bound. This pair of values defines a semi-open interval [range lower bound, range upper bound) (i.e., the range lower bound is the smallest member of the interval, and the range upper bound is just greater than any member of the interval).

NOTE—See 9.1.1 of IEEE Std 1516.1-2000.

3.1.69 range lower bound: The first component of the ordered pair of values that is part of a range.

NOTE—See 9.1 of IEEE Std 1516.1-2000.

3.1.70 range upper bound: The second component of the ordered pair of values that is part of a range.

NOTE—See IEEE 9.1 of Std 1516.1-2000.

3.1.71 receive order (RO): A characteristic of no ordering guarantee for messages. Messages that are received as RO messages will be received in an arbitrary order by the respective joined federate. A time stamp value will be provided with the message if one was specified when the message was sent, but that time stamp has no bearing on message receipt order.

3.1.72 received class: The class that was an interaction's candidate received class at the joined federate when that interaction was received at that joined federate via an invocation of the *Receive Interaction †* service.

NOTE—See 5.1.3 of IEEE Std 1516.1-2000.

3.1.73 received parameters: The set of parameters received when the *Receive Interaction †* service is invoked. These parameters consist of the subset of the sent parameters of an interaction that are available parameters of the interaction's received class.

NOTE—See 5.1.3 of IEEE Std 1516.1-2000.

3.1.74 reflect: Receive new values for one or more instance attributes via invocation of the *Reflect Attribute Values †* service.

NOTE—See 6.7 of IEEE Std 1516.1-2000.

3.1.75 reflected instance attribute: An instance attribute that is represented but not explicitly modeled in a joined federate. The reflecting joined federate accepts new values of the reflected instance attribute as they are produced by some other federation member and provided to it by the runtime infrastructure (RTI), via the *Reflect Attribute Values †* service.

3.1.76 region: A generic term that refers to either a region specification or a region realization. If not using Data Distribution Management (DDM), region arguments may be ignored.

NOTE—See 9.1 of IEEE Std 1516.1-2000.

3.1.77 region realization: A region specification (set of ranges) that is associated with an instance attribute for update, with a sent interaction, or with a class attribute or interaction class for subscription. Region realizations are created from region specifications via the *Commit Region Modifications* (only when modifying a region specification from which region realizations are already derived), *Register Object Instance With Regions*, *Associate Regions for Updates*, *Subscribe Object Class Attributes With Regions*, *Subscribe Interaction Class With Regions*, *Send Interaction With Regions*, or *Request Attribute Value Update With Regions* services.

NOTE—See 9.1.1 of IEEE Std 1516.1-2000.

3.1.78 region specification: A set of ranges. Region specifications are created using the *Create Region* service, and a runtime infrastructure (RTI) is notified of changes to a region specification using the *Commit Region Modifications* service.

NOTE—See 9.1.1 of IEEE Std 1516.1-2000.

3.1.79 register: To invoke the *Register Object Instance* or the *Register Object Instance With Regions* service to create a unique object instance designator.

NOTE—See 6.4 of IEEE Std 1516.1-2000.

3.1.80 registered class: The object class that was an argument to the *Register Object Instance* or the *Register Object Instance With Regions* service invocation that resulted in the creation of the object instance designator for a given object instance.

3.1.81 resolution: The smallest resolvable value separating attribute or parameter values that can be discriminated. Resolution may vary with magnitude for certain datatypes.

3.1.82 retraction: An action performed by a federate to unschedule a previously scheduled message. Message retraction may be visible to the federate to whom the scheduled message was to be delivered. Retraction is widely used in classic event-oriented discrete event simulations to model behaviors such as preemption and interrupts.

3.1.83 runtime infrastructure (RTI) initialization data (RID): RTI vendor-specific information needed to run an RTI. If required, an RID is supplied when an RTI is initialized.

3.1.84 runtime infrastructure (RTI): The software that provides common interface services during a High Level Architecture (HLA) federation execution for synchronization and data exchange.

3.1.85 sent class: The interaction class that was an argument to the *Send Interaction* or *Send Interaction With Regions* service invocation that initiated the sending of a given interaction.

NOTE—See 5.1.3 of IEEE Std 1516.1-2000.

3.1.86 sent interaction: A specific interaction that is transmitted by a joined federate via the *Send Interaction* or *Send Interaction With Regions* service and received by other joined federates in the federation execution via the *Receive Interaction †* service.

3.1.87 sent parameters: The parameters that were arguments to the *Send Interaction* or *Send Interaction With Regions* service invocation for a given interaction.

NOTE—See 5.1.3 of IEEE Std 1516.1-2000.

3.1.88 set: A group of elements in which each element occurs at most once (this corresponds to the mathematical notion of sets). An example of a set would be a group of class attributes, each of which belongs to the same object class.

3.1.89 Simulation Object Model (SOM): A specification of the types of information that an individual federate could provide to High Level Architecture (HLA) federations as well as the information that an individual federate can receive from other federates in HLA federations. The standard format in which SOMs are expressed facilitates determination of the suitability of federates for participation in a federation.

3.1.90 specified dimensions: The dimensions that are explicitly provided when the region specification is created or modified.

NOTE—See 9.1.2 of IEEE Std 1516.1-2000.

3.1.91 stop publishing: To take action that results in a class attribute that had been a published attribute of a class no longer being a published attribute of that class.

3.1.92 subclass: A class that adds additional detail to (specializes) another more generic class (superclass). A subclass, by inheriting the properties from its parent class (closest superclass), also inherits the properties of all superclasses of its parent as well.

3.1.93 subscribed:

a) Pertaining to an object class for which, from the perspective of a given joined federate, there are subscribed attributes of that class or subscribed attributes of that class with regions, for some region. *See also*: **subscribed attributes of an object class** and **subscribed attributes of an object class with regions**.

b) Pertaining to an interaction class that is a subscribed interaction class or a subscribed interaction class with regions, for some region. *See also*: **subscribed interaction class** and **subscribed interaction class with regions**.

3.1.94 subscribed attributes of an object class: The class attributes that have been arguments to *Subscribe Object Class Attributes* service invocations by a given joined federate for a given object class that have not subsequently been unsubscribed, either individually or by unsubscribing the whole object class.

NOTE—See 5.1.2 and 5.6 of IEEE Std 1516.1-2000.

3.1.95 subscribed attributes of an object class with regions: The class attributes that have been arguments to *Subscribe Object Class Attributes With Regions* service invocations by a given joined federate for a given object class and a given region, assuming the joined federate did not subsequently invoke the *Unsubscribe Object Class Attributes With Regions* service for that object class and region.

NOTE—See 9.8 of IEEE Std 1516.1-2000.

3.1.96 subscribed interaction class: Pertaining to an interaction class that, from the perspective of a given joined federate, was an argument to a *Subscribe Interaction Class* or *Subscribe Interaction Class With*

Regions service invocation that was not subsequently followed by an *Unsubscribe Interaction Class* or *Unsubscribe Interaction Class With Regions* service invocation for that interaction class.

NOTE—See 5.1.3 and 5.8 of IEEE Std 1516.1-2000.

3.1.97 subscribed interaction class with regions: Pertaining to an interaction class and a region that, from the perspective of a given joined federate, were arguments to a *Subscribe Interaction Class With Regions* service invocation that was not subsequently followed by an *Unsubscribe Interaction Class With Regions* service invocation for that interaction class and that region.

NOTE—See 9.10 of IEEE Std 1516.1-2000.

3.1.98 subscription region set: A set of regions used for subscription of a class attribute or used for subscription of an interaction class. *See also*: **used for subscription of a class attribute** and **used for subscription of an interaction class**.

3.1.99 superclass: A class that generalizes a set of properties that may be inherited by more refined (i.e., detailed) versions of the class. In High Level Architecture (HLA) applications, a class may have at most one immediate superclass (i.e., can only inherit from a single class at the next highest level of the class hierarchy).

3.1.100 synchronization point: A logical point in the sequence of a federation execution that all joined federates forming a synchronization set for that point attempt to reach and, if they are successful, thereby synchronize their respective processing at that point.

3.1.101 time advancing state: A joined federate may advance its logical time only by requesting a time advancement from the runtime infrastructure (RTI) via one of the following services:

a) Time Advance Request
b) Time Advance Request Available
c) Next Message Request
d) Next Message Request Available
e) Flush Queue Request

The joined federate's logical time will not actually be advanced until the RTI responds with a *Time Advance Grant †* service invocation at that joined federate. During the interval between a request to advance its logical time and the corresponding grant, the joined federate is in the time advancing state.

3.1.102 time-constrained federate: A joined federate that may receive time stamp order (TSO) messages and whose time advances are constrained by other joined federates within a federation execution.

NOTE—See 8.1 of IEEE Std 1516.1-2000.

3.1.103 time-regulating federate: A joined federate that may send time stamp order (TSO) messages and that constrains the time advances of other joined federates within a federation execution.

NOTE—See 8.1 of IEEE Std 1516.1-2000.

3.1.104 time management: A collection of High Level Architecture (HLA) services that support controlled message ordering and delivery to the cooperating joined federates within a federation execution in a way that is consistent with federation requirements.

3.1.105 time stamp (of message or save): The value of the time stamp argument provided to the relevant service invocation.

3.1.106 time stamp order (TSO): An ordering of messages provided by a runtime infrastructure (RTI) for joined federates making use of time management services and messages containing time stamps. Messages having different time stamps are said to be delivered in TSO if for any two messages M1 and M2 (time stamped with T1 and T2, respectively) that are delivered to a single joined federate where T1 < T2, then M1 is delivered before M2. Messages having the same time stamp will be delivered in an arbitrary order (i.e., no tie-breaking mechanism is provided by an RTI).

3.1.107 transportation type: A runtime infrastructure (RTI) provided means of transmitting messages between joined federates. Different categories of service are defined with different characteristics such as reliability of delivery and message latency.

3.1.108 unspecified dimensions: The available dimensions of a class attribute, instance attribute, interaction class, or sent interaction less the specified dimensions of the region specification from which the region realization is derived.

NOTE—See 9.1.3 of IEEE Std 1516.1-2000.

3.1.109 update: Invoke the *Update Attribute Values* service for one or more instance attributes.

NOTE—See 6.6 of IEEE Std 1516.1-2000.

3.1.110 update region set: A set of regions used for sending interactions or used for update of instance attributes. *See also*: **used for sending** and **used for update**.

3.1.111 used for sending:

a) Pertaining to a region that, along with the specified interaction class designator, is being used as an argument in the *Send Interaction With Regions* service.

b) Pertaining to the default region when the specified interaction class designator is being used as an argument in the *Send Interaction* service.

NOTE—See 9.1 of IEEE Std 1516.1-2000.

3.1.112 used for subscription of a class attribute:

a) Pertaining to a region, an object class, and a class attribute for which the class attribute is a subscribed attribute of the object class with that region. *See also*: **subscribed attributes of an object class with regions.**

NOTE—See 9.1 of IEEE Std 1516.1-2000.

b) Pertaining to the default region when the specified class attribute is a subscribed attribute of the specified class. *See also*: **subscribed attributes of an object class with regions.**

NOTE—See 9.1 of IEEE Std 1516.1-2000.

3.1.113 used for subscription of an interaction class:

a) Pertaining to a region and an interaction class for which the interaction class is a subscribed interaction class with regions. *See also*: **subscribed interaction class**.

NOTE—See 9.1 of IEEE Std 1516.1-2000.

b) Pertaining to the default region when the specified interaction class is a subscribed interaction class. *See also*: **subscribed interaction class**.

NOTE—See 9.1 of IEEE Std 1516.1-2000.

3.1.114 used for update:

a) Pertaining to a region that, along with the specified object instance and attribute designators, has been used as an argument in either the *Register Object Instance With Regions* service or the *Associate Regions For Updates* service; and the region has not subsequently been used along with the specified object instance designator as an argument in the *Unassociate Regions For Updates* service; nor has the joined federate subsequently lost ownership of the specified instance attribute(s).

NOTE—See 9.1 of IEEE Std 1516.1-2000.

b) Pertaining to the default region when the specified instance attribute(s) is not currently used for update with any other region.

NOTE—See 9.1 of IEEE Std 1516.1-2000.

3.2 Abbreviations and acronyms

The following abbreviations and acronyms pertain to this standard.

API	application programmer's interface
BNF	Backus Naur Form
DDM	Data Distribution Management
DIF	data interchange format
DM	Declaration Management
DTD	Document Type Declaration
FDD	FOM Document Data
FOM	Federation Object Model
FQR	*Flush Queue Request*
GALT	Greatest Available Logical Time
HLA	High Level Architecture
LITS	Least Incoming Time Stamp
MOM	Management Object Model
M&S	modeling and simulation
NA	not applicable
NMR	*Next Message Request*
NMRA	*Next Message Request Available*
OMT	Object Model Template
OO	object oriented
OOAD	object-oriented analysis and design
POC	point of contact
RID	RTI initialization data
RO	receive order
RTI	runtime infrastructure
SISC	Simulation Interoperability Standards Committee
SOM	Simulation Object Model
TAR	*Time Advance Request*
TARA	*Time Advance Request Available*
TRADT	time representation abstract datatype
TSO	time stamp order
XML	eXtensible Markup Language

4. Federation management

4.1 Overview

Federation management refers to the creation, dynamic control, modification, and deletion of a federation execution.

Figure 1 shows the overall state of a federation execution as certain basic federation management services are employed. The figure starts in the Federation Execution Does Not Exist state, representing that the RTI has been started up, but that no services have been invoked relative to the federation execution. After the federation execution is created, it is now in the No Joined Federates state. This state represents the clean, or freshly initialized, federation execution; no federates are joined, no object instances exist, and no messages are queued.[4] From the point that the first federate joins this execution until the point that the last federate resigns from the execution, the federation execution is in the Supporting Joined Federates state. This state represents an operational federation execution in which the services documented in this specification are usable.

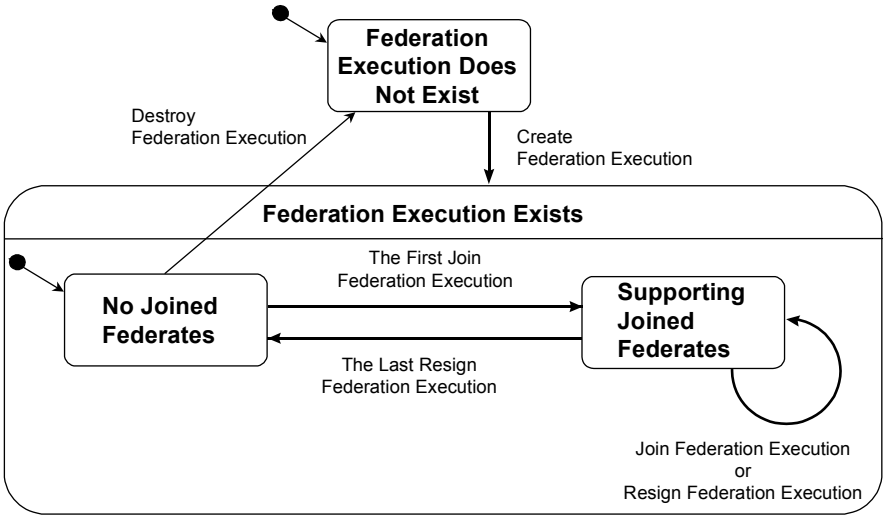

Figure 1—Basic states of the federation execution

Once a federation execution exists, federates may join and resign from it in any sequence that is meaningful to the federation user. Figure 2 presents a generalized view of the basic relationship between a federate and the RTI during the federate participation in a federation execution. The broad arrows in Figure 2 represent the general invocation of HLA service groups and are not intended to demonstrate strict ordering requirements on the use of the services. An RTI shall allow an application to participate in a federation as multiple joined federates, and an RTI shall allow a single application to participate in multiple federations as a joined federate or federates.

The state diagram in Figure 3 is the first of a series of hierarchical state diagrams that formally describe the exchanges between federates and an RTI. These state diagrams are formal, accurate descriptions of federate state information depicted in the highly structured, compact, and expressive statechart notation pioneered by

[4]Note that this state is also reachable after all federates have resigned from a running execution; in this case, the federation also reverts to the clean, or freshly initialized state.

David Harel [B5]. This specification contains two independent sets of statecharts. The first set, beginning in Figure 3 and including all other statecharts except Figure 5 through Figure 8, describes the state of a given federate, from the perspective of that federate, in varying levels of detail. The second set, Figure 5 through Figure 8, describes the state of a federation execution (including the RTI and all joined federates) strictly with respect to the use of synchronization points. Finally, Figure 1 is unrelated to either of these sets of statecharts. It is a stand-alone overview of the basic states of a federation execution.

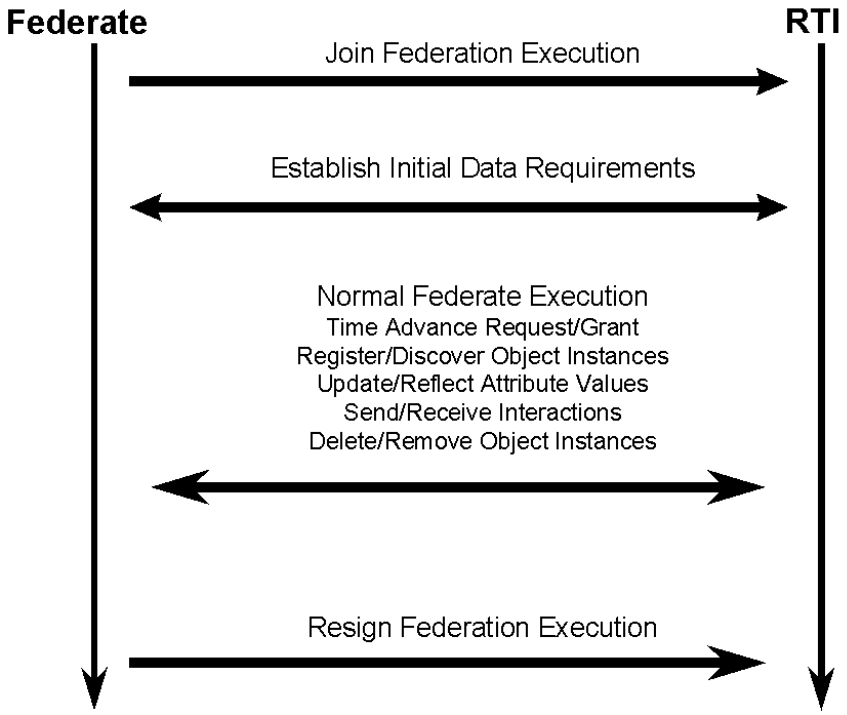

Figure 2—Overall view of federate-to-RTI relationship

The next few subclauses describe the first two of these statecharts in detail as a way of introducing some of Harel's notation and providing an understanding of how the complete collection of statecharts in this document are hierarchically interrelated.

4.1.1 Federate lifetime

As shown in Figure 3, with the successful completion of the *Join Federation Execution* service, a federate will be in the Joined Federate state, where it will remain until it resigns from the federation execution. As indicated by the dashed line in the Joined Federate state, the Joined Federate state consists of two parallel state machines: one having to do with whether the joined federate is in the process of saving or restoring the joined federate state (depicted to the left of the dashed line), and the other having to do with whether the joined federate is permitted to perform normal activity (depicted to the right of the dashed line). While in the Joined Federate state, the joined federate is simultaneously in both a state depicted in the state machine to the left of the dashed line and a state depicted in the state machine to the right of the dashed line. Initially, upon entering the Joined Federate state, the joined federate will be in the Active and Normal Activity Permitted states, as indicated by the dark-circle start transitions. There are interdependencies between these two parallel state machines and between the state machine on the left and the Temporal State state machine, Figure 16, which appears later in this document. These interdependencies are depicted by the guards (shown within square brackets) that are associated with some state transitions. If a transition has a guard associated with it, then when the assertion within the guard is true, the joined federate may make the associated

transition from one state to another. If a transition only has a guard (with no associated service invocation), the joined federate will immediately make the associated transition as soon as the guard is true.

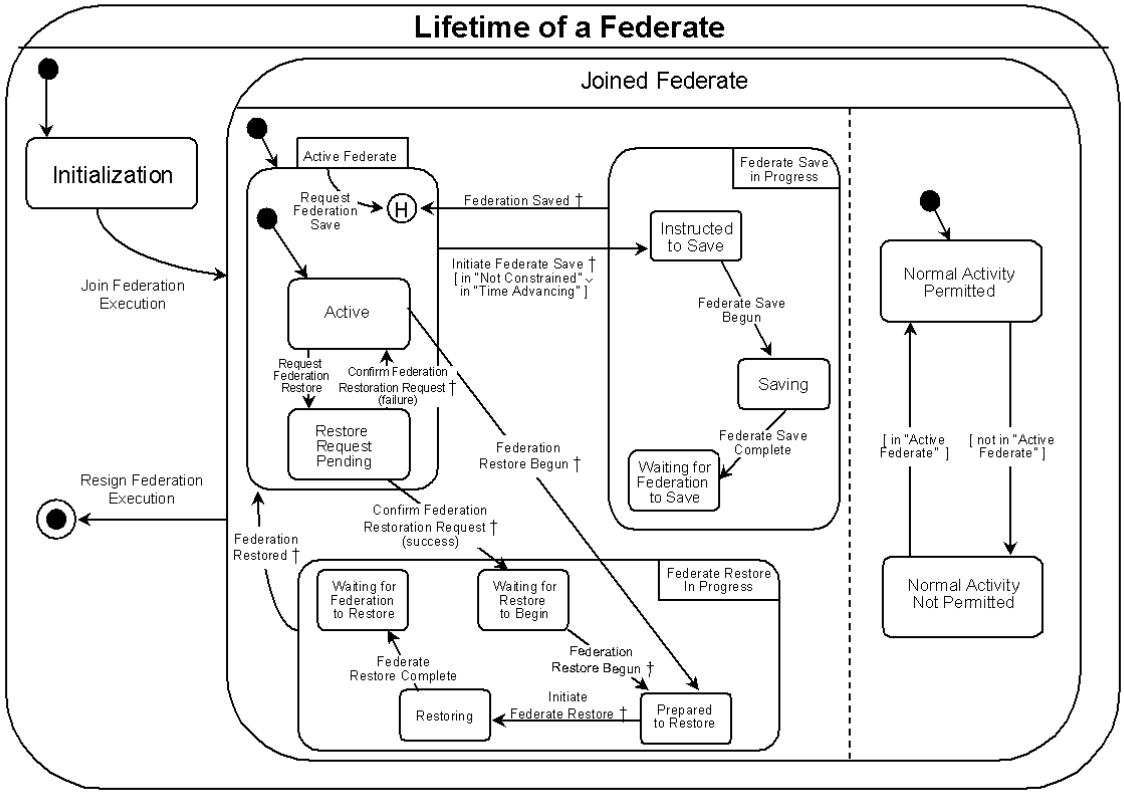

Figure 3—Lifetime of a federate

As an example of an interdependency between the two parallel state machines depicted in the Joined Federate state, if a joined federate that is in the Active state receives a *Federation Restore Begun †* [5] service invocation, it will transition into the Prepared to Restore state (as indicated by the label on the transition from the Active state to the Prepared to Restore state). Once the joined federate enters the Prepared to Restore state, it also enters the Normal Activity Not Permitted state (as indicated by the guard on the transition from the Normal Activity Permitted to the Normal Activity Not Permitted state). That is, the guards impose the following constraints on a joined federate: A joined federate may be in the Normal Activity Permitted state (right side) if and only if it is also in the Active Federate state (left side); and a joined federate may be in the Normal Activity Not Permitted state (right side) if and only if it is also in either the Instructed to Save, Saving, Waiting for Federation to Save, Prepared to Restore, Restoring, Waiting for Federation to Restore, or Waiting for Restore to Begin state (left side).

The interdependency between the state machine on the left and the Temporal State state machine, Figure 16, depicted later in this document, is this: a joined federate that is in the Active state will not receive an invocation of the *Initiate Federate Save †* service unless that joined federate is either in the Not Constrained or the Time Advancing state. The Not Constrained and Time Advancing states are depicted in Figure 16. The fact that these two time management-related states are mentioned in the guard on the transition from the Active to the Instructed to Save state demonstrates the interdependencies between a joined federate's save/restore state and its temporal state. Specifically, it indicates that a joined federate shall either not be

[5] All RTI initiated services are denoted with a † (printer's dagger) after the service name.

constrained by time management or be in a position to receive a time advance grant in order for it to receive an invocation of the *Initiate Federate Save †* service.

There is a statechart notation that needs additional explanation, that of the shallow history state, depicted by a state labeled **H.** This means to enter the most recently visited of all the states at the level of the **H** state, or enter the initial state (as indicated by the dark-circle start transition) if this is the first occurrence in. For example, in Figure 3 there is a shallow history state within the Active Federate state. The transition for Request Federation Save originates from the outer edge of the Active Federate state, meaning that the transition can occur from any state within the Active Federate state (i.e., Active or Restore Request Pending). The transition terminates at the shallow history state, meaning that on completing the transition, the federate will return to whatever state within Active Federate it was previously in (i.e., it will return to Active or to Restore Request Pending). Transition from a superstate to a shallow history state is a common idiom denoting that the transition can occur while in any substate of the super-state and will result in the machine returning to the same state it was in prior to the transition.

If a joined federate is in the Normal Activity Permitted state, the joined federate may perform normal joined federate activity, such as registering and discovering object instances, publishing and subscribing to object class attributes and interactions, updating and reflecting instance attribute values, sending and receiving interactions, deleting and removing object instances, and requesting or receiving time advance grants. The Normal Activity Permitted state, simple as it may appear in the Joined Federate statechart, actually contains all of the other states that appear in the first set of statecharts that appear subsequently in this document. Together, these statecharts formally describe the state of a joined federate from that joined federate's perspective. These statecharts are complete in the sense that all transitions shown represent legal operations and transitions that are not shown represent illegal operations. Illegal operations shall generate exceptions if invoked. The statecharts do not include MOM or DDM activities.

The Normal Activity Permitted state depicted in Figure 3 is elaborated in further detail in Figure 4, to identify the major portions of joined federate state. Additionally, the states in Figure 4 are further elaborated as follows:

— Temporal state (Figure 16)
— States associated with each object class (Figure 9)
— States associated with each interaction class (Figure 12)

When a federate enters the Joined Federate state, the joined federate will have a Temporal State, object class states, and interaction class states. The joined federate will have an Object Class State for each object class that is defined in the FDD that is associated with the federation execution. Likewise, the joined federate will have an Interaction Class State for each interaction class that is defined in the FDD. A joined federate will be in the Temporal State, in each of these Object Class states, and in each of these Interaction Class states simultaneously (as depicted by the dashed lines separating the state machines within Normal Activity Permitted). Time management is elaborated in further detail in Figure 16. The state of an arbitrary object class is elaborated in further detail in Figure 9, and the state of an arbitrary interaction class is elaborated in further detail in Figure 12. Indices, such as "i" in Object Class (i), are used in the names of many statecharts to denote that the statechart refers to an arbitrary occurrence of something. These indices are used consistently throughout the specification and are described as follows:

i Object class designator
j Attribute designator
k Object instance designator
m Interaction class designator
L Synchronization label
s *s*th joined federate; also used as F_s

There is another statechart notation that needs additional explanation, that of the deep history state, depicted by a state labeled **H*.** This means to enter the most recently visited of all the states at the level of the **H*** state, or to enter the initial state (as indicated by the dark-circle start transition) if this is the first occurrence in; and to do the same for all relevant substates.

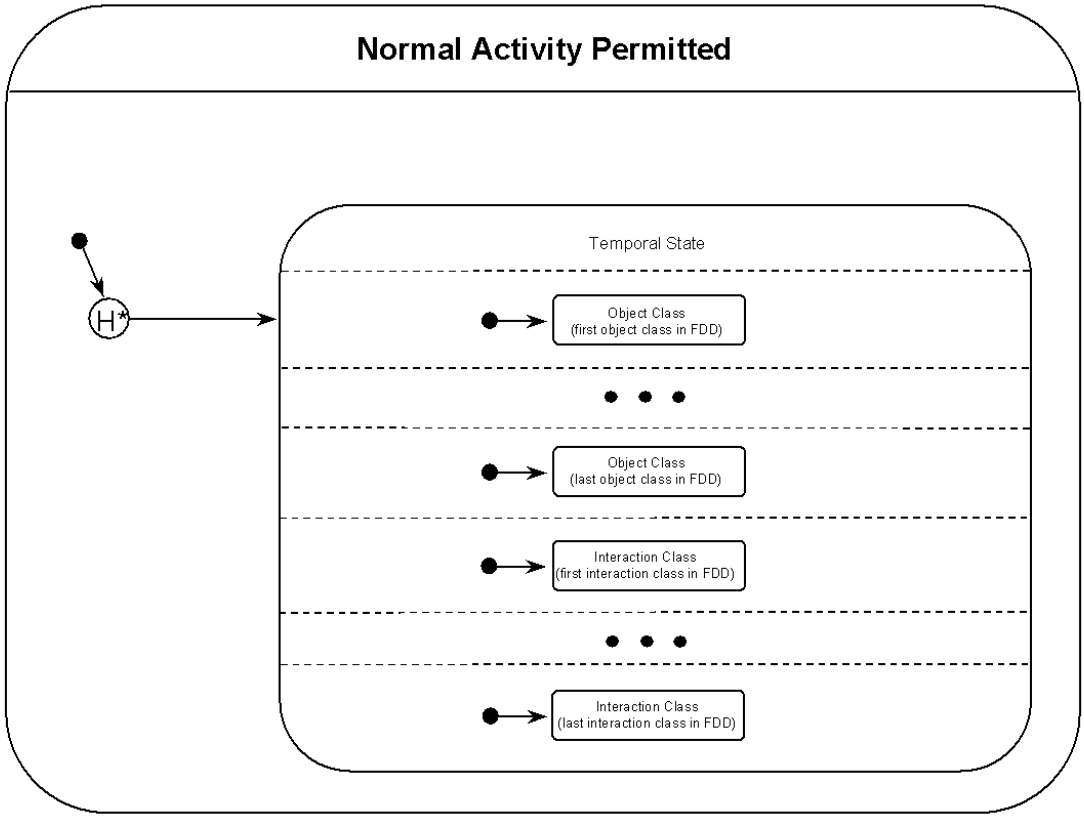

Figure 4—Normal activity permitted

4.1.2 Federation execution save and restore

Any joined federate in the federation execution may initiate a save by invoking the *Request Federation Save* service. If there is no time stamp argument provided with the invocation of this service, the RTI shall instruct all of the joined federates in the federation execution (including the requesting joined federate) to save state by invoking the *Initiate Federate Save †* service at all of these joined federates as soon as possible. If there is a time stamp argument provided, the RTI will invoke the *Initiate Federate Save †* service at each of the time-constrained joined federates when their value of logical time advances to the value provided, and it will invoke the *Initiate Federate Save †* service at all nontime-constrained joined federates as soon as all of the time-constrained joined federates are eligible to be instructed to save.

When a joined federate receives an *Initiate Federate Save †* service invocation and subsequently saves its state, it shall use the federation save label (that was specified by the joined federate requesting the save in the *Request Federation Save* service), its federate type (that it specified when it joined the federation execution), and its federate designator (that was returned when it joined the federation execution) to distinguish the saved information.

The saved information shall be persistent, meaning that it is stored onto disk or some other persistent medium, and it remains intact even after the federation execution is destroyed. The saved information can thus be used at a later date, by some new set of joined federates, to restore all joined federates in the

federation execution to the state that they were in when the save was accomplished. The federation can then resume execution of the execution from that saved point. The set of federates joined to a federation execution when state is restored from a previously saved state need not be the exact set of federates that were joined to the federation execution when the state being restored was saved. The number of federates of each federate type that are joined to the federation execution, however, shall be the same. The federate-type argument supplied in the *Join Federation Execution* service invocation, therefore, is crucial to the save-and-restore process. Declaring a federate to be of a given type shall be equivalent to asserting that the joined federate can be restored using the state information saved by any other joined federate of that type.

There is no requirement that a save taken by one RTI implementation be restorable by another RTI implementation.

4.1.3 Synchronization discussion

Figure 5 depicts the state of a federation execution with respect to synchronization points. In order to fully explain synchronization points, both the state of the RTI and the state of each joined federate are elaborated. Because multiple joined federates are involved, each transition within this set of statecharts includes an argument denoting to which joined federate a service invocation applies (i.e., an argument of F_s denotes the sth joined federate).

The RTI shall keep track of each synchronization point separately, and the treatment of an arbitrary synchronization point by the RTI is expanded in Figure 6. Not shown in this diagram is how the RTI calculates the synchronization set (shown as *synch_set* within the statechart); this information is elaborated in the service descriptions. Use of the synchronization set, however, is depicted in the statechart on the transition from the Synchronizing (s, L) states to the Synch Point Not Registered state. This transition denotes, by use of the universal quantifier over the synchronization set in the guard, that synchronization throughout a federation is only achieved when all joined federates in the synchronization set have concluded their individual synchronizations. A joined federate's individual synchronization, from the RTI's point of view, is elaborated in Figure 7.

The final statechart in the set describing synchronization points is Figure 8. This statechart depicts how an arbitrary joined federate keeps track of an arbitrary synchronization point. This diagram includes how a joined federate registers a synchronization point, as well as how a joined federate actually synchronizes.

.

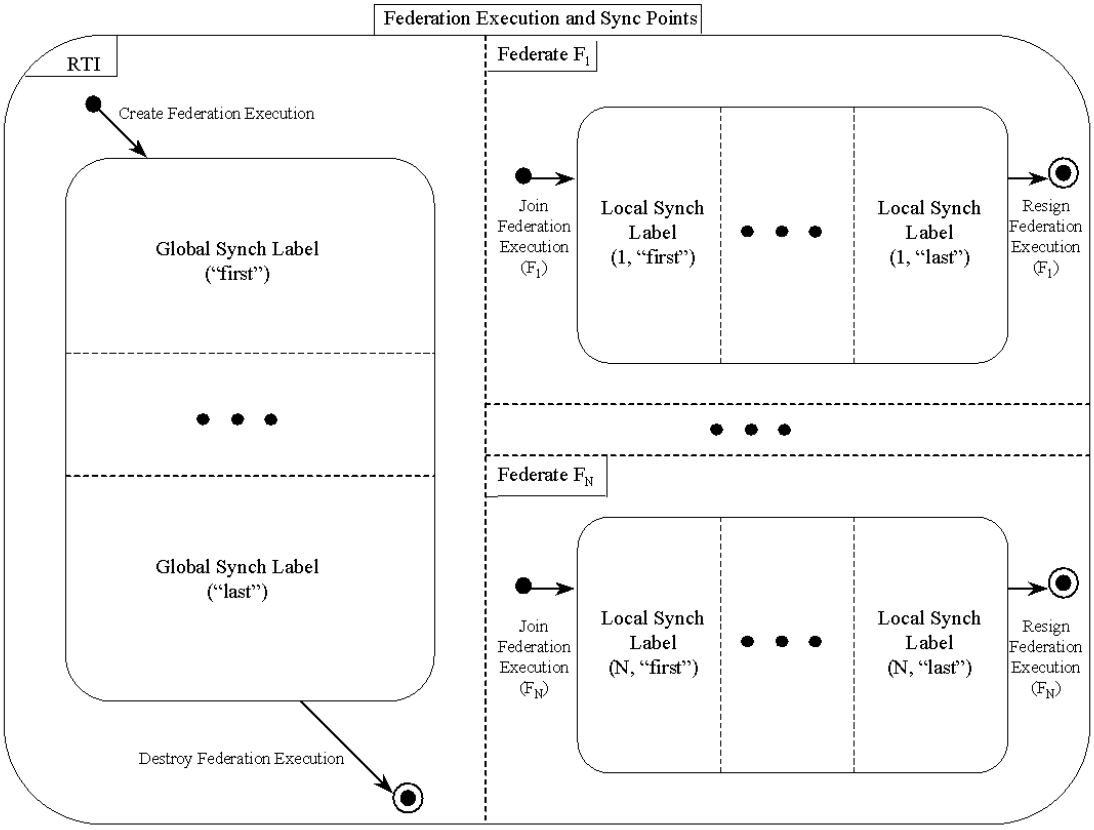

Figure 5—Federation execution and sync points

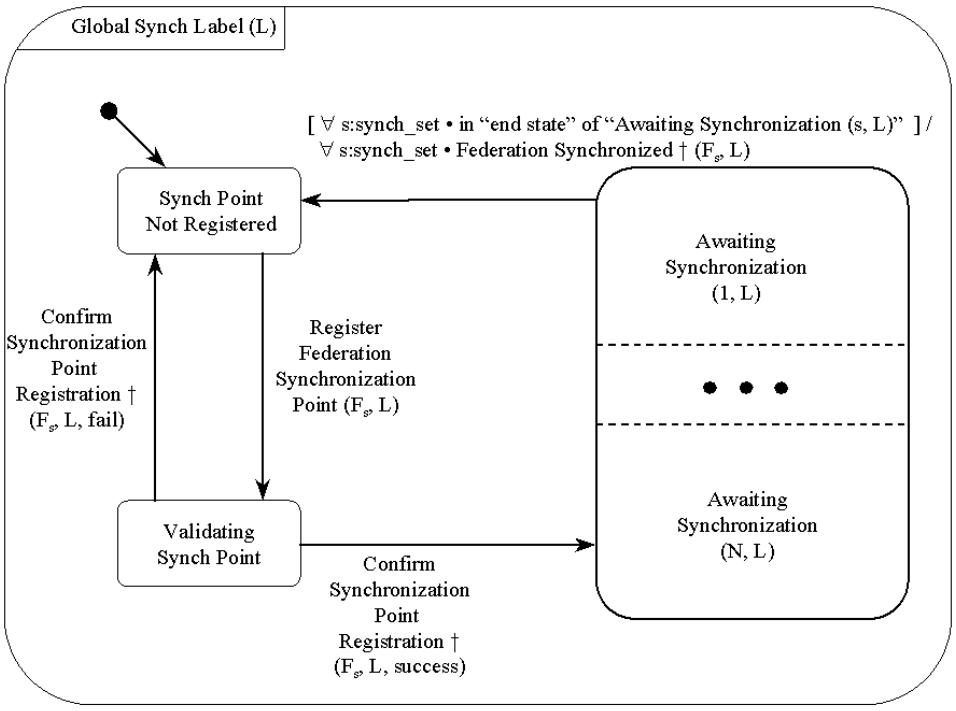

Figure 6—Global synch label (L)

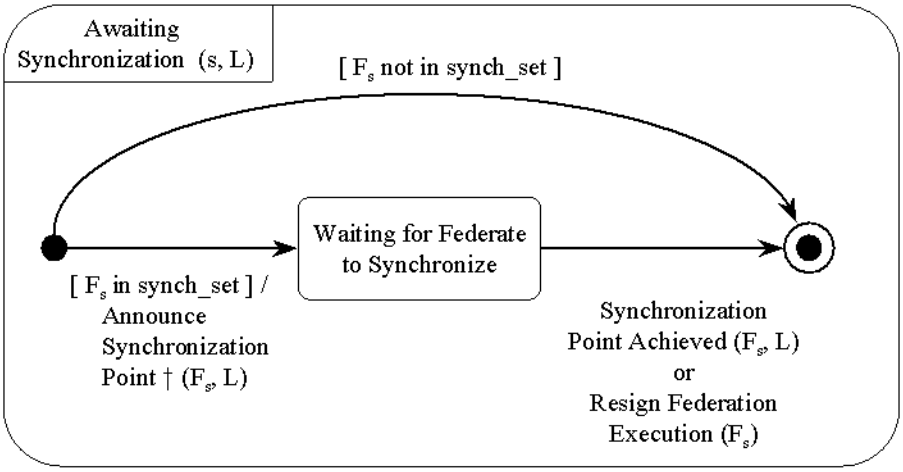

Figure 7—Awaiting synchronization (s, L)

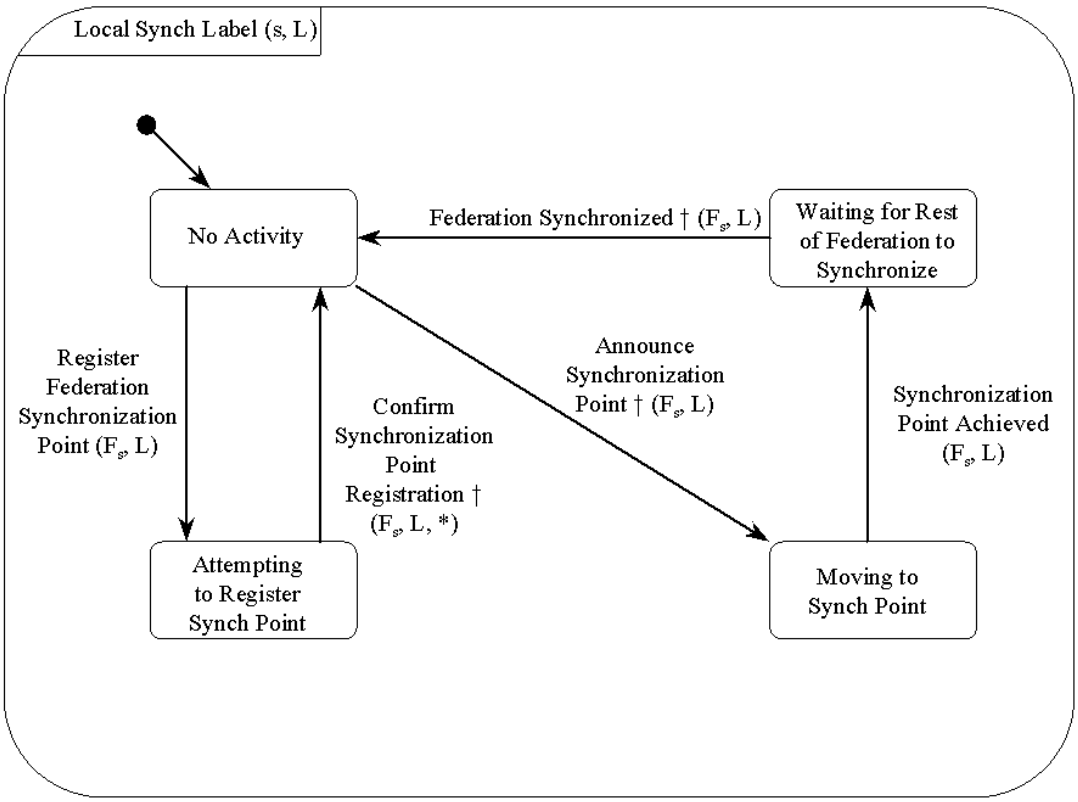

Figure 8—Local synch label (s, L)

4.1.4 FOM document data (FDD)

The FDD is that data and information in a FOM document, as defined in IEEE Std 1516.2-2000, that is used by the *Create Federation Execution* service to initialize a newly created federation execution. The FOM document designator argument supplied to that service shall be a compliant FOM document (IEEE Std 1516-2000 and IEEE Std 1516.2-2000). *Create Federation Execution* shall reject, issuing an exception, any entity that is not a compliant FOM document. In order to support functionality described in this specification, the FDD that the RTI uses from the FOM document shall be, at a minimum, the following:

— Object class structure table

— Interaction class structure table

— Attribute table (transportation, order, and available dimensions columns only)

— Parameter table (transportation, order, and available dimensions columns only)

— Dimension table

— Transportation type table

— Switches table

4.2 Create Federation Execution

The *Create Federation Execution* service shall create a new federation execution and add it to the set of supported federation executions. Each federation execution created by this service shall be independent of all other federation executions, and there shall be no intercommunication within the RTI between federation executions. The FOM document designator argument shall identify the FOM that furnishes the FDD for the federation execution to be created.

4.2.1 Supplied arguments

a) Federation execution name

b) FOM document designator

4.2.2 Returned arguments

a) None.

4.2.3 Preconditions

a) The federation execution does not exist.

4.2.4 Postconditions

a) A federation execution exists with the given name that may be joined by federates.

4.2.5 Exceptions

a) The specified federation execution already exists.

b) Could not locate FOM document indicated by supplied designator.

c) Invalid FOM document.

d) RTI internal error.

4.2.6 Reference state charts

a) Figure 1: Basic states of the federation execution.

b) Figure 5: Federation execution and sync points.

4.3 Destroy Federation Execution

The *Destroy Federation Execution* service shall remove a federation execution from the RTI set of supported federation executions. All federation activity shall have stopped, and there shall be no joined federates (all joined federates shall have resigned, either by explicit action or via MOM activity) before this service is invoked.

4.3.1 Supplied arguments

a) Federation execution name.

4.3.2 Returned arguments

a) None

4.3.3 Preconditions

a) The federation execution exists.

b) There are no federates joined to this federation execution.

4.3.4 Postconditions

a) The federation execution does not exist.

4.3.5 Exceptions

a) Federates are joined to the federation execution.

b) The federation execution does not exist.

c) RTI internal error.

4.3.6 Reference state charts

a) Figure 1: Basic states of the federation execution.

b) Figure 5: Federation execution and sync points.

4.4 Join Federation Execution

The *Join Federation Execution* service shall affiliate the federate with a federation execution. Invocation of the *Join Federation Execution* service shall indicate the intention to participate in the specified federation. The federate-type argument shall distinguish federate categories for federation save-and-restore purposes. The returned joined federate designator shall be unique for the lifetime of the federation execution.

4.4.1 Supplied arguments

a) Federate type.

b) Federation execution name.

4.4.2 Returned arguments

a) Joined federate designator.

4.4.3 Preconditions

a) The federation execution exists.

b) The federate is not joined to that federation execution.

c) Save not in progress.

d) Restore not in progress.

4.4.4 Postconditions

a) The joined federate is a member of the federation execution.

4.4.5 Exceptions

a) The federate is already joined to the federation execution.

b) The specified federation execution does not exist.

c) Save in progress.

d) Restore in progress.

e) RTI internal error.

4.4.6 Reference state charts

a) Figure 1: Basic states of the federation execution

b) Figure 3: Lifetime of a federate

c) Figure 5: Federation execution and sync points

4.5 Resign Federation Execution

The *Resign Federation Execution* service shall indicate the requested cessation of federation participation. Before resigning, ownership of instance attributes held by the joined federate should be resolved. The joined federate may transfer ownership of these instance attributes to other joined federates, unconditionally divest them for ownership acquisition at a later point, or delete the object instance of which they are a part (assuming the joined federate has the privilege to delete these object instances). As a convenience to the joined federate, the *Resign Federation Execution* service shall accept an action argument that directs the RTI to perform zero or more of the following actions:

— Unconditionally divest all owned instance attributes for future ownership acquisition. This shall place the instance attributes into an unowned state (implying that their values are not being updated), which shall make them eligible for ownership by another joined federate. See Clause 7 for a more detailed description.

— Delete all object instances for which the joined federate has that privilege (implied invocation of the *Delete Object Instance* service).

— Cancel all pending instance attribute ownership acquisitions. The use of this directive may interfere with the intended semantics of negotiated instance attribute ownership divestiture by allowing instance attributes divested in this way to be unowned (because the cancellation directive may not succeed).

4.5.1 Supplied arguments

a) Directive to

1) Unconditionally divest ownership of all owned instance attributes.

2) Delete all object instances for which the joined federate has the delete privilege.

3) Cancel all pending instance attribute ownership acquisitions.

4) Perform action 2) and then action 1).

5) Perform action 3), action 2), and then action 1).

6) Perform no actions.

4.5.2 Returned arguments

a) None.

4.5.3 Preconditions

a) The federation execution exists.

b) The federate is joined to that federation execution.

c) If directive 2) is supplied, the joined federate does not own any instance attributes of object instances for which it does not also have the delete privilege.

d) If directive 3) or 6) is supplied, the joined federate does not own any instance attributes in the federation execution.

e) If either directive 1), 2), 4), or 6) is specified, there are no instance attributes for which the joined federate has either

   1) Invoked the *Attribute Ownership Acquisition* service and has not yet received a corresponding invocation of either the *Confirm Attribute Ownership Acquisition Cancellation †* service or the *Attribute Ownership Acquisition Notification †* service.

   2) Invoked the *Attribute Ownership Acquisition If Available* service and has not yet received a corresponding invocation of either the *Attribute Ownership Unavailable †* service or the *Attribute Ownership Acquisition Notification †* service.

   3) Invoked the *Attribute Ownership Acquisition If Available* service and has subsequently invoked the *Attribute Ownership Acquisition* service [after which condition 1) applies]

4.5.4 Postconditions

a) The federate is not a member of the federation execution.

b) There are no instance attributes in the federation execution owned by the joined federate.

c) If directive 2), 4), or 5) is supplied, all object instances for which the joined federate has the delete privilege are deleted.

d) If directive 3) or 5) is supplied, all pending instance attribute ownership acquisitions for the joined federate are canceled.

4.5.5 Exceptions

a) Cannot resign due to pending attempt to acquire instance attribute ownership.

b) The joined federate owns instance attributes.

c) Invalid resign directive.

d) The federate is not a federation execution member.

e) RTI internal error.

4.5.6 Reference state charts

a) Figure 1: Basic states of the federation execution

b) Figure 3: Lifetime of a federate

c) Figure 5: Federation execution and sync points

d) Figure 7: Awaiting synchronization (s, L)

4.6 Register Federation Synchronization Point

The *Register Federation Synchronization Point* service shall be used to initiate the registration of an upcomwsing synchronization point label. When a synchronization point label has been successfully registered (indicated through the *Confirm Synchronization Point Registration †* service), the RTI shall inform some or all joined federates of the label's existence by invoking the *Announce Synchronization Point †* service at those joined federates. The optional set of joined federate designators shall be used by the joined federate to specify the joined federates in the federation execution that should be informed of the label existence. If the optional set of joined federate designators is empty or not supplied, all joined federates in the federation execution shall be informed of the label's existence. If the optional set of designators is not empty, all designated joined federates shall be federation execution members. The user-supplied tag shall provide a vehicle for information to be associated with the synchronization point and shall be announced along with the synchronization label. It is possible for multiple synchronization points registered by the same or different joined federates to be pending at the same point.

4.6.1 Supplied arguments

a) Synchronization point label

b) User-supplied tag

c) Optional set of joined federate designators

4.6.2 Returned arguments

a) None

4.6.3 Preconditions

a) The federation execution exists.

b) The federate is joined to that federation execution.

c) If an optional set of joined federate designators is supplied, those federates shall be joined to the federation execution.

d) The supplied synchronization point label shall not be in use.

e) Save not in progress.

f) Restore not in progress.

4.6.4 Postconditions

a) The synchronization label is known to the RTI.

4.6.5 Exceptions

a) The federate is not a federation execution member.

b) Save in progress.

c) Restore in progress.

d) RTI internal error.

4.6.6 Reference state charts

a) Figure 6: Global synch label (L)

b) Figure 8: Local synch label (s, L)

4.7 Confirm Synchronization Point Registration †

The *Confirm Synchronization Point Registration †* service shall indicate to the requesting joined federate the status of a requested federation synchronization point registration. This service shall be invoked in response to a *Register Federation Synchronization Point* service invocation. If the registration-success indicator argument indicates success, this shall mean the label has been successfully registered.

If the registration-success indicator argument indicates failure, the optional failure reason argument shall be provided to identify the reason that the synchronization point registration failed. Possible reasons for the synchronization point registration failure are the following:

— The specified label is already in use.

— A synchronization set member is not a joined federate.

A registration attempt that ends with a negative success indicator shall have no other effect on the federation execution.

4.7.1 Supplied arguments

a) Synchronization point label.

b) Registration-success indicator.

c) Optional failure reason.

4.7.2 Returned arguments

a) None.

4.7.3 Preconditions

a) The federation execution exists.

b) The federate is joined to that federation execution.

c) The joined federate has invoked *Register Federation Synchronization Point* service for the specified label.

4.7.4 Postconditions

a) If the registration success indicator is positive, the specified label and associated user-supplied tag will be announced to the appropriate joined federates.

b) If the registration success indicator is negative, this service and the corresponding *Register Federation Synchronization Point* service invocation have no consequence.

4.7.5 Exceptions

a) Federate internal error.

4.7.6 Reference state charts

a) Figure 6: Global synch label (L)

b) Figure 8: Local synch label (s, L)

4.8 Announce Synchronization Point †

The *Announce Synchronization Point †* service shall inform a joined federate of the existence of a new synchronization point label. When a synchronization point label has been registered with *the Register Federation Synchronization Point* service, the RTI shall invoke the *Announce Synchronization Point †* service, at either all the joined federates in the federation execution or at the specified set of joined federates, to inform them of the label existence. The joined federates informed of the existence of a given synchronization point label via the *Announce Synchronization Point †* service shall form the synchronization set for that point. If the optional set of joined federate designators was null or not provided when the synchronization point label was registered, the RTI shall also invoke the *Announce Synchronization Point †* service at all federates that join the federation execution after the synchronization label was registered but before the RTI has ascertained that all joined federates that were informed of the synchronization label existence have invoked the *Synchronization Point Achieved* service. These newly joining federates shall also become part of the synchronization set for that point. Joined federates that resign from the federation execution after the announcement of a synchronization point but before the federation synchronizes at that point shall be removed from the synchronization set. The user-supplied tag supplied by the *Announce Synchronization Point †* service shall be the tag that was supplied to the corresponding *Register Federation Synchronization Point* service invocation.

4.8.1 Supplied arguments

a) Synchronization point label

b) User-supplied tag

4.8.2 Returned arguments

a) None

4.8.3 Preconditions

a) The federation execution exists.

b) The federate is joined to that federation execution.

c) The synchronization point has been registered.

4.8.4 Postconditions

a) The synchronization label is known to the joined federate and may be used in the *Synchronization Point Achieved* and *Federation Synchronized †* services.

4.8.5 Exceptions

a) Federate internal error

4.8.6 Reference state charts

a) Figure 7: Awaiting synchronization (s, L)

b) Figure 8: Local synch label (s, L)

4.9 Synchronization Point Achieved

The *Synchronization Point Achieved* service shall inform the RTI that the joined federate has reached the specified synchronization point. Once all joined federates in the synchronization set for a given point have invoked this service, the RTI shall not invoke the *Announce Synchronization Point †* on any newly joining federates.

4.9.1 Supplied arguments

a) Synchronization point label

4.9.2 Returned arguments

a) None

4.9.3 Preconditions

a) The federation execution exists.

b) The federate is joined to that federation execution.

c) The synchronization point has been announced at this federate.

d) Save not in progress.

e) Restore not in progress.

4.9.4 Postconditions

a) The joined federate is noted as having reached the specified synchronization point.

4.9.5 Exceptions

a) The synchronization point label is not announced.

b) The federate is not a federation execution member.

c) Save in progress.

d) Restore in progress.

e) RTI internal error.

4.9.6 Reference state charts

a) Figure 7: Awaiting synchronization (s, L).

b) Figure 8: Local synch label (s, L).

4.10 Federation Synchronized †

The *Federation Synchronized †* service shall inform the joined federate that all joined federates in the synchronization set of the specified synchronization point have invoked the *Synchronization Point Achieved* service for that point. This service shall be invoked at all joined federates that are in the synchronization set for that point, indicating that the joined federates in the synchronization set have synchronized at that point. Once the synchronization set for a point synchronizes (the *Federation Synchronized †* service invoked at all

joined federates in the set), that point shall no longer be registered and the synchronization set for that point shall no longer exist.

4.10.1 Supplied arguments

a) Synchronization point label

4.10.2 Returned arguments

a) None

4.10.3 Preconditions

a) The federation execution exists.

b) The federate is joined to that federation execution.

c) The synchronization point has been registered.

d) The synchronization point has been announced at this federate.

e) All joined federates in the synchronization set have invoked *Synchronization Point Achieved* using the specified label.

4.10.4 Postconditions

a) The joined federate is informed that all joined federates in the synchronization set, including it if appropriate, have invoked *Synchronization Point Achieved* using the specified label.

4.10.5 Exceptions

a) Federate internal error

4.10.6 Reference state charts

a) Figure 6: Global synchlabel (L)

b) Figure 8: Local synch label (s, L)

4.11 Request Federation Save

The *Request Federation Save* service shall specify that a federation save should take place.

If the optional time stamp argument is not present, the RTI shall instruct all federation execution members to save state as soon as possible after the invocation of the *Request Federation Save* service.

An optional time stamp argument may be supplied only if the joined federate is time-regulating or if GALT is defined for the joined federate. Supplied time stamp arguments shall be larger than the logical time of all time-constrained joined federates in the federation execution. For time-regulating joined federates this is ensured by only using time stamps that are valid for the joined federate to send in a TSO message (see 8.1 for more information on valid TSO message time stamps). For nonregulating joined federates at which GALT is defined, this can be ensured by only using time stamps that are greater than the joined federate's GALT value.

If a valid optional time stamp argument is present, the RTI shall instruct each time-constrained joined federate to save state (via the *Initiate Federate Save †* service) as soon as the joined federate has received all messages that it will receive as TSO messages that have time stamps less than or equal to the time stamp of the scheduled save[6] and either

— The joined federate is in the Time Advancing state as a result of invocation of either a *Time Advance Request Available*, *Next Message Request Available*, or *Flush Queue Request* service and the logical time of the next grant is greater than the time stamp of the scheduled save, or

— The joined federate is in the Time Advancing state as a result of invocation of either a *Time Advance Request* or *Next Message Request* service and the logical time of the next grant is greater than or equal to the time stamp of the scheduled save

When all time-constrained joined federates are eligible to be instructed to save, the RTI shall instruct all non-time-constrained joined federates to save state.

The RTI shall instruct a joined federate to save state by invoking the *Initiate Federate Save †* service at that joined federate.

At most, one requested save shall be outstanding. A new save request shall replace any outstanding save request. However, a save request cannot happen during a save in progress, which is between the RTI invocation of the *Initiate Federate Save †* service and the RTI invocation of the *Federation Saved †* service.

4.11.1 Supplied arguments

a) Federation save label

b) Optional time stamp

4.11.2 Returned arguments

a) None

4.11.3 Preconditions

a) The federation execution exists.

b) The federate is joined to that federation execution.

c) If an optional time stamp is supplied, either

  1) The joined federate is time-regulating and the time stamp is a valid time stamp for the joined federate to send in a TSO message, or

  2) GALT is defined for the joined federate and the time stamp is greater than the joined federate's GALT value.

d) Save not in progress.

e) Restore not in progress.

4.11.4 Postconditions

a) A federation save has been requested.

b) All previous requested saves are canceled.

[6]This means that the federate's GALT must be greater than the time stamp of the scheduled save.

4.11.5 Exceptions

a) Time stamp has already passed (if optional time stamp argument supplied).

b) Time stamp is invalid (if optional time stamp argument is supplied).

c) Federate may not use the time stamp argument (if optional time stamp argument is supplied).

d) The federate is not a federation execution member.

e) Save in progress.

f) Restore in progress.

g) RTI internal error.

4.11.6 Reference state charts

a) Figure 3: Lifetime of a federate

4.12 Initiate Federate Save †

The *Initiate Federate Save †* service shall instruct the joined federate to save state. The joined federate should save as soon as possible after the invocation of the *Initiate Federate Save †* service. The label provided to the RTI when the save was requested, via the *Request Federation Save* service, shall be supplied to the joined federate. The joined federate shall use this label, the name of the federation execution, its joined federate designator, and its federate type, which it supplied when it invoked the *Join Federation Execution* service, to distinguish the saved state information. If a joined federate is not time constrained, it shall expect to receive an *Initiate Federate Save †* service invocation at any point. If a joined federate is time constrained, it shall expect to receive an *Initiate Federate Save †* service invocation only when it is in the Time Advancing state. The joined federate shall stop providing new information to the federation immediately after receiving the *Initiate Federate Save †* service invocation. The joined federate may resume providing new information to the federation only after receiving the *Federation Saved †* service invocation.

If the corresponding *Request Federation Save* service invocation had a time stamp argument, that time stamp will be provided to the resulting *Initiate Federate Save†* service invocation.

4.12.1 Supplied Arguments

a) Federation save label
b) Optional time stamp

4.12.2 Returned arguments

a) None

4.12.3 Preconditions

a) The federation execution exists.

b) The federate is joined to that federation execution.

c) A federation save has been scheduled.

4.12.4 Postconditions

a) The joined federate has been instructed to begin saving its state.

4.12.5 Exceptions

a) The time stamp is invalid.

b) Unable to perform save.

c) Federate internal error.

4.12.6 Reference state charts

a) Figure 3: Lifetime of a federate

4.13 Federate Save Begun

The *Federate Save Begun* service shall notify the RTI that the joined federate is beginning to save its state.

4.13.1 Supplied arguments

a) None

4.13.2 Returned arguments

a) None

4.13.3 Preconditions

a) The federation execution exists.

b) The federate is joined to that federation execution.

c) The joined federate has received an *Initiate Federate Save †* invocation.

d) The joined federate is ready to start saving its state.

4.13.4 Postconditions

a) The RTI has been informed that the joined federate has begun saving its state.

4.13.5 Exceptions

a) Save not initiated

b) The federate is not a federation execution member

c) Restore in progress

d) RTI internal error

4.13.6 Reference state charts

a) Figure 3: Lifetime of a federate

4.14 Federate Save Complete

The *Federate Save Complete* service shall notify the RTI that the joined federate has completed its save attempt. The save-success indicator shall inform the RTI that the joined federate save either succeeded or failed.

4.14.1 Supplied arguments

a) Federate save-success indicator

4.14.2 Returned arguments

b) None

4.14.3 Preconditions

a) The federation execution exists.

b) The federate is joined to that federation execution.

c) The joined federate has invoked the *Federate Save Begun* service for this save.

d) The joined federate has completed the attempt to save its state.

e) Restore not in progress.

4.14.4 Postconditions

a) The RTI has been informed of the status of the state save attempt.

4.14.5 Exceptions

a) Invalid save-success indicator.

b) Federate has not begun save.

c) The federate is not a federation execution member.

d) Restore in progress.

e) RTI internal error.

4.14.6 Reference state charts

a) Figure 3: Lifetime of a federate

4.15 Federation Saved †

The *Federation Saved †* service shall inform the joined federate that the federation save process is complete, and it shall indicate whether it completed successfully or not. If the save-success indicator argument indicates success, this shall mean that all joined federates at which the *Initiate Federate Save †* service was invoked have invoked the *Federate Save Complete* service with a save-success indicator that indicated success.

If the save-success indicator argument indicates failure, the optional failure reason argument shall be provided to identify the reason that the federation save failed. Possible reasons for the failure of a federation save are as follows:

— RTI was unable to save.

— One or more joined federates at which the *Initiate Federate Save †* service was invoked has invoked the *Federate Save Complete* service with a save-success indicator that indicated failure.

— One or more joined federates at which the *Initiate Federate Save †* service was invoked has resigned from the federation execution.

— The RTI has detected failure at one or more of the joined federates at which the *Initiate Federate Save †* service was invoked.

— The optional time stamp supplied with the save request cannot be honored (due to possible race conditions in the distributed calculation of GALT).

All joined federates that received an invocation of the *Initiate Federate Save †* service shall receive an invocation of the *Federation Saved †* service. Also, a joined federate that has invoked the *Request Federation Save* service may receive an invocation of the *Federation Saved †* service without having first received an invocation of the *Initiate Federate Save †* service.

4.15.1 Supplied arguments

a) Federation save-success indicator

b) Optional failure reason

4.15.2 Returned arguments

a) None

4.15.3 Preconditions

a) The federation execution exists.

b) The federate is joined to that federation execution.

4.15.4 Postconditions

a) The joined federate has been informed of the success or failure of the federation save attempt.

b) The joined federate may resume providing new information to the federation.

4.15.5 Exceptions

a) Federate internal error

4.15.6 Reference state charts

a) Figure 3: Lifetime of a federate

4.16 Query Federation Save Status

The *Query Save Status* service shall be used to request the status of a federation save. The RTI shall provide the federation save status via a *Save Status Response †* service invocation. No save label argument is needed since only one federation save can be in progress at a time.

4.16.1 Supplied arguments

a) None

4.16.2 Returned arguments

a) None

4.16.3 Preconditions

a) The federation execution exists.

b) The federate is joined to that federation execution.

c) Save in progress.

d) Restore not in progress.

4.16.4 Postconditions

a) The request for a federation save status has been received by the RTI.

4.16.5 Exceptions

a) The federate is not a federation execution member.

b) Save not in progress.

c) Restore in progress.

d) RTI internal error.

4.16.6 Reference state charts

a) None

4.17 Federation Save Status Response †

The *Save Status Response †* service shall be used to provide the status of a federation save. This service shall be invoked by the RTI at a joined federate in response to a *Query Save Status* service invocation by the same joined federate. This service shall supply a list of the joined federates participating in the federation save and the save status for each one. A joined federate save status shall be one of the following values:

— No save in progress

— Federate instructed to save

— Federate saving

— Federate waiting for federation to save

4.17.1 Supplied arguments

a) List of joined federates and save status for each

4.17.2 Returned arguments

a) None

4.17.3 Preconditions

a) The federation execution exists.

b) The federate is joined to that federation execution.

4.17.4 Postconditions

a) The joined federate has been informed of the status of the federation save.

4.17.5 Exceptions

a) Federate internal error

4.17.6 Reference state charts

a) None

4.18 Request Federation Restore

The *Request Federation Restore* service shall direct the RTI to begin the federation execution restoration process. Federation restoration shall begin as soon after the validation of the *Request Federation Restore* service invocation as possible. A valid federation restoration request shall be indicated with the *Confirm Federation Restoration Request †* service.

4.18.1 Supplied arguments

a) Federation save label

4.18.2 Returned arguments

a) None

4.18.3 Preconditions

a) The federation execution exists.

b) The federate is joined to that federation execution.

c) The federation has a save with the specified label.

d) The correct number of joined federates of the correct types that were joined to the federation execution when the save was accomplished are currently joined to the federation execution.

e) All previous *Request Federation Restore* service invocations from the joined federate have been acknowledged with a corresponding *Confirm Federation Restoration Request †*.

f) Save not in progress.

g) Restore not in progress.

4.18.4 Postconditions

a) The RTI has been notified of the request to restore a former federation execution state.

4.18.5 Exceptions

a) The federate is not a federation execution member.

b) Save in progress.

c) Restore in progress.

d) RTI internal error.

4.18.6 Reference state charts

a) Figure 3: Lifetime of a federate

4.19 Confirm Federation Restoration Request †

The *Confirm Federation Restoration Request †* shall indicate to the requesting joined federate the status of a requested federation restoration. This service shall be invoked in response to a *Request Federation Restore* service invocation. A positive request success indicator informs the joined federate that the RTI restoration state information has been located that corresponds to the following:

— The indicated label.

— Federation execution name and FDD of this federation execution.

— The census of joined federates matches in number and type the census of joined federates present when the save was taken.

Additionally, no other joined federate is currently attempting to restore the federation. Should more than one joined federate attempt to restore the federation at a given point, at most, one joined federate shall receive a positive indication through this service and all others shall receive a negative indication. A federation restoration attempt that ends with a negative request success indicator shall have no other effect on the federation execution.

4.19.1 Supplied arguments

a) Federation save label

b) Request success indicator

4.19.2 Returned arguments

a) None

4.19.3 Preconditions

a) The federation execution exists.

b) The federate is joined to that federation execution.

c) The joined federate has requested a federation restore via the *Request Federation Restore* service.

4.19.4 Postconditions

a) If the request success indicator is positive, restore in progress.

b) If the request success indicator is positive, the federation has a saved state with the specified label.

c) If the request success indicator is positive, the correct number of joined federates of the correct types that were joined to the federation execution when the save was accomplished are currently joined to the federation execution.

d) If the request success indicator is negative, this service and the corresponding *Request Federation Restore* service invocation have no consequence.

4.19.5 Exceptions

a) Federate internal error

4.19.6 Reference state charts

a) Figure 3: Lifetime of a federate

4.20 Federation Restore Begun †

The *Federation Restore Begun †* service shall inform the joined federate that a federation restoration is imminent. The joined federate shall stop providing new information to the federation immediately after receiving the *Federation Restore Begun †* service invocation. The joined federate may resume providing new information to the federation only after receiving the *Federation Restored †* service invocation. The RTI shall invoke this service at all joined federates, including the joined federate that requested the restore.

4.20.1 Supplied arguments

a) None

4.20.2 Returned arguments

a) None

4.20.3 Preconditions

a) The federation execution exists.

b) The federate is joined to that federation execution.

4.20.4 Postconditions

a) The joined federate has been instructed to stop providing new information to the federation.

4.20.5 Exceptions

a) Federate internal error

4.20.6 Reference state charts

a) Figure 3: Lifetime of a federate

4.21 Initiate Federate Restore †

The *Initiate Federate Restore †* service shall instruct the joined federate to return to a previously saved state. The joined federate shall select the appropriate restoration state information based on the name of the current federation execution, the supplied federation save label, and the supplied joined federate designator. As a result of this service invocation, a joined federate's designator could change from the value supplied by the *Join Federation Execution* service.

4.21.1 Supplied arguments

a) Federation save label

b) Joined federate designator

4.21.2 Returned arguments

a) None

4.21.3 Preconditions

a) The federation execution exists.

b) The federate is joined to that federation execution.

c) The joined federate has a save with the specified label.

4.21.4 Postconditions

a) The joined federate has been informed to begin restoring state.

b) The joined federate's designator may be changed.

4.21.5 Exceptions

a) There is no federate save associated with the label

b) Could not initiate restore

c) Federate internal error

4.21.6 Reference state charts

a) Figure 3: Lifetime of a federate

4.22 Federate Restore Complete

The *Federate Restore Complete* service shall notify the RTI that the joined federate has completed its restore attempt. If restore was successful, the joined federate shall be in the state that either it or some other joined federate of its type was in when the federation save associated with the label occurred, with the distinction that the joined federate shall now be waiting for an invocation of the *Federation Restored †* service.

4.22.1 Supplied arguments

a) Federate restore-success indicator

4.22.2 Returned arguments

a) None

4.22.3 Preconditions

a) The federation execution exists.

b) The federate is joined to that federation execution.

c) The joined federate was directed to restore through invocation of the *Initiate Restore* service.

d) If restore was successful, the joined federate is in a state identical to the state that either it or some other joined federate of its type was in when the federation save associated with the supplied label occurred, with the distinction that the joined federate is now waiting for an invocation of the *Federation Restored †* service. If restore was unsuccessful, the joined federate is in an undefined state.

e) Save not in progress.

4.22.4 Postconditions

a) The RTI has been informed of the status of the restore attempt.

4.22.5 Exceptions

a) Invalid restore-success indicator.

b) Restore not requested.

c) The federate is not a federation execution member.

d) Save in progress.

e) RTI internal error.

4.22.6 Reference state charts

a) Figure 3: Lifetime of a federate

4.23 Federation Restored †

The *Federation Restored †* service shall inform the joined federate that the federation restore process is complete, and it shall indicate whether it completed successfully or not. If the restore-success indicator argument indicates success, this shall mean that all joined federates at which the *Federation Restore Begun †* service was invoked have invoked the *Federate Restore Complete* service with a restore-success indicator that indicated success.

If the restore-success indicator argument indicates failure, the optional failure reason argument shall be provided to identify the reason that the federation restore failed. Possible reasons for the failure of a federation restore are as follows:

— The RTI was unable to restore.

— One or more joined federates at which the *Federation Restore Begun †* service was invoked have invoked the *Federate Restore Complete* service with a restore-success indicator that indicated failure.

— One or more joined federates at which the *Federation Restore Begun †* service was invoked have resigned.

— The RTI detected failure at one or more of the joined federates at which the *Federation Restore Begun †* service was invoked.

All joined federates that received an invocation of the *Federation Restore Begun †* service shall receive an invocation of the *Federation Restored †* service. If a joined federate that received an invocation of the *Federation Restore Begun †* service resigns from the federation execution before the *Federation Restored †* service for that restore is invoked, this resignation shall be considered a failure of the federation restoration, and the *Federation Restored †* service shall be invoked with a restore-success indicator of failure.

4.23.1 Supplied arguments

a) Federation restore-success indicator

b) Optional failure reason

4.23.2 Returned arguments

a) None

4.23.3 Preconditions

a) The federation execution exists.

b) The federate is joined to that federation execution.

c) The joined federate has a save with the specified label.

4.23.4 Postconditions

a) The joined federate has been informed regarding the success or failure of the restoration attempt.

b) The joined federate may resume providing new information to the federation.

4.23.5 Exceptions

a) Federate internal error

4.23.6 Reference state charts

a) Figure 3: Lifetime of a federate

4.24 Query Federation Restore Status

The *Query Restore Status* service shall be used to request the status of a federation restore. The RTI shall provide the federation restore status via a *Restore Status Response †* service invocation. No save label argument is needed because only one federation restore can be in progress at a time.

4.24.1 Supplied arguments

a) None

4.24.2 Returned arguments

a) None

4.24.3 Preconditions

a) The federation execution exists.

b) The federate is joined to that federation execution.

c) Save not in progress.

d) Restore in progress.

4.24.4 Postconditions

a) The request for a federation restore status has been received by the RTI.

4.24.5 Exceptions

a) The federate is not a federation execution member.

b) Save in progress.

c) Restore not in progress.

d) RTI internal error.

4.24.6 Reference state charts

a) None

4.25 Federation Restore Status Response †

The *Restore Status Response †* service shall be used to provide the status of a federation restore. This service shall be invoked by the RTI at a joined federate in response to a *Query Restore Status* service invocation by the same joined federate. This service shall supply a list of the joined federates participating in the federation restore and the restore status for each one. A joined federate restore status shall be one of the following values:

— No restore in progress

— Federate restore request pending

— Federate waiting for restore to begin

— Federate prepared to restore

— Federate restoring

— Federate waiting for federation to restore

4.25.1 Supplied arguments

a) List of joined federates and restore status for each

4.25.2 Returned arguments

a) None

4.25.3 Preconditions

a) The federation execution exists.

b) The federate is joined to that federation execution.

4.25.4 Postconditions

a) The joined federate has been informed of the status of the federation restore.

4.25.5 Exceptions

a) Federate internal error

4.25.6 Reference state charts

a) None

5. Declaration management

5.1 Overview

Joined federates shall use DM services to declare their intention to generate information. A joined federate shall invoke appropriate DM services before it may register object instances, update instance attribute values, and send interactions. Joined federates shall use DM services or DDM services to declare their intention to receive information. (A joined federate may use DM services exclusively, DDM services exclusively, or both DM and DDM services to declare its intention to receive information. This clause describes how DM services work when they are used exclusively by a joined federate. See 9.1.5 for more information on using DDM services in lieu of or in conjunction with DM services.) A joined federate shall invoke appropriate DM or DDM services before it may discover object instances, reflect instance attribute values, and receive interactions. DM and DDM services, together with object management services, ownership management services, and the object class and interaction class hierarchies defined in the FDD shall determine the following:

— Object classes at which object instances may be registered

— Object classes at which object instances are discovered

— Instance attributes that are available to be updated and reflected

— Interactions that may be sent

— Interaction classes at which interactions are received

— Parameters that are available to be sent and received

The effects of DM services shall be independent of the logical time of any joined federate in the federation execution.

5.1.1 Static properties of the FDD

The following static properties of the FDD shall establish vocabulary for subsequent DM discussion:

b) Every class shall have at most one immediate superclass. A class shall not be a superclass of a class that is its superclass.

c) Every object class shall have an associated set of class attributes declared in the FDD.

d) An **inherited attribute** of an object class is a class attribute that was declared in a superclass.

e) The **available attributes** of an object class are the set of declared attributes of that object class in union with the set of inherited attributes of that object class.

f) Every interaction class shall have an associated set of parameters declared in the FDD.

g) An **inherited parameter** of an interaction class is a parameter that was declared in a superclass.

h) The **available parameters** of an interaction class are the set of declared parameters of that interaction class in union with the set of inherited parameters of that interaction class.

i) For any service that takes an object class and a set of class attribute designators as arguments, only the available attributes of that object class may be used in the set of attribute designators. Being an available attribute of an object class shall be a necessary, but not necessarily a sufficient, condition for an attribute to be used in the set of attribute designators for such a service.

j) For any service that takes an object instance and a set of attribute designators as arguments, only the available attributes of that object instance's known class at the involved (invoking or invoked)

joined federate may be used in the set of attribute designators. Being an available attribute of the object instance's known class shall be a necessary, but not necessarily a sufficient, condition for an attribute to be used in the set of attribute designators for such a service.

5.1.2 Definitions and constraints for object classes and class attributes

The following DM definitions and constraints shall pertain to object classes and class attributes as declared in the class hierarchy of the FDD:

a) A class attribute may be used as an argument to *Subscribe Object Class Attributes*, *Unsubscribe Object Class Attributes, Publish Object Class Attributes*, and *Unpublish Object Class Attributes* service invocations for a particular object class if and only if the attribute is an available attribute of that object class.

b) From a joined federate's perspective, the **subscribed attributes of an object class** shall be all of the class attributes that have been arguments to *Subscribe Object Class Attributes* service invocations by that joined federate for that object class, and that have not subsequently been unsubscribed, either individually or by unsubscribing the whole object class. *Subscribe Object Class Attributes* and *Unsubscribe Object Class Attributes* service invocations for one object class shall have no effect on the subscribed attributes of any other object class.

c) If a class attribute is a subscribed attribute of an object class, the joined federate shall be subscribed to that class attribute either actively or passively, but not both.

d) From a joined federate's perspective, the **explicitly published attributes of an object class** shall be all the class attributes that have been arguments to *Publish Object Class Attributes* service invocations by that joined federate for that object class, and that have not subsequently been unpublished, either individually or by unpublishing the whole object class.

e) When there are no explicitly published attributes of an object class for a given joined federate, the first *Publish Object Class Attributes* service invocation that results in explicitly published attributes for that object class for that joined federate also results in **publication being established** for that object class at that joined federate.

f) The **HLAprivilegeToDeleteObject** class attribute shall be implicitly published for a joined federate for an object class, if and only if

   1) The **HLAprivilegeToDeleteObject** class attribute for the object class is not, nor has been since publication of the object class was most recently established, explicitly published by the joined federate.

   2) The **HLAprivilegeToDeleteObject** class attribute for the object class has not been explicitly unpublished by the joined federate since publication of the object class was most recently established.

   3) At least one attribute of the object class is explicitly published by the joined federate.

g) From a joined federate's perspective, the published attributes of an object class shall be the explicitly published and the implicitly published, if any, attributes of the object class for the joined federate. *Publish Object Class Attributes* and *Unpublish Object Class Attributes* service invocations for one object class shall have no effect on the published attributes of any other object class.

h) From a joined federate's perspective, an object class shall be **subscribed** if and only if the joined federate is subscribed to at least one attribute of the object class.

i) From a joined federate's perspective, an object class shall be **published** if and only if at least one attribute of the object class is published by the joined federate.

j) Joined federates may invoke the *Register Object Instance* service only with a published object class as an argument.

k) The **registered class** of an object instance shall be the object class that was an argument to the *Register Object Instance* service invocation for that object instance.

l) Every object instance shall have one federation-wide registered class that cannot change.

m) If the *Discover Object Instance †* service is invoked at a joined federate, the object instance discovered as a result of this service invocation shall have a **discovered class** at that joined federate. The discovered class of the object instance shall be a supplied argument to the *Discover Object Instance †* service invocation.

n) An object instance may have at most one discovered class in each joined federate. This discovered class may vary from joined federate to joined federate. Once an object instance is discovered, its discovered class shall not change. If a joined federate invokes the *Local Delete Object Instance* service for a given object instance, that object instance may be newly discovered, at a possibly different object class than previously known.

o) If a joined federate has registered or discovered an object instance and it has not subsequently

1) Invoked the *Local Delete Object Instance* service for that object instance

2) Invoked the *Delete Object Instance* service for that object instance, or

3) Received an invocation of the *Remove Object Instance †* service for that object instance

the object instance shall be known to that joined federate, and that object instance has a known class at that joined federate. The known class of that object instance at that joined federate shall be the object instance's registered class if the joined federate knows about the object instance as a result of having registered it. The known class of that object instance at that joined federate shall be the object instance's discovered class if the joined federate knows about the object instance as a result of having discovered it.

p) From a joined federate's perspective, the corresponding class attribute of an instance attribute is the class attribute of the joined federate's known class for the object instance containing the instance attribute that has the same attribute designator as the instance attribute.

q) From a joined federate's perspective, the corresponding instance attributes of a class attribute are

1) All unowned instance attributes of object instances that have a known class at the joined federate equal to the object class of the class attribute and that have the same attribute designator as the class attribute, and

2) All instance attributes owned by the joined federate that belong to object instances that have a known class at the owning federate equal to the object class of the class attribute and that have the same attribute designator as the class attribute.

r) A joined federate may own and update only an instance attribute for which it is publishing the corresponding class attribute.

s) An update to an instance attribute by the joined federate that owns that instance attribute may be reflected only by other joined federates that are subscribed to the instance attribute's corresponding (from each subscriber's perspective) class attribute.

5.1.3 Definitions and constraints for interaction classes and parameters

The following DM definitions and constraints shall pertain to interaction classes and parameters as declared in the interaction class hierarchy of the FDD:

a) From a joined federate's perspective, an interaction class shall be **subscribed** if and only if it was an argument to a *Subscribe Interaction Class* service invocation by that joined federate that was not subsequently followed by an *Unsubscribe Interaction Class* service invocation for that interaction class.

b) If an interaction class is subscribed, the joined federate shall be subscribed to that interaction class either actively or passively, but not both.

c) From a joined federate's perspective, an interaction class shall be **published** if and only if it was an argument to a *Publish Interaction Class* service invocation by that joined federate that was not subsequently followed by an *Unpublish Interaction Class* service invocation for that interaction class.

d) Joined federates may invoke the *Send Interaction* service only with a published interaction class as an argument.

e) The sent class of an interaction shall be the interaction class that was an argument to the *Send Interaction* service invocation for that interaction.

f) Every interaction shall have one federation-wide sent class.

g) The *Receive Interaction †* service shall be invoked at a joined federate only with a subscribed interaction class as an argument.

h) If the *Receive Interaction †* service is invoked at a joined federate, the interaction received as a result of this service invocation shall have a **received class** at that joined federate. The received class of an interaction is the interaction class that is an argument to the *Receive Interaction †* service invocation.

i) An interaction may have at most one received class in each joined federate. This received class may vary from joined federate to joined federate.

j) Only the available parameters of an interaction class may be used in a *Send Interaction* service invocation with that interaction class as argument.

k) The sent parameters of an interaction shall be the parameters that were arguments to the *Send Interaction* service invocation for that interaction.

l) The received parameters of an interaction shall be the parameters that were arguments to the *Receive Interaction †* service invocation for that interaction.

m) The received parameters of an interaction shall be the subset of the sent parameters that are available parameters for the interaction's received class.

n) The received parameters for a given interaction may vary from joined federate to joined federate, depending on the received class of the interaction.

When an object instance's discovered class is a superclass of its registered class, the object instance shall be said to have been promoted from the registered class to the discovered class. Similarly, when an interaction's received class is a superclass of its sent class, the interaction shall be said to have been promoted from the sent class to the received class. Promotion is important for protecting federate application code from new subclasses added to the FDD. As the FDD is expanded to include new object and interaction classes that inherit from existing classes, promotion ensures that existing federate application code need not change to work with the expanded FDD.

5.1.4 Declaration management statecharts

The following figures depict formal representations of the state of an arbitrary object class, an arbitrary class attribute, the **HLAprivilegeToDeleteObject** class attribute of an arbitrary object class, and an arbitrary interaction class. Figure 9 depicts the state of an arbitrary object class, and it deals with object classes at two levels. First, it establishes that each class attribute of the object class has some state worth modeling. Second, it establishes that there may be an arbitrary number of instances of each object class. Further, it defines what conditions allow an object instance to be known by a joined federate as an instance of that object class.

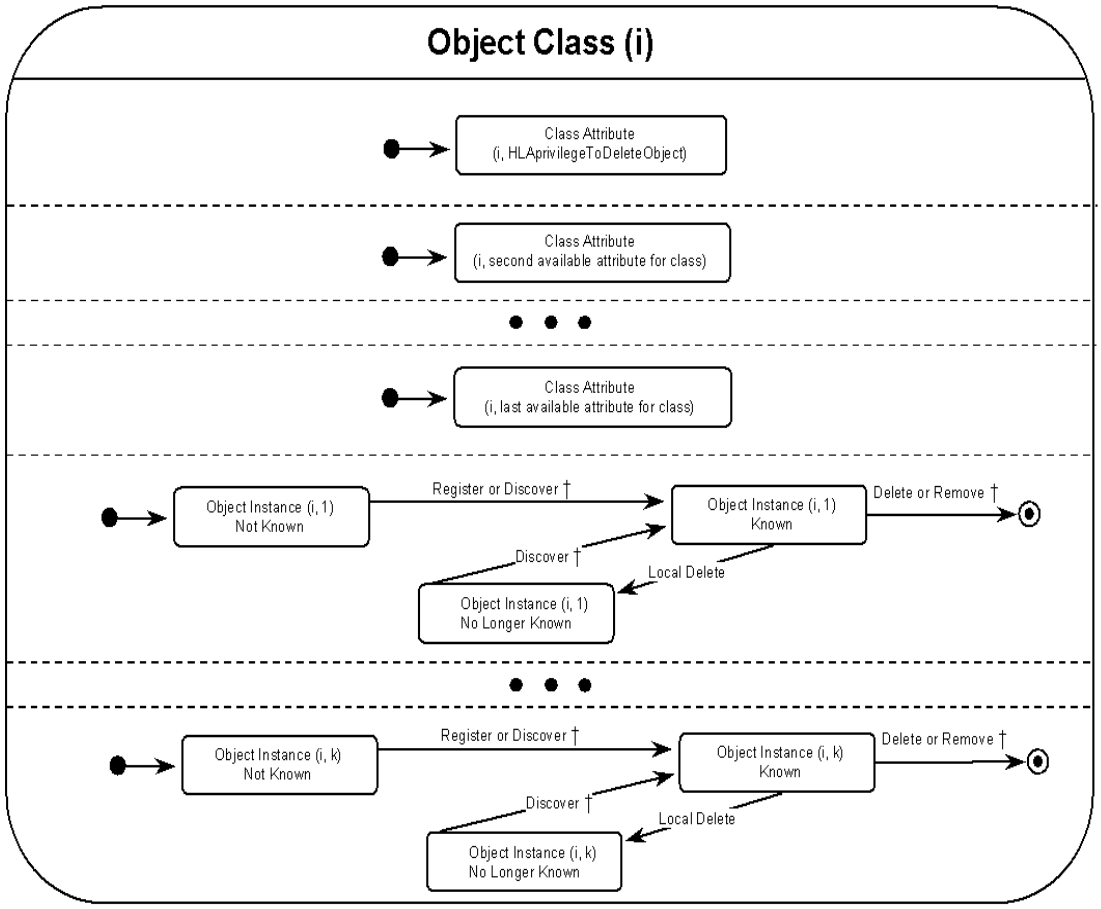

Figure 9—Object class (i)

Conceptually, the state of an object class comprises the state of the class attributes of that object class and of the object instances of that object class. Furthermore, the state of an object instance comprises the state of the instance attributes of that object instance. There is a correspondence between the instance attributes and their corresponding class attributes. This correspondence is modeled via the index to each attribute. A reference to an instance attribute (*i, k, j*) and to a class attribute (*i, j*) with the same *i*s and same *j*s means that the class attribute is that instance attribute's corresponding class attribute at that joined federate. This is because having the same *j*s means that they refer to the same attribute designator, and having the same *i*s means that the joined federate's known class for the object instance containing the instance attribute is the same as the object class designator of the class attribute.

Each object class shall have a fixed number of available class attributes as defined in the FDD. The number of object instances of a given class, however, shall be arbitrary.

An object instance of an object class shall become known by the registering joined federate when the object instance is registered. It may become known by other joined federates in the federation execution. If it becomes known by other joined federates in the federation execution, it shall become known by them as a result of being discovered.

Figure 10 depicts the state of an arbitrary class attribute (that is not the **HLAprivilegeToDeleteObject** class attribute of an object class) and shows the properties that may be controlled by a joined federate at the class

attribute level. Specifically, a joined federate may publish or subscribe to class attributes. Although the *Publish Object Class Attributes* and *Subscribe Object Class Attributes* service invocations can take sets of class attributes as an argument, Figure 10 depicts only a single class attribute. So, for example, *Publish (i, j)* shall mean that attribute *j* of object class *i* was an element of the set used as an argument to the *Publish Object Class Attributes* service.

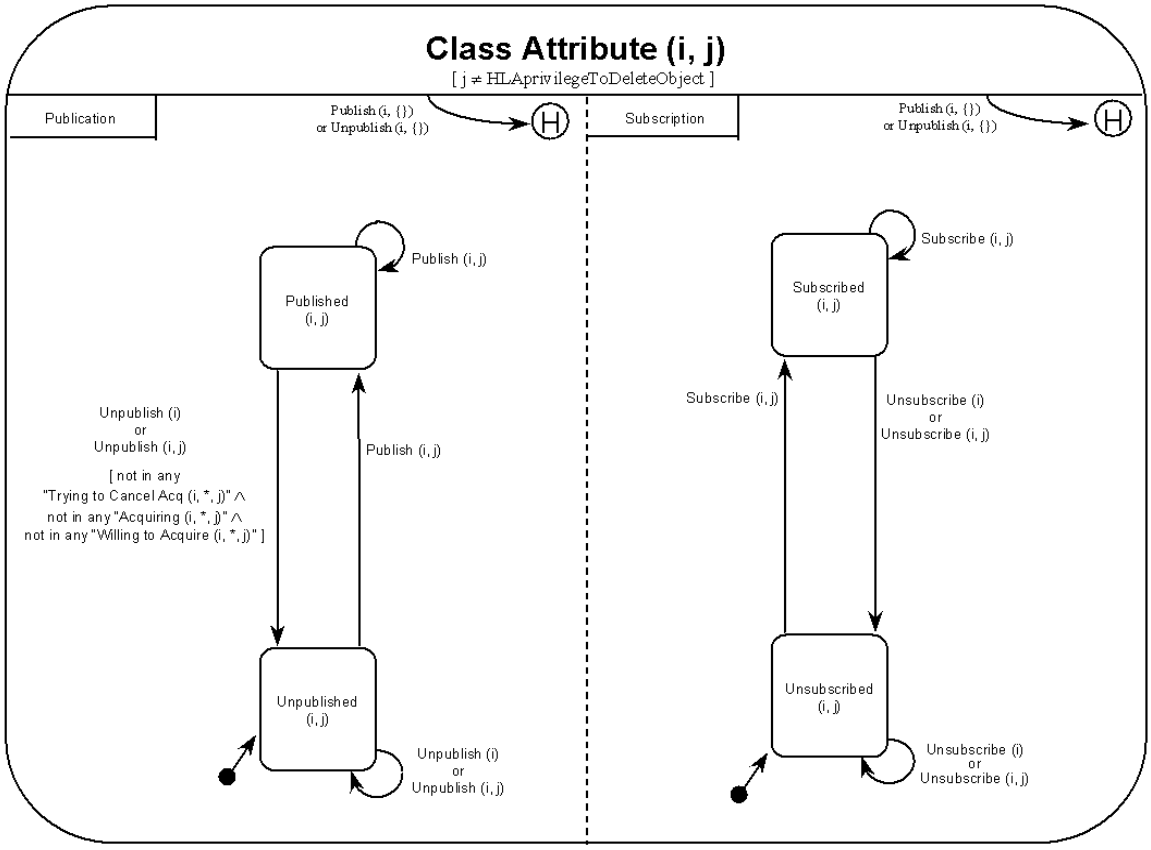

Figure 10—Class attribute (i, j)

The **HLAprivilegeToDeleteObject** class attribute of an arbitrary object class is treated slightly differently from other class attributes, and it is depicted in Figure 11. Unlike other class attributes, **HLAprivilegeToDeleteObject**, in addition to explicit publish/unpublish actions, may be implicitly published and unpublished for a given object class at a given federate if other circumstances are met (as shown in the statechart and described in 5.1.2). For purposes of subscription, the **HLAprivilegeToDeleteObject** class attribute of an arbitrary object class is treated no differently than any other class attribute.

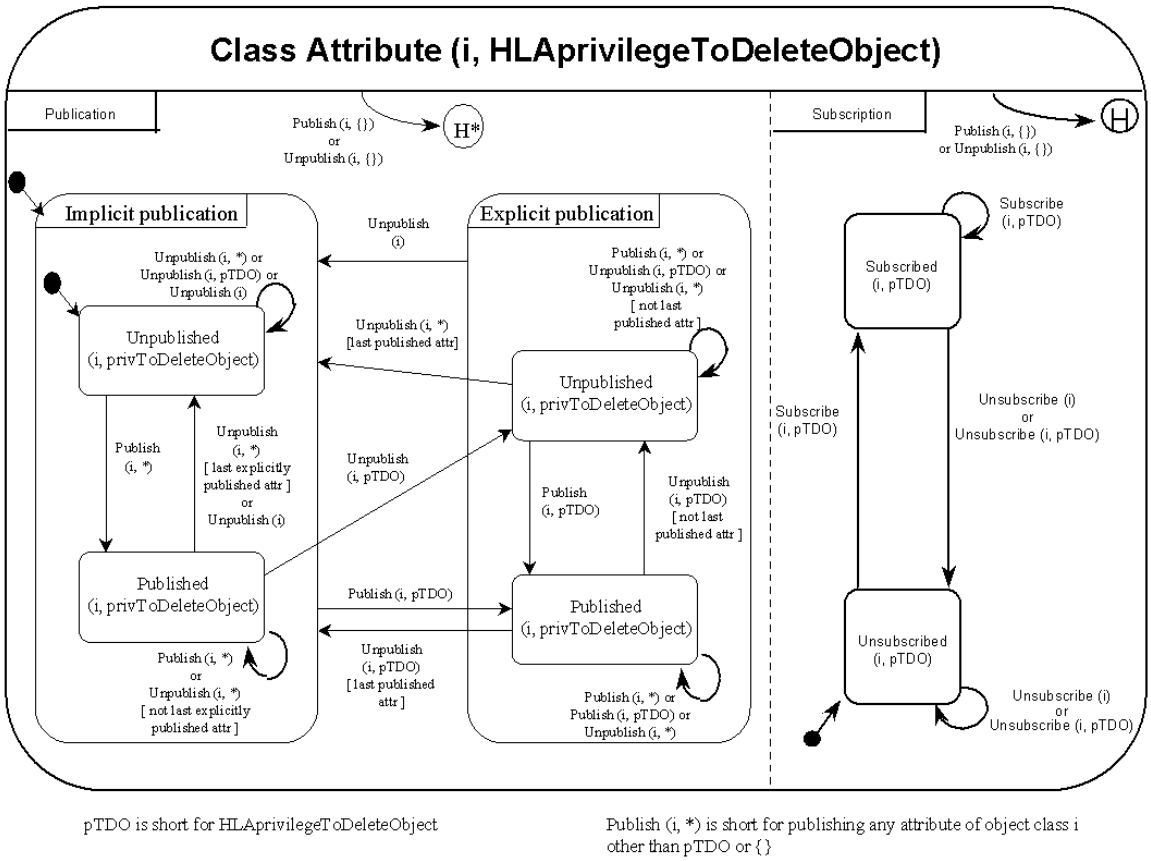

Figure 11—Class attribute (i, HLAprivilegeToDeleteObject)

Figure 12 depicts the state of an arbitrary interaction class and shows the properties relating to interaction classes that may be controlled by a joined federate. Specifically, a joined federate may publish or subscribe to interaction classes.

The joined federate may also direct the RTI via the *Enable/Disable Interaction Relevance Advisory Switch* services to indicate that the joined federate does or does not want the RTI to use the *Turn Interactions On †* and *Turn Interactions Off †* services to inform the joined federate when interactions of a given class are relevant to the other joined federates in the federation execution.

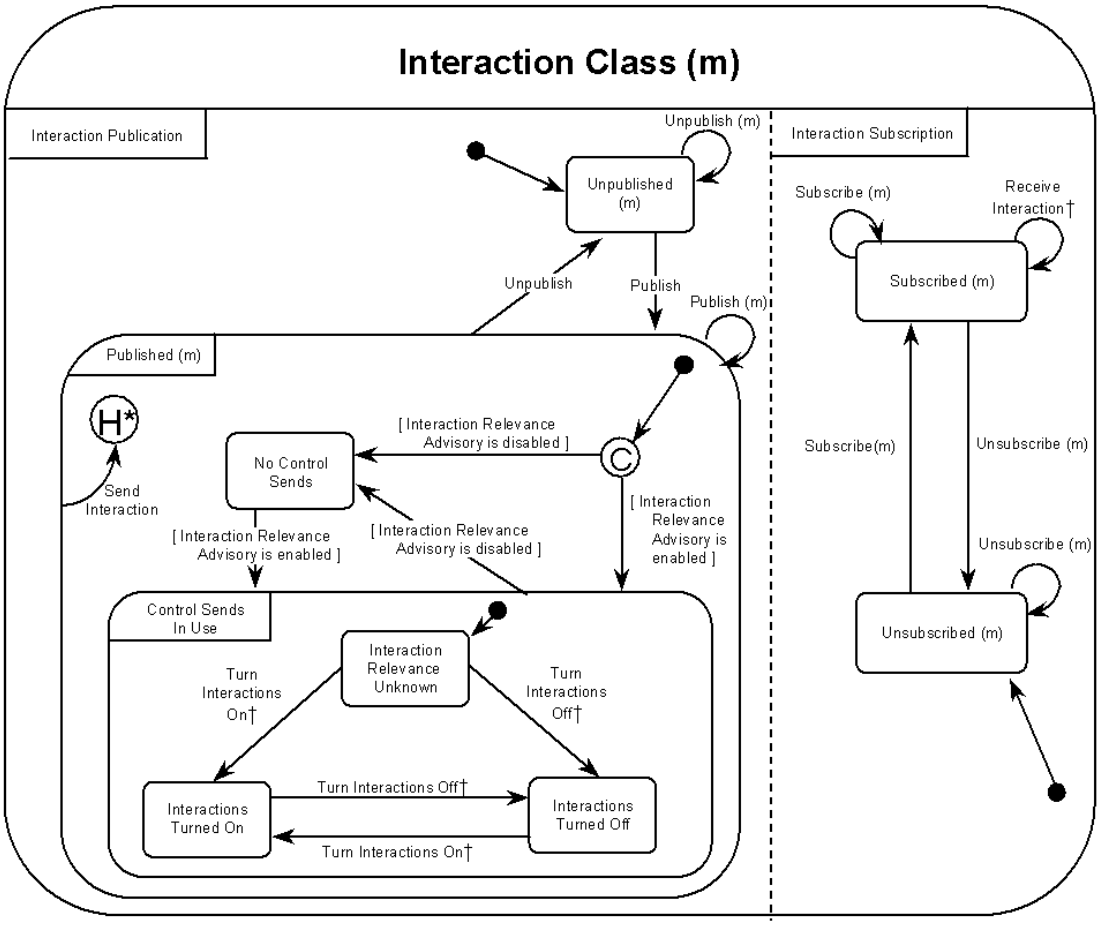

Figure 12—Interaction class (m)

5.1.5 Interaction with DDM services

A joined federate may use DM services, and it may also use DDM services. Joined federates that use DM services exclusively may be joined to the same federation execution as joined federates that use DM services exclusively, joined federates that use DDM services exclusively, and joined federates that use both DM services and DDM services. This clause describes how DM services work when they are used in the absence of the use of DDM services by a joined federate, from the perspective of that joined federate, regardless of whether other joined federates in the federation are using DM services exclusively, DDM services exclusively, or both DM services and DDM services. When both DM services and DDM services are used by a single joined federate, some of the terms and services defined in this clause are extended. See 9.1.5 for an expanded interpretation of how selected DM services work when they are used in conjunction with DDM services by a joined federate, from the perspective of that joined federate.

5.2 Publish Object Class Attributes

The information provided by the joined federate via the *Publish Object Class Attributes* service shall be used in multiple ways. First, it shall indicate an object class of which the joined federate may subsequently register object instances. Second, it shall indicate the class attributes of the object class for which the joined federate is capable of owning the corresponding instance attributes. Only the joined federate that owns an instance attribute shall provide values for that instance attribute to the federation. The joined federate may

become the owner of an instance attribute and thereby be capable of updating its value in the following two ways:

— By registering an object instance of a published class. Upon registration of an object instance, the registering joined federate shall become the owner of all instance attributes of that object instance for which the joined federate is publishing the corresponding class attributes at the registered class of the object instance.

— By using ownership management services to acquire instance attributes of object instances. The joined federate may acquire only those instance attributes for which the joined federate is publishing the corresponding class attributes.

Each use of this service shall add to the publications specified to the RTI in previous service invocations for the same object class. Invoking this service with an empty set of class attributes shall be equivalent to adding no publications for the object class.

5.2.1 Supplied arguments

a) Object class designator
b) Set of attribute designators

5.2.2 Returned arguments

a) None

5.2.3 Preconditions

a) The federation execution exists.

b) The federate is joined to that federation execution.

c) The specified object class is defined in the FDD.

d) The specified class attributes are available attributes of the specified object class.

e) Save not in progress.

f) Restore not in progress.

5.2.4 Postconditions

a) The specified class attributes shall now be published attributes of the specified object class for the joined federate.

b) If at least one attribute of the specified class is published by the joined federate, it may register object instances of the specified class.

c) If at least one attribute of the specified class is published by the joined federate, the **HLAprivilegeToDeleteObject** class attribute of the specified class may be implicitly published for the joined federate.

d) The joined federate may own instance attributes that correspond to the specified class attributes.

5.2.5 Exceptions

a) The object class is not defined in the FDD.

b) The specified class attributes are not available attributes of the specified object class.

c) The federate is not a federation execution member.

d) Save in progress.

e) Restore in progress.

f) RTI internal error.

5.2.6 Reference state charts

a) Figure 10: Class attribute (i, j)

b) Figure 11: Class attribute (i, HLAprivilegeToDeleteObject)

5.3 Unpublish Object Class Attributes

The *Unpublish Object Class Attributes* service can be used in multiple ways. First, it could be used to unpublish a whole object class, informing the RTI that the joined federate shall no longer be capable of registering object instances of the specified object class. Second, it could be used to unpublish specific class attributes, informing the RTI that the joined federate shall no longer be capable of owning instance attributes that correspond to the unpublished class attributes.

If the optional set of attributes designators argument is supplied, only those class attributes shall be unpublished for the joined federate. Otherwise, all published class attributes of the specified class shall be unpublished for the joined federate. Invoking this service with an empty set of class attributes shall be equivalent to removing no publications for the object class.

The joined federate shall lose ownership of all owned instance attributes whose corresponding class attributes are unpublished by the service invocation, which means that the joined federate shall no longer be able to update such instance attributes.

5.3.1 Arguments

a) Object class designator

b) Optional set of attribute designators

5.3.2 Returned arguments

a) None

5.3.3 Preconditions

a) The federation execution exists.

b) The federate is joined to that federation execution.

c) The object class is defined in the FDD.

d) If the optional set of attribute designators argument is supplied, the specified class attributes are available attributes of the specified object class.

e) For each class attribute of the specified class that is published by the joined federate and is to be unpublished by this service invocation, there are no joined federate-owned corresponding instance attributes for which the joined federate has either

1) invoked the *Attribute Ownership Acquisition* service, and has not yet received a corresponding invocation of either the *Confirm Attribute Ownership Acquisition Cancellation †* service or the *Attribute Ownership Acquisition Notification †* service, or

2) invoked the *Attribute Ownership Acquisition If Available* service, and has not yet received a corresponding invocation of either the *Attribute Ownership Unavailable †* service or the *Attribute Ownership Acquisition Notification †* service, or

3) invoked the *Attribute Ownership Acquisition If Available* service and has subsequently invoked the *Attribute Ownership Acquisition* service [after which condition 1) applies].

f) Save not in progress.

g) Restore not in progress.

5.3.4 Postconditions

a) If the optional set of attribute designators argument is supplied, the specified attributes are not published by the joined federate for the specified object class.

b) If the optional set of attribute designators argument is supplied and there are no longer any attributes of the class that are explicitly published by the joined federate, the **HLAprivilegeToDeleteObject** class attribute of the specified object class may be implicitly unpublished for the joined federate.

c) If the optional set of attribute designators is not supplied, no class attributes of the object class are published by the joined federate.

d) If there are no class attributes of the specified class that are still published by the joined federate, it shall not be able to register object instances of the specified object class.

e) The joined federate shall no longer own any instance attributes whose corresponding class attributes are no longer published by the joined federate.

5.3.5 Exceptions

a) The object class is not defined in the FDD.

b) The specified class attributes are not available attributes of the specified object class (if the optional set of attribute designators argument is supplied).

c) Cannot unpublish due to pending attempt to acquire instance attribute ownership.

d) The federate is not a federation execution member.

e) Save in progress.

f) Restore in progress.

g) RTI internal error.

5.3.6 Reference state charts

a) Figure 10: Class attribute (i, j)

b) Figure 11: Class attribute (i, HLAprivilegeToDeleteObject)

5.4 Publish Interaction Class

The *Publish Interaction Class* service shall inform the RTI of the classes of interactions that the joined federate will send to the federation execution.

5.4.1 Supplied arguments

a) Interaction class designator

5.4.2 Returned arguments

a) None

5.4.3 Preconditions

a) The federation execution exists.

b) The federate is joined to that federation execution.

c) The interaction class is specified in the FDD.

d) Save not in progress.

e) Restore not in progress.

5.4.4 Postconditions

a) The joined federate may now send interactions of the specified class.

5.4.5 Exceptions

a) The interaction class is not defined in the FDD.

b) The federate is not a federation execution member.

c) Save in progress.

d) Restore in progress.

e) RTI internal error.

5.4.6 Reference state charts

a) Figure 12: Interaction class (m)

5.5 Unpublish Interaction Class

The *Unpublish Interaction Class* service shall inform the RTI that the joined federate will no longer send interactions of the specified class.

5.5.1 Supplied arguments

a) Interaction class designator

5.5.2 Returned arguments

a) None

5.5.3 Preconditions

a) The federation execution exists.

b) The federate is joined to that federation execution.

c) The interaction class is specified in the FDD.

d) Save not in progress.

e) Restore not in progress.

5.5.4 Postconditions

a) The joined federate may not send interactions of the specified interaction class.

5.5.5 Exceptions

a) The interaction class is not defined in the FDD.

b) The federate is not a federation execution member.

c) Save in progress.

d) Restore in progress.

e) RTI internal error.

5.5.6 Reference state charts

a) Figure 12: Interaction class (m)

5.6 Subscribe Object Class Attributes

The *Subscribe Object Class Attributes* service shall specify an object class at which the RTI shall notify the joined federate of discovery of object instances. When subscribing to an object class, the joined federate shall also provide a set of class attributes. The values of only the instance attributes that correspond to the specified class attributes, for all object instances discovered as a result of this service invocation, shall be provided to the joined federate from the RTI (via the *Reflect Attribute Values †* service). The set of class attributes provided shall be a subset of the available attributes of the specified object class.

A joined federate shall only discover an object instance as being of a class to which the joined federate is subscribed.

If a joined federate subscribes to multiple locations in an object class inheritance tree, each relevant object instance registration shall result in at most one object instance discovery by the subscribing joined federate. The discovered class shall be the registered class, if subscribed by the discovering joined federate. Otherwise, the discovered class shall be the closest superclass of the registered class subscribed by the discovering joined federate.

Each use of this service shall add to the subscriptions specified to the RTI in any previous *Subscribe Object Class Attributes* service invocation for the same object class. Invoking this service with an empty set of class attributes shall be equivalent to adding no subscriptions for the specified object class.

If a subscription is provided for a class attribute that is already subscribed by the joined federate, the subscription shall take on the effect of the optional passive subscription indicator from the most recent *Subscribe Object Class Attributes* service invocation.

If the optional passive subscription indicator argument indicates that this is a passive subscription

a) The invocation of this service shall not cause the *Start Registration For Object Class †* service to be invoked at any other joined federate, and

b) If this invocation replaces a previous subscription that was active rather than passive, invocation of this service may cause the *Stop Registration for Object Class †* service to be invoked at one or more other joined federates

If the optional passive subscription indicator argument is not present or indicates that this is an active subscription, the invocation of this service may cause the *Start Registration For Object Class* service to be invoked at one or more other joined federates.

5.6.1 Supplied arguments

a) Object class designator

b) Set of attribute designators

c) Optional passive subscription indicator

5.6.2 Returned arguments

a) None

5.6.3 Preconditions

a) The federation execution exists.

b) The federate is joined to that federation execution.

c) The specified object class is defined in the FDD.

d) The specified class attributes are available attributes of the specified object class.

e) Save not in progress.

f) Restore not in progress.

5.6.4 Postconditions

a) The RTI has been informed of the joined federate's requested subscription.

5.6.5 Exceptions

a) The object class is not defined in the FDD.

b) The specified class attributes are not available attributes of the specified object class.

c) Invalid passive subscription indicator (if the optional passive subscription indicator argument is supplied).

d) The federate is not a federation execution member.

e) Save in progress.

f) Restore in progress.

g) RTI internal error.

5.6.6 Reference state charts

a) Figure 10: Class attribute (i, j)

b) Figure 11: Class attribute (i, HLAprivilegeToDeleteObject)

5.7 Unsubscribe Object Class Attributes

The *Unsubscribe Object Class Attributes* service can be used in multiple ways. First, it could be used to unsubscribe a whole class, informing the RTI that it shall stop notifying the joined federate of object instance discovery at the specified object class. Second, it could be used to unsubscribe specific class attributes, informing the RTI that the joined federate shall no longer receive values for any of the unsubscribed class attributes corresponding instance attributes.

If the optional set of attribute designators argument is supplied, only those class attributes shall be unsubscribed for the joined federate. Otherwise, all subscribed class attributes of the specified class shall be unsubscribed for the joined federate. Invoking this service with an empty set of class attributes shall be equivalent to removing no subscriptions for the object class.

All in-scope instance attributes whose corresponding class attributes are unsubscribed by the service invocation shall go out of scope. Refer to 9.1.1 for an expanded interpretation of this service when a joined federate is using DDM services in conjunction with DM services.

Invocation of this service, subject to other conditions, may cause an implicit out-of-scope situation for affected instance attributes, i.e. invoking joined federate will NOT be notified via an invocation of the *Attributes Out Of Scope †* service invocation when the instance attribute goes out-of-scope.

5.7.1 Supplied arguments

a) Object class designator

b) Optional set of attribute designators

5.7.2 Returned arguments

a) None

5.7.3 Preconditions

a) The federation execution exists.

b) The federate is joined to that federation execution.

c) The object class is defined in the FDD.

d) If the optional set of attribute designators argument is supplied, the specified class attributes are available attributes of the specified object class.

e) Save not in progress.

f) Restore not in progress.

5.7.4 Postconditions

a) If the optional set of attribute designators argument is supplied, the specified attributes are not subscribed by the joined federate.

b) If the optional set of attribute designators argument is not supplied, no class attributes of the object class are subscribed by the joined federate.

c) If there are no class attributes of the specified class that are still subscribed by the joined federate, the joined federate shall receive no subsequent *Discover Object Instance* service invocations for the specified object class.

d) The joined federate shall receive no subsequent *Reflect Attribute Values †* service invocations for any instance attributes whose corresponding class attributes are no longer subscribed by the joined federate.

e) May cause an implicit out-of-scope situation for affected instance attributes at invoking joined federate.

5.7.5 Exceptions

a) The object class is not defined in the FDD.

b) The specified class attributes are not available attributes of the specified object class (if the optional set of attribute designators argument is supplied).

c) The federate is not a federation execution member.

d) Save in progress.

e) Restore in progress.

f) RTI internal error.

5.7.6 Reference state charts

a) Figure 10: Class attribute (i, j)

b) Figure 11: Class attribute (i, HLAprivilegeToDeleteObject)

5.8 Subscribe Interaction Class

Specifies an interaction class for which the RTI shall notify the joined federate of sent interactions by invoking the *Receive Interaction †* service at the joined federate.

When an interaction is received by a joined federate, the received class of the interaction shall be the interaction's sent class, if subscribed. Otherwise, the received class is the closest superclass of the sent class that is subscribed at the time the interaction is received. Only the parameters from the interaction's received class and its superclasses will be received.

If a joined federate subscribes to multiple locations in an interaction class inheritance tree, each relevant interaction sent shall result in at most one received interaction in the subscribing joined federate.

If the optional passive subscription indicator indicates that this is a passive subscription

— The invocation of this service shall not cause the *Turn Interactions On †* service to be invoked at any other joined federate, and

— If this invocation replaces a previous subscription that was active rather than passive, invocation of this service may cause the *Turn Interactions Off †* service to be invoked at one or more other joined federates.

If the optional passive subscription indicator is not present or indicates that this is an active subscription, the invocation of this service may cause the *Turn Interactions On †* service to be invoked at one or more other joined federates.

5.8.1 Supplied arguments

a) Interaction class designator

b) Optional passive subscription indicator

5.8.2 Returned arguments

a) None

5.8.3 Preconditions

a) The federation execution exists.

b) The federate is joined to that federation execution.

c) The interaction class is defined in the FDD.

d) If the specified interaction class designator is the MOM interaction class HLAmanager. HLAfederate.HLAreport.HLAreportServiceInvocation, the federate's service invocations are not being reported via MOM interactions.

e) Save not in progress.

f) Restore not in progress.

5.8.4 Postconditions

a) The RTI has been informed of the joined federate's requested subscription.

5.8.5 Exceptions

a) The interaction class is not defined in the FDD.

b) Invalid passive subscription designator (if the optional passive subscription indicator argument is supplied).

c) Federate service invocations are being reported via MOM interactions.

d) The federate is not a federation execution member.

e) Save in progress.

f) Restore in progress.

g) RTI internal error.

5.8.6 Reference state charts

a) Figure 12: Interaction class (m)

5.9 Unsubscribe Interaction Class

The *Unsubscribe Interaction Class* service informs the RTI that it shall no longer notify the joined federate of sent interactions of the specified interaction class. See 9.1.5 for an expanded interpretation of this service when DDM is used.

5.9.1 Supplied arguments

a) Interaction class designator

5.9.2 Returned arguments

a) None

5.9.3 Preconditions

a) The federation execution exists.

b) The federate is joined to that federation execution.

c) The interaction class is defined in the FDD.

d) Save not in progress.

e) Restore not in progress.

5.9.4 Postconditions

a) The RTI shall not deliver interactions of the specified interaction class to the joined federate.

5.9.5 Exceptions

a) The interaction class is not defined in the FDD.

b) The federate is not a federation execution member.

c) Save in progress.

d) Restore in progress.

e) RTI internal error.

5.9.6 Reference state charts

a) Figure 12: Interaction class (m)

5.10 Start Registration For Object Class †

The *Start Registration For Object Class †* service shall notify the joined federate that registration of new object instances of the specified object class is advised because at least one of the class attributes that the joined federate is publishing at this object class is actively subscribed to at the specified object class or at a superclass of the specified object class by at least one other joined federate in the federation execution. The joined federate should commence with registration of object instances of the specified class. Generation of the *Start Registration For Object Class †* service advisory shall be controlled using the *Enable/Disable Object Class Relevance Advisory Switch* services. The *Start Registration For Object Class †* service shall be invoked only when the Object Class Relevance Advisory Switch is enabled for the joined federate.

5.10.1 Supplied arguments

a) Object class designator

5.10.2 Returned arguments

a) None

5.10.3 Preconditions

a) The federation execution exists.

b) The federate is joined to that federation execution.

c) At least one of the class attributes that the joined federate is publishing at the specified object class is actively subscribed to at the specified object class or at a superclass of the specified object class by at least one other joined federate in the federation execution.

d) The Object Class Relevance Advisory Switch is enabled for the joined federate.

5.10.4 Postconditions

a) The joined federate has been notified of the requirement to begin registering object instances of the specified object class.

5.10.5 Exceptions

a) The object class is not published.

b) Federate internal error.

5.10.6 Reference state charts

a) None

5.11 Stop Registration For Object Class †

The *Stop Registration For Object Class †* service shall notify the joined federate that registration of new object instances of the specified object class is not advised because none of the class attributes that the joined federate is publishing at this object class is actively subscribed to at the specified object class or at a superclass of the specified object class by any other joined federate in the federation execution. The joined federate should stop registration of new object instances of the specified class. Generation of the *Stop Registration For Object Class †* service advisory shall be controlled using the *Enable/Disable Object Class Relevance Advisory Switch* services. The *Stop Registration For Object Class †* service shall be invoked only when the Object Class Relevance Advisory Switch is enabled for the joined federate.

5.11.1 Supplied arguments

a) Object class designator

5.11.2 Returned arguments

a) None

5.11.3 Preconditions

a) The federation execution exists.

b) The federate is joined to that federation execution.

c) The joined federate is publishing the object class.

d) None of the class attributes that the joined federate is publishing at this object class is actively subscribed to at the specified object class or at a superclass of the specified object class by any other joined federate in the federation execution.

e) The Object Class Relevance Advisory Switch is enabled for the joined federate.

5.11.4 Postconditions

a) The joined federate has been notified of the requirement to stop registration of object instances of the specified object class.

5.11.5 Exceptions

a) The object class is not published.

b) Federate internal error.

5.11.6 Reference state charts

a) None

5.12 Turn Interactions On †

The *Turn Interactions On †* service shall notify the joined federate that the specified class of interactions is relevant because it or a superclass is actively subscribed to by at least one other joined federate in the federation execution. The joined federate should commence with the federation-agreed-upon scheme for sending interactions of the specified class. Generation of the *Turn Interactions On †* service advisory shall be controlled using the *Enable/Disable Interaction Relevance Advisory Switch* services. The *Turn Interactions On †* service shall be invoked only when the Interaction Relevance Advisory Switch is enabled for the joined federate.

5.12.1 Supplied arguments

a) Interaction class designator

5.12.2 Returned arguments

a) None

5.12.3 Preconditions

a) The federation execution exists.

b) The federate is joined to that federation execution.

c) The joined federate is publishing the interaction class.

d) Some other joined federate is actively subscribed to the interaction class or to a superclass of the interaction class.

e) The Interaction Relevance Advisory Switch is enabled for the joined federate.

5.12.4 Postconditions

a) The joined federate has been notified that some other joined federate in the federation execution is subscribed to the interaction class.

5.12.5 Exceptions

a) The interaction class is not published.

b) Federate internal error.

5.12.6 Reference state charts

a) Figure 12: Interaction class (m)

5.13 Turn Interactions Off †

The *Turn Interactions Off †* service shall indicate to the joined federate that the specified class of interactions is not relevant because it or a superclass is not actively subscribed to by any other joined federate in the federation execution. Generation of the *Turn Interactions Off †* service advisory shall be controlled using the *Enable/Disable Interaction Relevance Advisory Switch* services. The *Turn Interactions Off †* service shall be invoked only when the Interaction Relevance Advisory Switch is enabled for the joined federate.

5.13.1 Supplied arguments

a) Interaction class designator

5.13.2 Returned arguments

a) None

5.13.3 Preconditions

a) The federation execution exists.

b) The federate is joined to that federation execution.

c) The joined federate is publishing the interaction class.

d) No other joined federate is actively subscribed to the interaction class or to a superclass of the interaction class.

e) The Interaction Relevance Advisory Switch is enabled for the joined federate.

5.13.4 Postconditions

a) The joined federate has been notified that no other joined federate in the federation execution is subscribed to the interaction class.

5.13.5 Exceptions

a) The interaction class is not published.

b) Federate internal error.

5.13.6 Reference state charts

a) Figure 12: Interaction class (m)

6. Object management

6.1 Overview

This group of HLA services shall deal with the registration, modification, and deletion of object instances and the sending and receipt of interactions.

Object instance discovery is a prime concept in this service group. Object instance *O* shall have a candidate discovery class at joined federate *F* if joined federate *F* is subscribed to either the registered class of *O* or to a superclass of the registered class of *O*. (A joined federate F may be subscribed either by the DM subscription service *Subscribe Object Class Attributes* or by the DDM subscription service *Subscribe Object Class Attributes With Regions*.) If an object instance has a candidate discovery class at a joined federate, the candidate discovery class of the object instance at that joined federate shall be the object instance's registered class, if subscribed to by the joined federate. Otherwise, the candidate discovery class of the object instance shall be the closest superclass of the object instances's registered class to which the joined federate is subscribed.

A joined federate discovers an object instance via the *Discover Object Instance †* service. This service shall be invoked at a joined federate *F* for object instance *O* when

a) *O* is not known at *F,* and

b) There is an instance attribute *i* of *O* that has a corresponding class attribute *i',* and

   — Another joined federate (not *F*) owns *i,* and

   — Either

      1) *i'* is a subscribed attribute of *O*'s candidate discovery class, or

      2) *i'* is a subscribed attribute of *O*'s candidate discovery class with regions and the update region set of *i* by the owning joined federate overlaps the subscription region set of *i'* at *O*'s candidate discovery class at the subscribing joined federate

When the *Discover Object Instance †* service is invoked, the class that is an argument to this service invocation shall be called the discovered class of the object instance. At the moment of discovery, the discovered class shall be the same as the candidate discovery class. Subsequent to discovery, the discovered class cannot change. The candidate discovery class may change. As long as an object instance remains known, however, its candidate discovery class is not of interest.

When a joined federate either uses the *Register Object Instance* service to register an object instance or receives an invocation of the *Discover Object Instance †* to discover an object instance, that object instance becomes known to the joined federate and the object instance has a known class at that joined federate. If a joined federate knows about an object instance as a result of having registered it, that object instance's known class is its registered class. If the joined federate knows about the object instance as a result of having discovered it, the object instance's known class is its discovered class.

When the *Discover Object Instance †* service is invoked, there shall be an instance attribute that is part of the newly discovered object instance that immediately comes into scope at the discovering joined federate, both when DDM is used and when it isn't used. An instance attribute of an object instance shall be in scope for joined federate *F* if

a) The object instance is known to the joined federate

b) The instance attribute is owned by another joined federate

c) Either the instance attribute's corresponding class attribute is a subscribed attribute

   1) Of the known class of the object instance, or

   2) Of the instance attribute overlaps the subscription region set of the instance attribute's corresponding class attribute at the known class of the instance attribute at the subscribing joined federate

If an instance attribute is in scope for a given joined federate ***F***, exactly one other joined federate exists (***U***) in the federation execution that is capable of invoking the *Update Attribute Values* service for that instance attribute such that if ***U*** were to invoke the *Update Attribute Values* service for that instance attribute, all of the preconditions for the invocation of the corresponding *Reflect Attribute Values †* service at ***F*** are satisfied.

The *Reflect Attribute Values †* service may be invoked at a joined federate for a given instance attribute when

a) The instance attribute is in-scope.

b) A corresponding *Update Attribute Values* service for the instance attribute was invoked by another federate.

The *Reflect Attribute Values †* service may also be invoked at a joined federate for a given instance attribute when

a) The instance attribute is out-of-scope.

b) A corresponding *Update Attribute Values* service for the instance attribute was invoked by another federate when the instance attribute was in-scope.

A joined federate may also direct the RTI, via the *Enable/Disable Attribute Relevance Advisory Switch* services, to indicate that the joined federate does or does not want the RTI to use the *Turn Updates On For Object Instance †* and *Turn Updates Off For Object Instance †* services to inform the joined federate when updates to particular instance attributes are relevant to the other joined federates in the federation execution.

Note that there are obscure cases under which an object instance shall become orphaned within a federation execution. Orphaned object instances are unknown by all joined federates and cannot be discovered by any means. An object instance shall become orphaned only if no joined federate knows the object instance (i.e., because the last joined federate to know the object instance has resigned or invoked *Local Delete Object Instance*). A consequence of no joined federate knowing the object instance is that no instance attributes of that object instance shall be owned. Because the definition of discovery requires that at least one instance attribute be owned, no joined federates can discover the object instance. Without a joined federate being able to discover the object instance, no joined federate can acquire ownership of an instance attribute via the ownership management services. Orphaned object instances shall be unreachable by joined federates and should be dealt with appropriately by the RTI.

Interaction receipt is also an important concept in the object management service group. Interaction *I* shall have a candidate received class at joined federate ***F*** if joined federate ***F*** is subscribed to either the sent class of *I* or to a superclass of the sent class of *I*. (A joined federate ***F*** may be subscribed to an interaction class either by the DM subscription service *Subscribe Interaction Class* or by the DDM subscription service *Subscribe Interaction Class With Regions*.) If an interaction has a candidate received class at a joined federate, the candidate received class of the interaction at that joined federate shall be the interaction's sent class, if subscribed to by the joined federate. Otherwise, the candidate received class of the interaction shall be the closest superclass of the interaction's sent class to which the joined federate is subscribed.

A joined federate receives an interaction via the *Receive Interaction †* service. This service may be invoked at a joined federate ***F*** when

a) Another joined federate (not *F*) has invoked the *Send Interaction* service to send interaction *I*, and

b) Either, *I* has a candidate received class at *F* and

   1) This candidate received class is a subscribed interaction class, or

   2) This candidate received class is a subscribed interaction class with regions, and the update region set of *I* overlaps the subscription region set for *I*'s candidate received class at the subscribing joined federate.

When the *Receive Interaction †* service is invoked, the class that is an argument to this service invocation shall be called the received class of the interaction that is received as a result of this service invocation. At the moment of receipt, the received class shall be the same as the candidate received class.

6.1.1 Statecharts

Figure 13 depicts formal representations of the state of an arbitrary object instance, and Figure 14 depicts the implications of ownership of an arbitrary instance attribute.

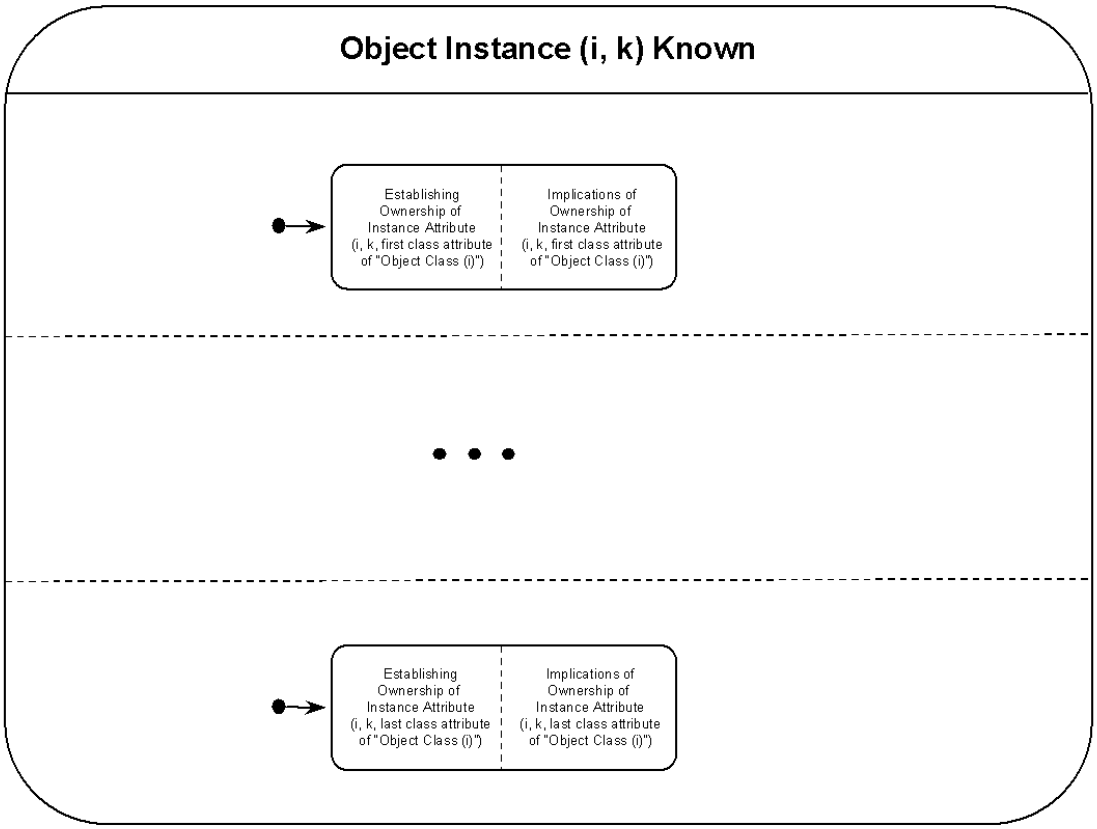

Figure 13—Object instance (i, k) known

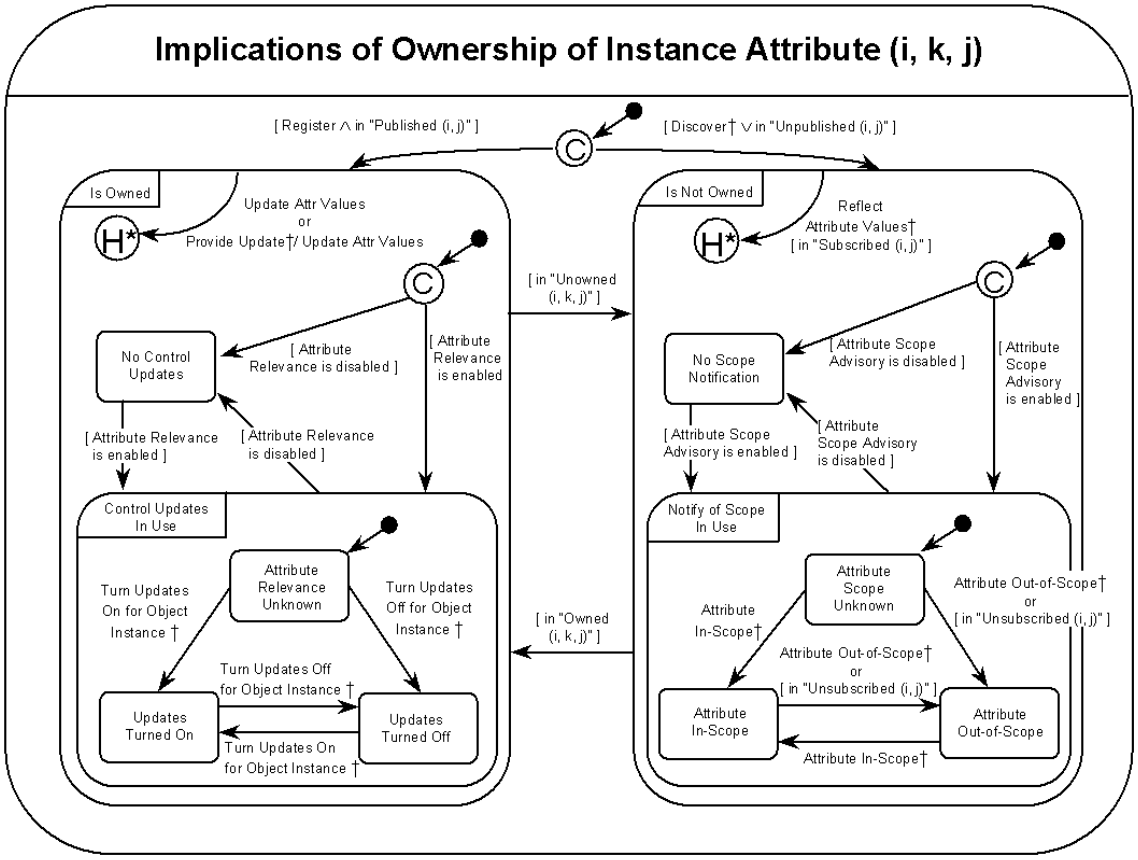

Figure 14—Implications of ownership of instance attribute (i, k, j)

6.1.2 Auto-provide switch

The federation execution-wide auto-provide switch indicates whether the RTI should automatically solicit attribute value updates from attribute owning federates. The initial setting for this switch shall come from the FDD at federation execution creation time. This switch setting shall be modifiable during a federation execution via the HLAmanager.HLAfederation.HLAadjust.HLAsetSwitches MOM interaction.

Whenever this switch is Enabled, the *Provide Attribute Value Update †* service shall be invoked, at an instance attribute owning federate, for every instance attribute in an execution, whenever it becomes inscope at another nonowning federate. If an instance attribute becomes in-scope at multiple federates at the same time, the *Provide Attribute Value Update †* service may be invoked at the owning federate just once, rather than once for each federate at which the instance attribute became in-scope.

When this switch changes to Enabled during a federation execution, the *Provide Attribute Value Update †* service shall be invoked at the appropriate owning federate for every in-scope instance attribute.

This switch relies on the definition of in-scope attributes, not on the setting of the Attribute Scope Advisory Switch. Auto-Provide shall function the same whether the Attribute Scope Advisory Switch is Enabled or Disabled.

6.1.3 Transportation types

The type of transportation that carries instance attribute values and interactions can be controlled using various HLA facilities. The following transportation types shall be supported by the RTI:

— HLAreliable: provides reliable delivery of data in the sense that TCP/IP delivers its data reliably
— HLAbestEffort: makes an effort to deliver data in the sense that UDP provides best-effort delivery.

Other transportation types may be supported by the RTI.

6.1.4 Passelization

All of the instance attribute/value pairs in an *Update Attribute Values* service invocation for instance attributes that have identical transportation type, sent message order type, and set of sent region designators (if present) shall be in one passel (e.g., shall be delivered in the same *Reflect Attribute Values †* service invocation). The availability of the passel at reflect time shall depend on the subscriptions and subscription region specifications at the time of the reflect, not at the time of the update. Elements of a passel shall be discarded (due to changes in subscriptions and subscription region specifications during the time between an update and a corresponding reflect), but elements shall not change passels, passels shall not be divided, and passels shall not be combined with other passels. Since one *Update Attribute Values* invocation can result in the creation of multiple passels, an *Update Attribute Values* invocation can result in multiple *Reflect Attribute Values †* invocations in a reflecting joined federate. A passel delivered via a *Reflect Attribute Values †* service invocation shall not contain elements from more than one *Update Attribute Values* service invocation. Invocation of the *Update Attribute Values* service may actually result in the sending of multiple messages as each passel resulting from the invocation shall be considered a separate message. Duplicate passels (messages) shall not be delivered.

6.2 Reserve Object Instance Name

This service requests that the RTI attempt to reserve the supplied name as a federation execution-wide unique name. The value of the supplied name argument shall not begin with “HLA.” The RTI shall provide the result of the reservation attempt via the *Object Instance Name Reserved †* service.

6.2.1 Supplied arguments

a) Name

6.2.2 Returned arguments

a) None

6.2.3 Preconditions

a) The federation execution exists.
b) The federate is joined to that federation execution.
c) The value of the name argument does not begin with “HLA.”
d) Save not in progress.
e) Restore not in progress.

6.2.4 Postconditions

a) The RTI has the candidate name.

6.2.5 Exceptions

a) Illegal name.

b) The federate is not a federation execution member.

c) Save in progress.

d) Restore in progress.

e) RTI internal error.

6.2.6 Reference state charts

a) None

6.3 Object Instance Name Reserved †

Notifies the joined federate whether the name provided in a previous invocation of *Register Object Instance Name* service has been reserved, which shall mean that the name is federation execution-wide unique. If the name is unique, no joined federate joined to the current federation execution shall subsequently receive a successful reservation for that name.

6.3.1 Supplied arguments

a) Name

b) Reservation success indicator

6.3.2 Returned arguments

a) None

6.3.3 Preconditions

a) The federation execution exists.

b) The federate is joined to that federation execution.

c) The joined federate had previously invoked the *Reserve Object Instance Name* with the name.

6.3.4 Postconditions

a) If the success indicator is true, the name shall be federation execution-wide unique.

6.3.5 Exceptions

a) Unknown name

b) Federate internal error

6.3.6 Reference state charts

a) None

6.4 Register Object Instance

The RTI shall create a federation execution-wide unique object instance handle and shall coadunate that handle with an instance of the supplied object class. All instance attributes of the object instance for which the corresponding class attributes are currently published by the registering joined federate shall be set as owned by the registering joined federate.

An object instance handle shall be uniform throughout the federation execution; i.e., with respect to a particular object instance, the handle provided to the joined federate which registers the object instance shall be the same handle provided to all joined federates which discover the object instance.

If the optional object instance name argument is supplied, that name shall have been successfully reserved as indicated by the *Object Instance Name Reserved †* service and shall be coadunated with the object instance. If the optional object instance name argument is not supplied, the RTI shall create a federation execution-wide unique name and that name shall be coadunated with the object instance.

6.4.1 Supplied arguments

a) Object class designator

b) Optional object instance name

6.4.2 Returned arguments

a) Object instance handle

6.4.3 Preconditions

a) The federation execution exists.

b) The federate is joined to that federation execution.

c) The object class is defined in the FDD.

d) The joined federate is publishing the object class.

e) If the optional object instance name argument is supplied, that name is reserved and not already coadunated with another object instance.

f) Save not in progress.

g) Restore not in progress.

6.4.4 Postconditions

a) The returned object instance handle is coadunated with the object instance.

b) The joined federate acquires ownership of the instance attributes that correspond to the currently published class attributes for the specified object class.

c) If the optional object instance name argument is supplied, that name is coadunated with the object instance.

6.4.5 Exceptions

a) The object class is not defined in the FDD.

b) The joined federate is not publishing the specified object class.

c) The object instance name was not reserved (if optional object instance name argument is supplied).

d) The object instance name was already coadunated with another object instance (if optional object instance name argument is supplied).

e) The federate is not a federation execution member.

f) Save in progress.

g) Restore in progress.

h) RTI internal error.

6.4.6 Reference state charts

a) Figure 9: Object class (i)

b) Figure 14: Implications of ownership of instance attribute (i, k, j)

c) Figure 15: Establishing ownership of instance attribute (i, k, j)

6.5 Discover Object Instance †

The *Discover Object Instance †* service shall inform the joined federate to discover an object instance. Object instance discovery is described in 6.1. The object instance handle shall be unique to the federation execution and shall be uniform (see 6.4) throughout the federation execution.

6.5.1 Supplied arguments

a) Object instance handle

b) Object class designator

c) Object instance name

6.5.2 Returned arguments

a) None

6.5.3 Preconditions

a) The federation execution exists.

b) The federate is joined to that federation execution.

c) An object instance with the specified designator exists.

d) The joined federate is subscribed to some class attribute ***att*** at the specified object class, and ***att***'s corresponding instance attribute that is part of the specified object instance is owned by another joined federate.

e) The joined federate is subscribed with regions to some class attribute ***att*** at the specified object class, and the subscription region set overlaps the update region set of ***att***'s corresponding instance attribute that is part of the specified object instance by the joined federate that owns that instance attribute.

f) The joined federate does not know about the object instance with the specified designator.

6.5.4 Postconditions

a) The object instance is known to the joined federate.

6.5.5 Exceptions

a) The joined federate could not discover the object instance.

b) The object class is not recognized.

c) Federate internal error.

6.5.6 Reference state charts

a) Figure 9: Object class (i)

b) Figure 14: Implications of ownership of instance attribute (i, k, j)

c) Figure 15: Establishing ownership of instance attribute (i, k, j)

6.6 Update Attribute Values

The *Update Attribute Values* service shall provide instance attribute values to the federation. The joined federate shall be able to update only owned instance attributes. The joined federate should supply instance attribute values using the federation-agreed-upon update policies. This service, coupled with the *Reflect Attribute Values †* service, shall form the primary data exchange mechanism supported by the RTI.

If preferred order type (see 8.1.1) of at least one instance attribute is TSO, the joined federate invoking this service is time-regulating, and the time stamp argument is present, this service shall return a federation-unique message retraction designator. Otherwise, no message retraction designator shall be returned. A time stamp value should be provided if the preferred order type of at least one instance attribute is TSO and the joined federate invoking this service is time-regulating.

The user-supplied tag argument shall be provided with all corresponding *Reflect Attribute Value †* service invocations, even multiple ones at the same reflecting joined federate.

If at any time between the invocation of this service and what would be the invocation of the corresponding *Reflect Attribute Values †* service for a given instance attribute that is supplied as an argument to this service a potential joined federate either

— Is unsubscribed to that instance attribute's corresponding class attribute at the known class of the object instance at the potential reflecting joined federate, or

— Is using a subscription region set for the instance attribute's corresponding class attribute at the known class of the object instance at the potential reflecting joined federate and this region set does not overlap the update region set of the instance attribute at the time of update

the RTI may not invoke the corresponding *Reflect Attribute Values †* service for that instance attribute at that joined federate.

If at any time between the invocation of this service and what would be the invocation of the corresponding *Reflect Attribute Values †* service for a given instance attribute that is supplied as an argument to this service a potential reflecting joined federate either

— Is subscribed to that instance attribute's corresponding class attribute at the known class of the object instance at the potential reflecting joined federate, or

— Is using a subscription region set for the instance attribute's corresponding class attribute at the known class of the object instance at the potential reflecting joined federate and this region set overlaps the update region set of the instance attribute at the time of update

the RTI may invoke the corresponding *Reflect Attribute Values †* service for the instance attribute at that joined federate, subject to the preconditions of the *Reflect Attribute Values †* service and the rules of time management.

6.6.1 Supplied arguments

a) Object instance designator

b) Constrained set of attribute designator and value pairs

c) User-supplied tag

d) Optional time stamp

6.6.2 Returned arguments

a) Optional message retraction designator

6.6.3 Preconditions

a) The federation execution exists.

b) The federate is joined to that federation execution.

c) The joined federate owns the instance attributes for which values are provided.

d) The attributes are defined in the FDD.

e) An object instance with the specified designator exists.

f) The joined federate knows about the object instance with the specified designator.

g) The time stamp value shall be in accordance with the constraints stated in Clause 8 (if optional time stamp argument is supplied).

h) If the optional time stamp argument is supplied to this service invocation, the invoking joined federate is time-regulating, and it has invoked the *Delete Object Instance* service for the specified object instance with a time stamp argument (while time-regulating), then the update's time stamp shall be less than or equal to the delete's time stamp.

i) Save not in progress.

j) Restore not in progress.

6.6.4 Postconditions

a) The new instance attribute values have been supplied to the RTI.

6.6.5 Exception

a) The object instance is not known.

b) The specified class attributes are not available attributes of the instance object class.

c) The joined federate does not own the specified instance attributes.

d) The time stamp is invalid (if optional time stamp argument is supplied).

e) The federate is not a federation execution member.

f) Save in progress.

g) Restore in progress.

h) RTI internal error.

6.6.6 Reference state charts

a) Figure 14: Implications of ownership of instance attribute (i, k, j)

6.7 Reflect Attribute Values †

The *Reflect Attribute Values †* service shall provide the joined federate with new values for the specified instance attributes. This service, coupled with the *Update Attribute Values* service, shall form the primary data exchange mechanism supported by the RTI.

The user-supplied tag argument supplied to the corresponding *Update Attribute Values* service invocation shall be provided with all corresponding *Reflect Attribute Value †* service invocations, even multiple ones in the same reflecting joined federate.

The transportation type shall indicate the transportation type of the passel associated with this service invocation.

A time stamp shall be provided if one was specified in the corresponding *Update Attribute Values* service invocation. The receive message order type shall indicate the type (either RO or TSO) of the received message. The time stamp and receive message order type arguments shall be supplied together or not at all. The sent message order type shall indicate the type (either RO or TSO) of the sent message.

A retraction handle shall be provided if and only if the sent message order type is TSO.

If specified instance attributes have available dimensions and the Convey Region Designator Sets Switch is enabled, the set of sent region designators argument shall contain the update region set, if any, that was used for update of the instance attributes at the joined federate, which invoked the corresponding *Update Attribute Values* service.

A joined federate invoking the *Update Attribute Values* service shall not receive the induced *Reflect Attribute Values †* service invocation, regardless of the subscription situation.

6.7.1 Supplied arguments

a) Object instance designator

b) Constrained set of attribute designator and value pairs

c) User-supplied tag

d) Sent message order type

e) Transportation type

f) Optional time stamp

g) Optional receive message order type

h) Optional message retraction designator

i) Optional set of sent region designators

6.7.2 Returned arguments

a) None

6.7.3 Preconditions

a) The federation execution exists.

b) The federate is joined to that federation execution.

c) The joined federate knows about the object instance with the specified designator.

d) For each attribute designator specified.

   1) This joined federate is subscribed to the class attribute at the known class of the object instance at this joined federate.

   2) The joined federate has a subscription region set for the class attribute at the known class of the object instance at the subscribing joined federate and this region set overlaps the update region set of the instance attribute at the time of update.

e) The joined federate did not send the corresponding update.

6.7.4 Postconditions

a) The new instance attribute values have been supplied to the joined federate.

6.7.5 Exceptions

a) The object instance is not known.

b) The attribute designator is not recognized.

c) The joined federate is not subscribed to the class attribute of the known class of the object instance.

d) The time stamp is invalid (if receive message order type is TSO).

e) Federate internal error.

6.7.6 Reference state charts

a) Figure 14: Implications of ownership of instance attribute (i, k, j)

6.8 Send Interaction

The *Send Interaction* service shall send an interaction into the federation. The interaction parameters shall only be those in the specified class and all superclasses, as defined in the FDD.

If preferred order type (see 8.1.1) of the interaction is TSO, the joined federate invoking this service is time-regulating, and the time stamp argument is present, this service shall return a federation-unique message retraction designator. Otherwise, no message retraction designator shall be returned. A time stamp value should be provided if the preferred order type of the interaction is TSO and the joined federate invoking this service is time-regulating.

If at any time between the invocation of this service for a particular interaction class argument and what would be the invocation of the corresponding *Receive Interaction †* service at a potential receiving joined federate, the potential receiving joined federate is unsubscribed to the specified interaction class, the RTI may not invoke the corresponding *Receive Interaction †* service at that joined federate.

If at any time between the invocation of this service for a particular interaction class and what would be the invocation of the corresponding *Receive Interaction †* service at a potential receiving joined federate, the potential receiving joined federate either

— Is subscribed to the specified interaction class, or

— Is subscribed to the specified interaction class with regions

the RTI may invoke the corresponding *Receive Interaction †* service at that joined federate, subject to the preconditions of the *Receive Interaction †* service and the rules of time management.

A joined federate invoking the *Send Interaction* service shall not receive the induced *Receive Interaction †* service invocation, regardless of the subscription/region situation.

The user-supplied tag argument shall be provided with all corresponding *Receive Interaction †* service invocations.

6.8.1 Supplied arguments

a) Interaction class designator
b) Constrained set of interaction parameter designator and value pairs
c) User-supplied tag
d) Optional time stamp

6.8.2 Returned arguments

a) Optional message retraction designator

6.8.3 Preconditions

a) The federation execution exists.

b) The federate is joined to that federation execution.

c) The joined federate is publishing the interaction class.

d) The interaction class is defined in the FDD.

e) The parameters are defined in the FDD.

f) The time stamp value shall be in accordance with the constraints stated in clause 8 (if optional time stamp argument is supplied).

g) Save not in progress (There are certain MOM interactions that can be sent during a save. See 11.4.2).

h) Restore not in progress (There are certain MOM interactions that can be sent during a restore. See 11.4.2).

6.8.4 Postconditions

a) The RTI has received the interaction.

6.8.5 Exceptions

a) The joined federate is not publishing the specified interaction class.

b) The interaction class is not defined in the FDD.

c) The interaction parameter is not defined in FDD.

d) The time stamp is invalid (if optional time stamp argument is supplied).

e) The federate is not a federation execution member.

f) Save in progress.

g) Restore in progress.

h) RTI internal error.

6.8.6 Reference state charts

a) Figure 12: Interaction class (m)

6.9 Receive Interaction †

The *Receive Interaction †* service shall provide the joined federate with a sent interaction.

The user-supplied tag argument supplied to the corresponding *Send Interaction* or *Send Interaction With Regions* service invocation shall be provided with all corresponding *Receive Interaction †* service invocations.

The transportation type argument shall indicate the transportation type of the specified interaction. This shall be the transportation type of the published interaction at the joined federate, which invoked the corresponding *Send Interaction* or *Send Interaction With Regions* service.

A time stamp shall be provided if one was specified in the corresponding *Send Interaction* or *Send Interaction With Regions* service invocation. The receive message order type shall indicate the type (either RO or TSO) of the received message. The time stamp and receive message order type arguments shall be supplied together or not at all. The sent message order type shall indicate the type (either RO or TSO) of the sent message.

A retraction handle shall be provided if and only if the sent message order type is TSO.

If the specified interaction has available dimensions and the Convey Region Designator Sets Switch is enabled, the set of sent region designators argument shall contain the update region set, if any, that was supplied to the corresponding *Send Interaction With Regions* service invocation by the sending joined federate.

A joined federate invoking the *Send Interaction* or *Send Interaction With Regions* service shall not receive the induced *Receive Interaction †* service invocation, regardless of the subscription situation.

At most, one induced *Receive Interaction †* service invocation shall happen at a joined federate as a result of the invocation of the *Send Interaction* or *Send Interaction With Regions* service at another joined federate.

6.9.1 Supplied arguments

a) Interaction class designator

b) Constrained set of interaction parameter designator and value pairs

c) User-supplied tag

d) Sent message order type

e) Transportation type

f) Optional time stamp

g) Optional receive message order type

h) Optional message retraction designator

i) Optional set of sent region designators

6.9.2 Returned arguments

a) None

6.9.3 Preconditions

a) The federation execution exists

b) The federate is joined to that federation execution

c) The joined federate is subscribed to the interaction class

d) The joined federate has a subscription region set for the interaction class and this region set overlaps the update region set that was used for sending the interaction

e) The receiving joined federate did not send the corresponding interaction

6.9.4 Postconditions

a) The joined federate has received the interaction.

6.9.5 Exceptions

a) The interaction class is not recognized.

b) The interaction parameter is not recognized.

c) The joined federate is not subscribed to the interaction class.

d) The time stamp is invalid (if receive message order type is TSO).

e) Federate internal error.

6.9.6 Reference state charts

a) Figure 12: Interaction class (m)

6.10 Delete Object Instance

The *Delete Object Instance* service shall inform the federation that an object instance with the specified designator, which has the **HLAprivilegeToDeleteObject** instance attribute that is owned by the joined federate, is to be removed from the federation execution. Once the object instance is removed from the federation execution, the designator and name shall not be reused and all joined federates that owned attributes of the object instance shall no longer own those attributes. The RTI shall use the *Remove Object Instance †* service to inform all other joined federates which know the object instance that the object instance has been deleted. The invoking joined federate shall own the **HLAprivilegeToDeleteObject** attribute of the specified object instance.

The preferred order type (see 8.1.1) of the sent message representing a *Delete Object Instance* service invocation shall be based on the preferred order type of the **HLAprivilegeToDeleteObject** attribute of the specified object instance. If preferred order type of the **HLAprivilegeToDeleteObject** attribute of the

specified object instance is TSO, the joined federate invoking this service is time-regulating, and the time stamp argument is present, this service shall return a federation-unique message retraction designator. Otherwise, no message retraction designator shall be returned. A time stamp value should be provided if the preferred order type of the **HLAprivilegeToDeleteObject** attribute of the specified object instance is TSO and the joined federate invoking this service is time-regulating.

The user-supplied tag argument shall be provided with all corresponding *Remove Object Instance †* service invocations.

6.10.1 Supplied arguments

a) Object instance designator

b) User-supplied tag

c) Optional time stamp

6.10.2 Returned arguments

a) Optional message retraction designator

6.10.3 Preconditions

a) The federation execution exists.

b) The federate is joined to that federation execution.

c) An object instance with the specified designator exists.

d) The joined federate knows about the object instance with the specified designator.

e) The joined federate has the privilege to delete the object instance (it owns the **HLAprivilegeToDeleteObject** instance attribute).

f) The time stamp value shall be in accordance with the constraints stated in Clause 8 (if optional time stamp argument is supplied).

g) Save not in progress.

h) Restore not in progress.

6.10.4 Postconditions

a) The RTI has received the *Delete Object Instance* request.

b) The invoking joined federate may no longer update any previously owned attributes of the specified object instance if the sent message order type of the *Delete Object Instance* request is receive ordered.

c) The invoking joined federate may no longer update any previously owned attributes of the specified object instance with sent message order type of TSO and a time stamp value greater than that specified in the *Delete Object Instance* service invocation if the sent message order type of the *Delete Object Instance* request is TSO.

d) If the sent message order type is RO, the object instance is no longer known at the invoking federate, no longer exists as far as the federate is concerned, and can no longer be discovered at any other federate.

e) If the sent message order type is TSO, the object instance will no longer be known at the invoking federate (and will no longer exist as far as the federate is concerned) once the federate advances its

logical time to or past the specified time. If the federate disables time-regulation, the object instance immediately becomes unknown to the federate and ceases to exist as far as the federate is concerned. The object instance may still be discovered by other federates until their logical times are greater than or equal to the specified time of the deletion.

6.10.5 Exceptions

a) The joined federate does not own the delete privilege.
b) The object instance is not known.
c) The time stamp is invalid (if optional time stamp argument is supplied).
d) The federate is not a federation execution member.
e) Save in progress.
f) Restore in progress.
g) RTI internal error.

6.10.6 Reference state charts

a) Figure 9: Object class (i)

6.11 Remove object instance †

The *Remove Object Instance †* service shall inform the joined federate that an object instance has been deleted from the federation execution.

The user-supplied tag argument supplied to the corresponding *Delete Object Instance* service invocation shall be provided with all corresponding *Remove Object Instance †* service invocations.

A time stamp shall be provided if one was specified in the corresponding *Delete Object Instance* service invocation. The receive message order type shall indicate the type (either RO or TSO) of the received message. The time stamp and receive message order type arguments shall be supplied together or not at all. The sent message order type shall indicate the type (either RO or TSO) of the sent message.

A retraction handle shall be provided if the delete message had a sent order type of TSO. Otherwise, no retraction handle shall be provided.

A joined federate invoking the *Delete Object Instance* service shall not receive the induced *Remove Object Instance †* service invocation.

6.11.1 Supplied arguments

a) Object instance designator
b) User-supplied tag
c) Sent message order type
d) Optional time stamp
e) Optional receive message order type
f) Optional message retraction designator

6.11.2 Returned arguments

a) None

6.11.3 Preconditions

a) The federation execution exists.

b) The federate is joined to that federation execution.

c) The joined federate knows about the object instance with the specified designator.

d) The receiving joined federate did not send the corresponding *Delete Object Instance* service.

6.11.4 Postconditions

a) The joined federate has been notified to remove the object instance.

b) The object instance is no longer known by the federate and no longer exists as far as the federate is concerned.

6.11.5 Exceptions

a) The object instance is not known.

b) The time stamp is invalid (if receive message order type is TSO).

c) Federate internal error.

6.11.6 Reference state charts

a) Figure 9: Object class (i)

6.12 Local Delete Object Instance

The *Local Delete Object Instance* service shall inform the RTI that it shall treat the specified object instance as if the RTI had never notified the invoking joined federate to discover the object instance. The object instance shall not be removed from the federation execution. The joined federate shall not own any attributes of the specified object instance when invoking this service.

After the termination of invocation of this service, the specified object instance may subsequently be discovered by the invoking joined federate, at a possibly different class than previously known.

6.12.1 Supplied arguments

a) Object instance designator

6.12.2 Returned arguments

a) None

6.12.3 Preconditions

a) The federation execution exists.

b) The federate is joined to that federation execution.

c) An object instance with the specified designator exists.

d) The joined federate knows about the object instance with the specified designator.

e) The joined federate owns no attributes of the specified object instance.

f) Instance attributes of the specified object instance do not exist for which the joined federate has either

   1) Invoked the *Attribute Ownership Acquisition* service, but has not yet received an invocation of either the *Confirm Attribute Ownership Acquisition Cancellation †* service or the *Attribute Ownership Acquisition Notification †* service, or

   2) Invoked the *Attribute Ownership Acquisition If Available* service, but has not yet received an invocation of the *Attribute Ownership Unavailable †* service, received an invocation of the *Attribute Ownership Acquisition Notification †* service, or invoked the *Attribute Ownership Acquisition* service [after which condition 1) applies].

g) Save not in progress.

h) Restore not in progress.

6.12.4 Postconditions

a) The object instance does not exist with respect to the invoking joined federate.

6.12.5 Exceptions

a) The object instance is not known.

b) The joined federate owns instance attributes.

c) Cannot local delete due to pending attempt to acquire instance attribute ownership.

d) The federate is not a federation execution member.

e) Save in progress.

f) Restore in progress.

g) RTI internal error.

6.12.6 Reference state charts

a) Figure 9: Object class (i)

6.13 Change Attribute Transportation Type

The transportation type for each attribute of an object instance shall be initialized from the object class description in the FDD. A joined federate may choose to change the transportation type during execution. Invoking the *Change Attribute Transportation Type* service shall change the transportation type for all future *Update Attribute Values* service invocations for the specified attributes of the specified object instance only for the invoking joined federate. If the invoking joined federate loses ownership of an instance attribute after changing its transportation type and later acquires ownership of that instance attribute again, the transportation type will be as defined in the FDD.

6.13.1 Supplied arguments

a) Object instance designator

b) Set of attribute designators

c) Transportation type

6.13.2 Returned arguments

a) None

6.13.3 Preconditions

a) The federation execution exists.

b) The federate is joined to that federation execution.

c) An object instance with the specified designator exists.

d) The joined federate knows about the object instance with the specified designator.

e) The specified class attributes are available attributes of the known class of the specified object instance designator.

f) The joined federate owns the instance attributes.

g) The transportation type is valid (see 6.1.3).

h) Save not in progress.

i) Restore not in progress.

6.13.4 Postconditions

a) The transportation type is changed for the specified instance attributes.

6.13.5 Exceptions

a) The object instance is not known.

b) The class attribute is not available at the known class of the object instance.

c) The joined federate does not own the specified instance attributes.

d) The transportation type is invalid.

e) The federate is not a federation execution member.

f) Save in progress.

g) Restore in progress.

h) RTI internal error.

6.13.6 Reference state charts

a) None

6.14 Change Interaction Transportation Type

The transportation type for each interaction shall be initialized from the interaction class description in the FDD. A joined federate may choose to change the transportation type during execution. Invoking the *Change Interaction Transportation Type* service shall change the transportation type for all future *Send Interaction* and *Send Interaction with Regions* service invocations for the specified interaction class for the invoking joined federate only.

6.14.1 Supplied arguments

a) Interaction class designator

b) Transportation type

6.14.2 Returned arguments

a) None

6.14.3 Preconditions

a) The federation execution exists.

b) The federate is joined to that federation execution.

c) The interaction class is defined in the FDD.

d) The joined federate is publishing the interaction class.

e) The transportation type is valid (see 6.1.3).

f) Save not in progress.

g) Restore not in progress.

6.14.4 Postconditions

a) The transportation type is changed for the specified interaction class.

6.14.5 Exceptions

a) The interaction class is not defined in the FDD.

b) The joined federate is not publishing the interaction class.

c) The transportation type is invalid.

d) The federate is not a federation execution member.

e) Save in progress.

f) Restore in progress.

g) RTI internal error.

6.14.6 Reference state charts

a) None

6.15 Attributes In Scope †

The *Attributes In Scope †* service shall notify the joined federate that the specified attributes for the object instance are in scope for the joined federate. Generation of the *Attributes In Scope †* service advisory can be controlled using the *Enable/Disable Attribute Scope Advisory Switch* services. The *Attributes In Scope †* service shall be invoked only when the Attribute Scope Advisory Switch is enabled.

6.15.1 Supplied arguments

a) Object instance designator

b) Set of attribute designators

6.15.2 Returned arguments

a) None

6.15.3 Preconditions

a) The federation execution exists.

b) The federate is joined to that federation execution.

c) The joined federate knows about the object instance with the specified designator.

d) For each attribute designator specified, the joined federate

   1) Is subscribed to the class attribute at the known class of the object instance at this joined federate, or

   2) Has a subscription region set for the class attribute at the known class of the object instance at the subscribing joined federate and this region set overlaps the update region set of the instance attribute at the time of update.

e) The instance attributes are now in scope.

f) The attribute scope advisory switch is enabled.

6.15.4 Postconditions

a) The joined federate has been advised that the instance attributes are in scope.

6.15.5 Exceptions

a) The object instance is not known.

b) The attribute designator is not recognized.

c) The joined federate is not subscribed to the class attribute of the known class of the object instance.

d) Federate internal error.

6.15.6 Reference state charts

a) None

6.16 Attributes Out Of Scope †

The *Attributes Out Of Scope †* service shall notify the joined federate that the specified attributes of the object instance are out of scope for the joined federate. Generation of the *Attributes Out Of Scope †* service advisory can be controlled using the *Enable/Disable Attribute Scope Advisory Switch* services. The *Attributes Out of Scope †* service shall be invoked only when the Attribute Scope Advisory Switch is enabled.

6.16.1 Supplied arguments

a) Object instance designator

b) Set of attribute designators

6.16.2 Returned arguments

a) None

6.16.3 Preconditions

a) The federation execution exists.

b) The federate is joined to that federation execution.

c) The joined federate knows about the object instance with the specified designator.

d) For each attribute designator specified, the joined federate

   1) Is subscribed to the class attribute at the known class of the object instance at this joined federate, or

   2) Has a subscription region set for the class attribute at the known class of the object instance at the subscribing joined federate.

e) The instance attributes are no longer in scope.

f) The attribute scope advisory switch is enabled.

6.16.4 Postconditions

a) The joined federate has been advised that the instance attributes are out of scope.

6.16.5 Exceptions

a) The object instance is not known.

b) The attribute designator is not recognized.

c) The joined federate is not subscribed to the class attribute of the known class of the object instance.

d) Federate internal error.

6.16.6 Reference state charts

a) None

6.17 Request Attribute Value Update

The *Request Attribute Value Update* service shall be used to stimulate the update of values of specified attributes. When this service is used, the RTI shall solicit the current values of the specified attributes from their owners using the *Provide Attribute Value Update †* service. The RTI shall not invoke the *Provide Attribute Value Update †* service for unowned instance attributes. When an object class is specified, the RTI shall solicit the values of the specified instance attributes for all object instances registered at that class and all subclasses of that class. When an object instance designator is specified, the RTI shall solicit the values of the specified instance attributes for the particular object instance. The time stamp of any resulting *Reflect Attribute Values †* service invocations is determined by the updating joined federate.

If the federate invoking this service also owns some of the instance attributes whose values are being requested, it will not have the *Provide Attribute Value Update †* service invoked on it by the RTI. The request is taken as an implicit invocation of the provide service at the requesting federate for all qualified instance attributes that it owns.

The user-supplied tag argument shall be provided with all corresponding *Provide Attribute Value Update †* service invocations.

6.17.1 Supplied arguments

a) Object instance designator or object class designator

b) Set of attribute designators

c) User-supplied tag

6.17.2 Returned arguments

a) None

6.17.3 Preconditions

a) The federation execution exists.

b) The federate is joined to that federation execution.

c) The specified class attributes are available attributes of the known class of the specified object instance (when first argument is an object instance designator).

d) An object instance with the specified designator exists (when first argument is an object instance designator).

e) The joined federate knows about the object instance with the specified designator (when first argument is an object instance designator).

f) The specified object class is defined in the FDD (when first argument is an object class).

g) The specified class attributes are available attributes of the specified object class (when first argument is an object class).

h) Save not in progress.

i) Restore not in progress.

6.17.4 Postconditions

a) The request for the updated attribute values has been received by the RTI.

6.17.5 Exceptions

a) The object instance is not known (if an object instance designator was specified).

b) The object class is not defined in FDD (if an object class designator was specified).

c) The class attribute is not available at the known class of the object instance.

d) The federate is not a federation execution member.

e) Save in progress.

f) Restore in progress.

g) RTI internal error.

6.17.6 Reference state charts

a) None

6.18 Provide Attribute Value Update †

The *Provide Attribute Value Update †* service shall request the current values for attributes owned by the joined federate for a given object instance. The joined federate shall respond to the *Provide Attribute Value Update †* service with an invocation of the *Update Attribute Values* service to provide the requested instance attribute values to the federation.

The user-supplied tag argument supplied to the corresponding *Request Attribute Value Update* service invocation shall be provided with all corresponding *Provide Attribute Value Update †* service invocations.

6.18.1 Supplied arguments

a) Object instance designator

b) Set of attribute designators

c) User-supplied tag

6.18.2 Returned arguments

a) None

6.18.3 Preconditions

a) The federation execution exists.

b) The federate is joined to that federation execution.

c) The joined federate knows about the object instance with the specified designator.

d) The joined federate owns the specified instance attributes.

6.18.4 Postconditions

a) The joined federate has been notified to provide updates of the specified instance attribute values.

6.18.5 Exceptions

a) The object instance is not known

b) The attribute designator is not recognized

c) The instance attribute is not owned

d) Federate internal error

6.18.6 Reference state charts

a) Figure 14: Implications of ownership of instance attribute (i, k, j)

6.19 Turn Updates On For Object Instance †

The *Turn Updates On For Object Instance †* service shall indicate to the joined federate that the values of the specified attributes of the specified object instance are required somewhere in the federation execution. The joined federate should commence with the federation-agreed-upon update scheme for the specified instance attributes. Generation of the *Turn Updates On For Object Instance †* service advisory can be controlled using the *Enable/Disable Attribute Relevance Advisory Switch* services. The *Turn Updates On For Object Instance †* service shall be invoked only when the Attribute Relevance Advisory Switch is enabled.

6.19.1 Supplied arguments

a) Object instance designator

b) Set of attribute designators

6.19.2 Returned arguments

a) None

6.19.3 Preconditions

a) The federation execution exists.

b) The federate is joined to that federation execution.

c) The joined federate owns the instance attributes.

d) The joined federate knows about the object instance with the specified designator.

e) For each attribute designator specified, there is at least one joined federate in the federation execution for which the instance attribute is in scope via an active subscription.

f) The Attribute Relevance Advisory Switch is enabled.

6.19.4 Postconditions

a) The joined federate has been notified by another joined federate in the federation execution of the requirement for updates of the specified attributes of the specified object instance.

6.19.5 Exceptions

a) The object instance is not known.

b) The attribute designator is not recognized.

c) The instance attribute is not owned.

d) Federate internal error.

6.19.6 Reference state charts

a) Figure 14: Implications of ownership of instance attribute (i, k, j)

6.20 Turn Updates Off For Object Instance †

The *Turn Updates Off For Object Instance †* service shall indicate to the joined federate that the values of the specified attributes of the object instance are not required anywhere in the federation execution. Generation of the *Turn Updates Off For Object Instance †* service advisory can be controlled using the *Enable/Disable Attribute Relevance Advisory Switch* services. The *Turn Updates Off For Object Instance †* service shall be invoked only when the Attribute Relevance Advisory Switch is enabled.

6.20.1 Supplied arguments

a) Object instance designator

b) Set of attribute designators

6.20.2 Returned arguments

a) None

6.20.3 Preconditions

a) The federation execution exists.

b) The federate is joined to that federation execution.

c) The joined federate owns the specified instance attributes.

d) The joined federate knows about the object instance with the specified designator.

e) For each attribute designator specified, there is no other joined federate in the federation execution for which the instance attribute is in scope via an active subscription.

f) The attribute relevance advisory switch is enabled.

6.20.4 Postconditions

a) The joined federate has been notified by another joined federate in the federation execution that updates of the specified attributes of the specified object instance are not required.

6.20.5 Exceptions

a) The object instance is not known.

b) The attribute designator is not recognized.

c) The attribute is not owned.

d) Federate internal error.

6.20.6 Reference state charts

a) Figure 14: Implications of ownership of instance attribute (i, k, j)

7. Ownership management

7.1 Overview

Ownership management shall be used by joined federates and the RTI to transfer ownership of instance attributes among joined federates. The ability to transfer ownership of instance attributes among joined federates shall be required to support the cooperative modeling of a given object instance across a federation. Only the joined federate that owns an instance attribute shall

— Invoke the *Update Attribute Values* service to provide a new value for that instance attribute

— Receive invocations of the *Provide Attribute Value Update †* service for that instance attribute

— Shall receive invocations of the *Turn Updates On For Object Instance †* and *Turn Updates Off For Object Instance †* services pertaining to that instance attribute

Figure 15 depicts the ways that ownership of a single instance attribute may be established from the viewpoint of a given joined federate. This diagram is complete insofar as all transitions shown represent legal operations, and transitions that are not shown represent illegal operations. Illegal operations shall generate exceptions if invoked.

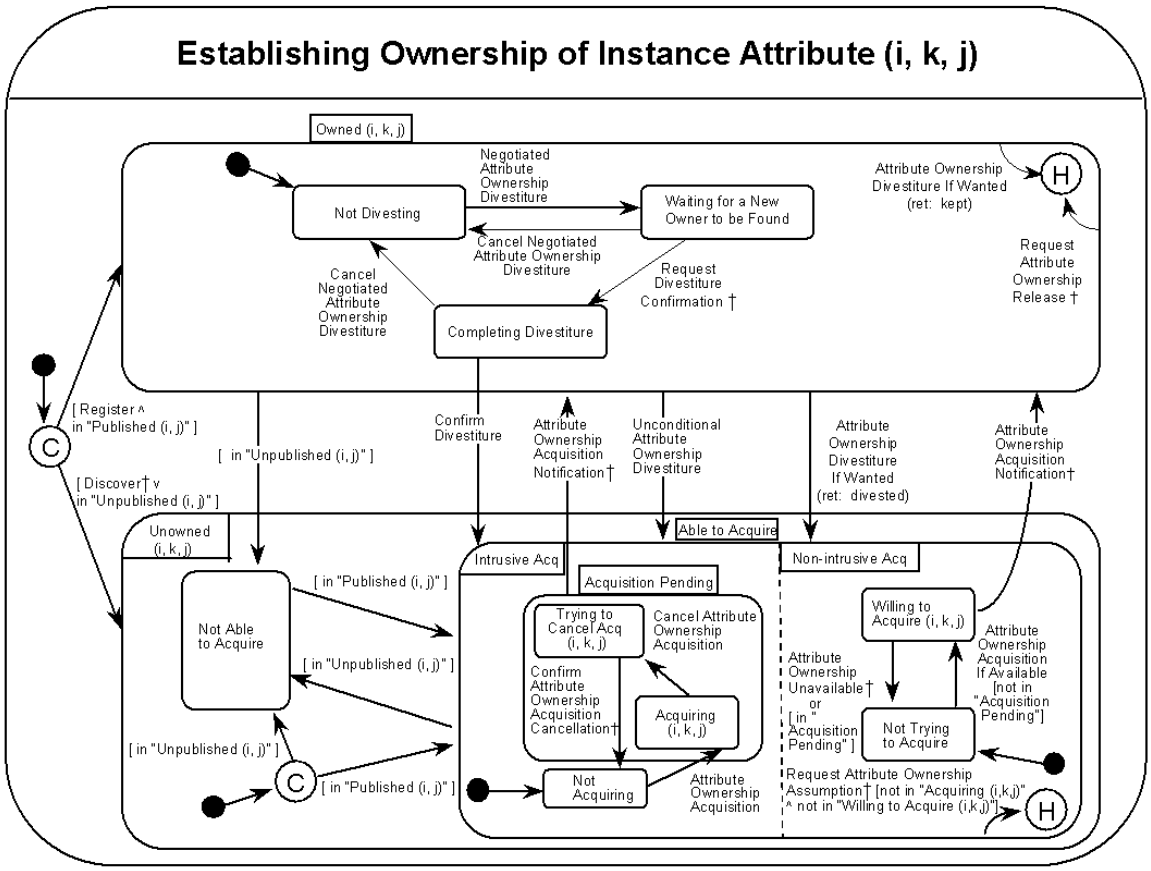

Figure 15—Establishing ownership of instance attribute (i, k, j)

An instance attribute shall not be owned by more than one joined federate at any given time, and an instance attribute may be unowned by all joined federates. The RTI shall be responsible for attempting to find an owner for instance attributes that are left unowned (either via registration, federate resignation, or some form of divestiture). The RID provided to an RTI may allow the federation to control how often or for how long an RTI will attempt to find an owner for unowned instance attributes. From a given joined federate's perspective, every instance attribute shall be either owned or unowned. Hence, within the state machine depicted in Figure 15, the owned and unowned states are exclusive.

Upon registration of an object instance, the registering joined federate shall own all instance attributes of that object instance for which the joined federate is publishing the corresponding class attributes at the registered class of the object instance. All other instance attributes of that object instance shall be unowned by all joined federates. Upon discovery of an object instance, the discovering joined federate shall not own any instance attributes of that object instance. If a joined federate does not own an instance attribute, it shall not own that instance attribute until it has received an *Attribute Ownership Acquisition Notification †* service invocation for it.

Ownership of an instance attribute shall be transferred from one joined federate to another either by the owning joined federate requesting to divest itself of the instance attribute or by a nonowning joined federate requesting to acquire it. Whether an instance attribute changes ownership as a result of being divested by its owner or acquired by a non-owner, however, the instance attribute shall change ownership only as a result of explicit service invocations by the owning and acquiring joined federates. Ownership shall not be taken away from, nor shall it be given to, a joined federate without the joined federate's consent, with the exception of the following MOM interactions:

— HLAmanager.HLAfederate.HLAadjust.HLAmodifyAttributeState

— HLAmanager.HLAfederate.HLAservice.HLAunpublishObjectClassAttributes

— HLAmanager.HLAfederate.HLAservice.HLAunconditionalAttributeOwnershipDivestiture

— HLAmanager.HLAfederate.HLAservice.HLAresignFederationExecution

7.1.1 Ownership and publication

The ownership of an instance attribute is closely related to whether that instance attribute's corresponding class attribute is published at the known class of the instance attribute. This means that the ownership state machine in Figure 15, which operates in parallel with the publication state machine in Figure 10, also shares interdependencies with the publication state machine. A joined federate shall be publishing a class attribute at the known class of an object instance in order to own the corresponding instance attribute of that object instance. This means the following:

a) A joined federate shall be publishing a class attribute at the known class of an object instance before it may become the owner of the corresponding instance attribute of that object instance. This interdependency between ownership and publication is expressed in Figure 15 by the Not Able to Acquire state, the [in "Unpublished (i, j)"] and [in "Published (i, j)"] transitions in the Unowned (i, k, j) state, and the conditional transition into the Owned (i, k, j) and Unowned (i, k, j) states from the start state.

b) If the joined federate that owns an instance attribute stops publishing the corresponding class attribute at the known class of the instance attribute, the instance attribute shall immediately become unowned. This interdependency between ownership and publication is expressed in Figure 15 by the transition from the Owned (i, k, j) to the Unowned (i, k, j) state that is labeled [in "Unpublished (i, j)"]. As depicted by the guard on the transition from the Published (i, j) to the Unpublished (i, j) state in the publication state machine shown in Figure 10, a joined federate shall not stop publication of a class attribute at a given class if there is an object instance that has that

class as its known class and that has a corresponding instance attribute that is in either the Acquisition Pending or Willing to Acquire (i, k, j) state at that joined federate. That is, a joined federate shall not stop publishing a class attribute at a given class if there is an object instance that has that class as its known class and that has a corresponding instance attribute for which the joined federate has invoked the following:

1) *Attribute Ownership Acquisition* service, but has not yet received an invocation of either the *Confirm Attribute Ownership Acquisition Cancellation †* service or the *Attribute Ownership Acquisition Notification †* service

2) *Attribute Ownership Acquisition If Available* service, but has not yet received an invocation of the *Attribute Ownership Unavailable †* service, received an invocation of the *Attribute Ownership Acquisition Notification †* service, or invoked the *Attribute Ownership Acquisition service* [after which condition 1) (above) applies].

7.1.2 Ownership transfer

An instance attribute that is successfully divested shall become unowned by the divesting joined federate. If an instance attribute is unowned, its corresponding class attribute at the known class of the instance attribute may be either published or unpublished. If the class attribute is published at that class, the joined federate shall be eligible to acquire the corresponding instance attribute and it may be offered ownership of that instance attribute by the RTI via the *Request Attribute Ownership Assumption †* service. There are five ways in which an owning joined federate may attempt to divest itself of an instance attribute and two ways in which a nonowning joined federate may attempt to acquire one.

7.1.2.1 Divestiture

The five actions that a joined federate may take to cause an instance attribute that it owns to become unowned are as follows:

a) The joined federate may invoke the *Unconditional Attribute Ownership Divestiture* service, in which case, the instance attribute shall immediately become unowned by that joined federate and, in fact, by all joined federates.

b) The joined federate may invoke the *Negotiated Attribute Ownership Divestiture* service, which notifies the RTI that the joined federate wishes to divest itself of the instance attribute, providing that the RTI can locate a joined federate that is willing to own the instance attribute. If any joined federates are in the process of trying to acquire the instance attribute, these joined federates are willing to own the instance attribute. The RTI can try to identify other joined federates that are willing to own the instance attribute by invoking the *Request Attribute Ownership Assumption †* service at all joined federates that are not in the process of trying to acquire the instance attribute, but that are publishing the instance attribute's corresponding class attribute at the publishing federate's known class of the instance attribute. If the RTI is able to locate a joined federate that is willing to acquire the instance attribute, the RTI shall notify the divesting joined federate by invoking the *Request Divestiture Confirmation †* service. The divesting joined federate may then complete the negotiated divestiture using the *Confirm Divestiture* service, which causes ownership of the instance attribute to be lost.

c) The joined federate may invoke the *Attribute Ownership Divestiture If Wanted* service, which notifies the RTI that the joined federate wishes to divest itself of ownership of the instance attribute, providing that another joined federate is attempting to acquire ownership of it. This service invocation has a return argument that the RTI shall use to indicate the set of instance attributes for which the joined federate has divested ownership. If an instance attribute is not in the returned instance attribute set, it was not divested and the request to divest this instance attribute shall be concluded. If the *Attribute Ownership Divestiture If Wanted* service returns with the designated

instance attribute among the set of released instance attributes, the instance attribute shall be unowned by the invoking joined federate. [In Figure 15, the transition from the owned to the unowned state via an *Attribute Ownership Divestiture If Wanted* service invocation is labeled "*Attribute Ownership Divestiture If Wanted* (ret: divested)." This is a convenience notation indicating that the instance attribute in question is a member of the returned instance attribute set.]

d) The joined federate may stop publishing the instance attribute's corresponding class attribute at the known class of the instance attribute, which shall result in the instance attribute immediately becoming unowned by that joined federate and, in fact, by all joined federates.

e) The joined federate may resign from the federation execution. When a joined federate successfully resigns from the federation execution with the Unconditionally Divest Attributes option, all of the instance attributes that are owned by that joined federate shall become unowned by that joined federate and, in fact, by all joined federates. This transition is not depicted in Figure 15 because it occurs at a joined federate, rather than at an instance attribute, level of operation.

Of the five ways a joined federate may divest itself of an instance attribute, only the *Negotiated Attribute Ownership Divestiture* service may be canceled. A *Negotiated Attribute Ownership Divestiture* service invocation shall remain pending until either the instance attribute becomes unowned or the divesting joined federate cancels the divestiture request by invoking the *Cancel Negotiated Attribute Ownership Divestiture* service. Cancellation of the divestiture shall be guaranteed to be successful.

Of the five ways a joined federate may divest itself of an instance attribute, three ways (invocation of the *Unconditional Attribute Ownership Divestiture* service, a request to stop publication of the instance attribute's corresponding class attribute at the known class of the instance attribute, and invocation of the *Resign Federation Execution* service) shall result in the instance attribute becoming unowned by all joined federates. When either the *Negotiated Attribute Ownership Divestiture* or the *Attribute Ownership Divestiture If Wanted* service is used, the RTI shall guarantee that immediately after the owning joined federate loses ownership of the instance attribute, another joined federate shall be granted ownership of it. For purposes of determining an instance attribute's scope, the instance attribute shall be considered to be continuously owned during its transfer of ownership from the divesting joined federate to the acquiring joined federate via either the *Negotiated Attribute Ownership Divestiture* or the *Attribute Ownership Divestiture If Wanted* service.

When the value of an instance attribute is being updated by a joined federate using TSO messages and that joined federate is initiating a negotiated divestiture of that instance attribute, the divesting and receiving joined federates should be aware of potential causality or reflection anomalies.

Because the divestiture is not associated with a time stamp, the logical time of the receiving joined federate at which ownership of the instance attribute is acquired may be earlier or later than that of the divesting joined federate, giving either a gap or an overlap in the time intervals on the HLA time axis for which the joined federates own the instance attribute.

Some issues to consider are as follows:

a) There may be points on the HLA time axis at which neither joined federate owns the instance attribute. This may result in an interval of time stamps for which no updates to the value of the instance attribute are updated, unless special steps are taken by one or both of the joined federates.

b) There may be points on the HLA time axis at which both joined federates own the instance attribute. This may result in updates to the value of the instance attribute being produced by both federates that contain the same time stamp, unless special steps are taken by one or both of the joined federates.

c) There may be TSO updates sent by the divesting joined federate that have not been delivered to other joined federates (i.e., they are queued in the RTI) when the divestiture occurs. Some of these outstanding TSO updates may be retractable by the divesting joined federate, others may not be retractable. See restrictions in 8.21 for details.

d) There may be additional information that the divesting joined federate needs to transfer along with ownership of the instance attribute.

7.1.2.2 Acquisition

There shall be the following two ways for a joined federate that is publishing a class attribute at a given class to acquire a corresponding instance attribute of an object instance that has that class as the known class at the federate attempting to acquire the attribute:

a) The joined federate may invoke the *Attribute Ownership Acquisition* service, which shall inform the RTI that it shall invoke the *Request Attribute Ownership Release †* service at the joined federate that owns the designated instance attribute.

b) The joined federate may invoke the *Attribute Ownership Acquisition If Available* service, which shall inform the RTI that it wants to acquire the designated instance attribute only if it is already unowned by all joined federates or if it is in the process of being divested by its owner.

The first method of acquisition can be thought of as an intrusive acquisition, because the RTI will notify the joined federate that owns the instance attribute that another joined federate wants to acquire it and request that the owning joined federate release the instance attribute for acquisition by the requesting joined federate. The second method of acquisition can be thought of as a nonintrusive acquisition because the RTI will not notify the owning joined federate of the request to acquire the instance attribute. The *Attribute Ownership Acquisition* service can also be thought of as taking precedence over the *Attribute Ownership Acquisition If Available* service. A joined federate that has invoked the *Attribute Ownership Acquisition* service and is in the Acquisition Pending state shall not invoke the *Attribute Ownership Acquisition If Available* service. If a joined federate that has invoked the *Attribute Ownership Acquisition If Available* service and is in the Willing to Acquire state invokes the *Attribute Ownership Acquisition* service, that joined federate shall enter the Acquisition Pending state.

An *Attribute Ownership Acquisition* service invocation may be explicitly canceled, but an *Attribute Ownership Acquisition If Available* service invocation may not be explicitly canceled. When a joined federate invokes the *Attribute Ownership Acquisition If Available* service, either the *Attribute Ownership Acquisition Notification †* service or the *Attribute Ownership Unavailable †* service shall be invoked at that joined federate in response. (If the instance attribute is unowned by all joined federates or in the process of being divested by its owner, the *Attribute Ownership Acquisition Notification †* service shall be invoked. Otherwise, the *Attribute Ownership Unavailable †* service shall be invoked.)

When a joined federate invokes the *Attribute Ownership Acquisition* service invocation, on the other hand, this request shall remain pending until either the instance attribute is acquired (as indicated by an invocation of the *Attribute Ownership Acquisition Notification †* service) or the joined federate successfully cancels the acquisition request. A joined federate may attempt to cancel the acquisition request by invoking the *Cancel Attribute Ownership Acquisition* service. The *Cancel Attribute Ownership Acquisition* service is not guaranteed to be successful. If it is successful, the RTI shall indicate this success to the canceling joined federate by invoking the *Confirm Attribute Ownership Acquisition Cancellation †* service. If it fails, the RTI shall indicate this failure to the canceling joined federate by invoking the *Attribute Ownership Acquisition Notification †* service, thereby granting ownership of the instance attribute to the joined federate.

An *Attribute Ownership Acquisition* service invocation shall override an *Attribute Ownership Acquisition If Available* service invocation. This means that a joined federate that has invoked the *Attribute Ownership Acquisition If Available* service may, before it receives an invocation of either the *Attribute Ownership*

Acquisition Notification † service or the *Attribute Ownership Unavailable †* service, invoke the *Attribute Ownership Acquisition* service. In this case, the *Attribute Ownership Acquisition If Available* service request shall be implicitly canceled and the *Attribute Ownership Acquisition* service request shall remain pending until either the instance attribute is acquired or the joined federate successfully cancels the acquisition request. A joined federate that has invoked the *Attribute Ownership Acquisition* service, but has not yet received an invocation of either the *Attribute Ownership Acquisition Notification †* service or the *Confirm Attribute Ownership Acquisition Cancellation †* service, shall not invoke the *Attribute Ownership Acquisition If Available* service.

7.1.3 Privilege to delete object

All object classes shall have an available attribute called **HLAprivilegeToDeleteObject**. As with all other available attributes, a joined federate shall be publishing the **HLAprivilegeToDeleteObject** class attribute at the known class of an object instance in order to be eligible for ownership of a corresponding **HLAprivilegeToDeleteObject** instance attribute.

A **HLAprivilegeToDeleteObject** instance attribute shall be transferable among joined federates. All ownership management services for **HLAprivilegeToDeleteObject** instance attributes shall act the same as they are for all other instance attributes. The reason that a joined federate may want to own the **HLAprivilegeToDeleteObject** instance attribute, however, is different. Ownership of a typical instance attribute shall give a joined federate the privilege to provide new values for that instance attribute. Ownership of the **HLAprivilegeToDeleteObject** instance attribute of an object instance shall give the joined federate the additional right to delete that object instance from the federation execution.

7.1.4 User-supplied tags

Several of the ownership management services take a user-supplied tag as an argument. These arguments shall be provided as a mechanism for conducting information between joined federates that could be used to implement priority or other schemes. Although the content and use of these tags are outside of the scope of this specification, the RTI shall pass these user-supplied tags from joined federates that are trying to acquire an instance attribute to the joined federate that owns the instance attribute, and from the joined federate that is trying to divest itself of an instance attribute to the joined federates that are able to acquire the instance attribute. In particular, the user-supplied tag present in the

— *Negotiated Attribute Ownership Divestiture* service shall be present in any resulting *Request Attribute Ownership Assumption †* service invocations

— *Confirm Divestiture* service shall be present in any resulting *Attribute Ownership Acquisition Notification †* service invocations

— *Attribute Ownership Acquisition* service shall be present in any resulting *Request Attribute Ownership Release †* service invocations

7.1.5 Sets of attribute designators

Although many of the ownership management services take a set of instance attributes as an argument, the RTI treats ownership management operations on a per-instance-attribute basis. The fact that some ownership management service invocations take sets of instance attributes as an argument is a feature provided to joined federate designers for convenience. A single request with an instance attribute set as an argument can result in multiple responses pertaining to disjoint subsets of those instance attributes. For example, a single *Negotiated Attribute Ownership Divestiture* that has a set of instance attributes as an argument could result in multiple *Request Divestiture Confirmation †* service invocations. If one instance attribute in the set of instance attributes provided as an argument to an ownership management service invocation violates the preconditions of the service, an exception shall be generated and the entire service invocation shall fail. The

RTI shall treat all federate requests to divest or acquire attributes separately. Responses to or requests of federates shall not contain information from separate federate requests.

7.2 Unconditional Attribute Ownership Divestiture

The *Unconditional Attribute Ownership Divestiture* service shall notify the RTI that the joined federate no longer wants to own the specified instance attributes of the specified object instance. This service shall immediately relieve the divesting joined federate of the ownership, causing the instance attribute(s) to go (possibly temporarily) into the unowned state, without regard to the existence of an accepting joined federate.

7.2.1 Supplied arguments

a) Object instance designator

b) Set of attribute designators

7.2.2 Returned arguments

a) None.

7.2.3 Preconditions.

a) The federation execution exists.

b) The federate is joined to that federation execution.

c) An object instance with the specified designator exists.

d) The joined federate knows about the object instance with the specified designator.

e) The joined federate owns the specified instance attributes.

f) Save not in progress.

g) Restore not in progress.

7.2.4 Postconditions

a) The joined federate no longer owns the specified instance attributes.

7.2.5 Exceptions

a) The object instance is not known.

b) The class attribute is not available at the known class of the object instance.

c) The joined federate does not own the instance attribute.

d) The federate is not a federation execution member.

e) Save in progress.

f) Restore in progress.

g) RTI internal error.

7.2.6 Reference state charts

a) Figure 15: Establishing ownership of instance attribute (i, k, j)

7.3 Negotiated Attribute Ownership Divestiture

The *Negotiated Attribute Ownership Divestiture* service shall notify the RTI that the joined federate no longer wants to own the specified instance attributes of the specified object instance. Ownership shall be transferred only if some joined federate(s) accepts. When the RTI finds federates willing to accept ownership of any or all of the instance attributes, it will inform the divesting federate using the *Request Divestiture Confirmation †* service (supplying the appropriate instance attributes as arguments). The divesting federate may then complete the negotiated divestiture by invoking the *Confirm Divestiture* service to inform the RTI of which instance attributes it is divesting ownership. The invoking joined federate shall continue its update responsibility for the specified instance attributes until it divests ownership via the *Confirm Divestiture* service. The joined federate may receive one or more *Request Divestiture Confirmation †* invocations for each invocation of this service because different joined federates may wish to become the owner of different instance attributes.

A request to divest ownership shall remain pending until either the request is completed (via the *Request Divestiture Confirmation †* and *Confirm Divestiture* services), the requesting joined federate successfully cancels the request (via the *Cancel Negotiated Attribute Ownership Divestiture* service), or the joined federate divests itself of ownership by other means (e.g., the *Attribute Ownership Divestiture If Wanted* or *Unpublish Object Class Attributes* service). A second negotiated divestiture for an instance attribute already in the process of a negotiated divestiture shall not be legal.

7.3.1 Supplied arguments

a) Object instance designator

b) Set of attribute designators

c) User-supplied tag

7.3.2 Returned arguments

a) None

7.3.3 Preconditions

a) The federation execution exists.

b) The federate is joined to that federation execution.

c) An object instance with the specified designator exists.

d) The joined federate knows about the object instance with the specified designator.

e) The joined federate owns the specified instance attributes.

f) The specified instance attributes are not in the negotiated divestiture process.

g) Save not in progress.

h) Restore not in progress.

7.3.4 Postconditions

a) No change has occurred in instance attribute ownership.

b) The RTI has been notified of the joined federate's request to divest ownership of the specified instance attributes.

7.3.5 Exceptions

a) The object instance is not known.

b) The class attribute is not available at the known class of the object instance.

c) The joined federate does not own the instance attribute.

d) The instance attribute is already in the negotiated divestiture process.

e) The federate is not a federation execution member.

f) Save in progress.

g) Restore in progress.

h) RTI internal error.

7.3.6 Reference state charts

a) Figure 15: Establishing ownership of instance attribute (i, k, j)

7.4 Request Attribute Ownership Assumption †

The *Request Attribute Ownership Assumption †* service shall inform the joined federate that the specified instance attributes are available for transfer of ownership to the joined federate. The RTI shall supply an object instance designator and set of attribute designators. The joined federate may return a subset of the supplied attribute designators for which it is willing to assume ownership via the *Attribute Ownership Acquisition* service or via the *Attribute Ownership Acquisition If Available* service. In the case that the supplied instance attributes are unowned as a result of a joined federate invoking the *Unconditional Attribute Ownership Divestiture* service, the divesting joined federate shall not be asked to assume ownership.

7.4.1 Supplied arguments

a) Object instance designator

b) Set of attribute designators

c) User-supplied tag

7.4.2 Returned arguments

a) None

7.4.3 Preconditions

a) The federation execution exists.

b) The federate is joined to that federation execution.

c) The joined federate knows about the object instance with the specified designator.

d) The joined federate is publishing the corresponding class attributes at the known class of the specified object instance.

e) The joined federate does not own the specified instance attributes.

7.4.4 Postconditions

a) Instance attribute ownership has not changed.

b) The joined federate has been informed of the set of instance attributes for which the RTI is requesting that the joined federate assume ownership.

7.4.5 Exceptions

a) The object instance is not known.

b) The attribute designator is not recognized.

c) The joined federate already owns the instance attribute.

d) The joined federate is not publishing the class attribute at the known class of the object instance.

e) Federate internal error.

7.4.6 Reference state charts

a) Figure 15: Establishing Ownership of Instance Attribute (i, k, j)

7.5 Request Divestiture Confirmation †

The *Request Divestiture Confirmation †* service shall notify the joined federate that new owners have been found for the specified instance attributes, and that the negotiated divestiture of the specified instance attributes can now be completed at the joined federate's discretion. The joined federate can either complete the negotiated divestiture of the specified instance attributes using the *Confirm Divestiture* service, divest ownership of the instance attributes by some other means (e.g., using the *Unconditional Attribute Ownership Divestiture* service), or it can cancel the negotiated divestiture using the *Cancel Negotiated Attribute Ownership Divestiture* service. Ownership of the specified instance attributes is not lost upon invocation of this service; ownership will only be divested if the joined federate confirms its intent to divest the instance attributes or divests ownership by some other means.

7.5.1 Supplied arguments

a) Object instance designator

b) Set of attribute designators

7.5.2 Returned arguments

a) None

7.5.3 Preconditions

a) The federation execution exists.

b) The federate is joined to that federation execution.

c) The joined federate knows about the object instance with the specified designator.

d) The joined federate owns the specified instance attributes.

e) The joined federate has previous attempted to divest ownership of the specified instance attributes and has not subsequently canceled that request.

7.5.4 Postconditions

a) The joined federate has been informed that new owners have been found for the specified instance attributes, and that ownership of the instance attributes will be divested as soon as the joined federate confirms its intent to divest.

7.5.5 Exceptions

a) The object instance is not known.

b) The attribute designator is not recognized.

c) The joined federate does not own the instance attributes.

d) The joined federate had not previously attempted to divest ownership of the instance attributes.

e) Federate internal error.

7.6 Confirm Divestiture

The *Confirm Divestiture* service shall inform the RTI that the joined federate wants to complete negotiated divestiture for the specified instance attributes. After invocation of this service, the joined federate shall not own any of the specified instance attributes, and the RTI shall give ownership of the instance attributes to another joined federate.

7.6.1 Supplied arguments

a) Object instance designator

b) Set of attribute designators

c) User-supplied tag

7.6.2 Returned arguments

a) None

7.6.3 Preconditions

a) The federation execution exists.

b) The federate is joined to that federation execution.

c) An object instance with the specified designator exists.

d) The joined federate knows about the object instance with the specified designator.

e) The joined federate owns the specified instance attributes.

f) The joined federate has been asked to confirm its request for a negotiated divestiture of the specified instance attributes.

g) Save not in progress.

h) Restore not in progress.

7.6.4 Postconditions

a) The joined federate no longer owns the specified instance attributes.

7.6.5 Exceptions

a) The object instance is not known.

b) The class attribute is not available at the known class of the object instance.

c) The joined federate does not own the instance attributes.

d) The joined federate had not previously been asked to confirm its request for negotiated divestiture of the instance attributes.

e) The federate is not a federation execution member.

f) Save in progress.

g) Restore in progress.

h) RTI internal error.

7.7 Attribute Ownership Acquisition Notification †

The *Attribute Ownership Acquisition Notification †* service shall notify the joined federate that it now owns the specified set of instance attributes. The joined federate may then begin updating those instance attribute values. The joined federate may receive multiple notifications for a single invocation of the *Attribute Ownership Acquisition* service because the joined federate may wish to become the owner of instance attributes owned by different joined federates.

7.7.1 Supplied arguments

a) Object instance designator

b) Set of attribute designators

c) User-supplied tag

7.7.2 Returned arguments

a) None

7.7.3 Preconditions

a) The federation execution exists.

b) The federate is joined to that federation execution.

c) The joined federate knows about the object instance with the specified designator.

d) The joined federate is publishing the corresponding class attributes at the known class of the specified object instance.

e) The joined federate has previously attempted to acquire ownership of the specified instance attributes.

f) The specified instance attributes are not owned by any joined federate in the federation execution.

7.7.4 Postconditions

a) The joined federate owns the specified instance attributes.

b) The joined federate may stop publishing the corresponding class attributes at the known class of the specified object instance if it does not own any corresponding instance attributes for which the joined federate has either invoked the

   1) *Attribute Ownership Acquisition* service, and has not yet received a corresponding invocation of either the *Confirm Attribute Ownership Acquisition Cancellation †* service or the *Attribute Ownership Acquisition Notification †* service, or

   2) *Attribute Ownership Acquisition If Available* service, and has not yet received a corresponding invocation of either the *Attribute Ownership Unavailable †* service or the *Attribute Ownership Acquisition Notification †* service, or

   3) *Attribute Ownership Acquisition If Available* service and has subsequently invoked the *Attribute Ownership Acquisition* service [after which condition 1) applies].

c) May cause an implicit out-of-scope situation for affected instance attributes at the joined federate.

7.7.5 Exceptions

a) The object instance is not known.

b) The attribute designator is not recognized.

c) The joined federate had not previously attempted to acquire ownership of the instance attribute.

d) The joined federate already owns the instance attribute.

e) The joined federate is not publishing the class attribute at the known class of the object instance.

f) Federate internal error.

7.7.6 Reference state charts

a) Figure 15: Establishing ownership of instance attribute (i, k, j)

7.8 Attribute Ownership Acquisition

The *Attribute Ownership Acquisition* service shall request the ownership of the specified instance attributes of the specified object instance. If a specified instance attribute is owned by another joined federate, the RTI shall invoke the *Request Attribute Ownership Release †* service for that instance attribute at the owning joined federate. The joined federate may receive one or more *Attribute Ownership Acquisition Notification †* invocations for each invocation of this service.

A request to acquire ownership shall remain pending until either the request is granted (via the *Attribute Ownership Acquisition Notification †* service) or the requesting joined federate successfully cancels the request (via the *Cancel Attribute Ownership Acquisition* and *Confirm Attribute Ownership Acquisition Cancellation †* services).

7.8.1 Supplied arguments

a) Object instance designator

b) Set of attribute designators

c) User-supplied tag

7.8.2 Returned arguments

a) None

7.8.3 Preconditions

a) The federation execution exists.

b) The federate is joined to that federation execution.

c) An object instance with the specified designator exists.

d) The joined federate knows about the object instance with the specified designator.

e) The joined federate is publishing the corresponding class attributes at the known class of the specified object instance.

f) The joined federate does not own the specified instance attributes.

g) Save not in progress.

h) Restore not in progress.

7.8.4 Postconditions

a) The RTI has been informed of the joined federate's request to acquire ownership of the specified instance attributes.

b) The joined federate shall not stop publishing the corresponding class attributes at the known class of the specified object instance.

7.8.5 Exceptions

a) The object instance is not known.

b) The joined federate is not publishing the object class.

c) The class attribute is not available at the known class of the object instance.

d) The joined federate is not publishing the class attribute at the known class of the object instance.

e) The joined federate already owns the instance attribute.

f) The federate is not a federation execution member.

g) Save in progress.

h) Restore in progress.

i) RTI internal error.

7.8.6 Reference state charts

a) Figure 15: Establishing ownership of instance attribute (i, k, j)

7.9 Attribute Ownership Acquisition If Available

The *Attribute Ownership Acquisition If Available* service shall request the ownership of the specified instance attributes of the specified object instance only if the instance attribute is unowned by all joined federates or it is in the process of being divested by its owner. If a specified instance attribute is owned by

another joined federate, the RTI shall not invoke the *Request Attribute Ownership Release †* service for that instance attribute at the owning joined federate. For each of the specified instance attributes, the joined federate shall receive either a corresponding *Attribute Ownership Acquisition Notification †* service invocation or a corresponding *Attribute Ownership Unavailable †* service invocation.

7.9.1 Supplied arguments

a) Object instance designator

b) Set of attribute designators

7.9.2 Returned arguments

a) None

7.9.3 Preconditions

a) The federation execution exists.

b) The federate is joined to that federation execution.

c) An object instance with the specified designator exists.

d) The joined federate knows about the object instance with the specified designator.

e) The joined federate is publishing the corresponding class attributes at the known class of the specified object instance.

f) The joined federate does not own the specified instance attributes.

g) For each of the specified instance attributes, it is not the case that the joined federate has invoked the *Attribute Ownership Acquisition* service, but has not yet received an invocation of either the *Confirm Attribute Ownership Acquisition Cancellation †* service or the *Attribute Ownership Acquisition Notification †* service.

h) Save not in progress.

i) Restore not in progress.

7.9.4 Postconditions

a) The RTI has been informed of the joined federate's request to acquire ownership of the specified instance attributes. The joined federate shall not stop publishing the corresponding class attributes at the known class of the specified object instance.

7.9.5 Exceptions

a) The object instance is not known.

b) The joined federate is not publishing the object class.

c) The class attribute is not available at the known class of the object instance.

d) The joined federate is not publishing the class attribute at the known class of the object instance.

e) The joined federate already owns the instance attribute.

f) The joined federate is already acquiring the instance attribute.

g) The federate is not a federation execution member.

h) Save in progress.

i) Restore in progress.

j) RTI internal error.

7.9.6 Reference state charts

a) Figure 15: Establishing ownership of instance attribute (i, k, j)

7.10 Attribute Ownership Unavailable †

The *Attribute Ownership Unavailable †* service shall inform the joined federate that the specified instance attributes were not available for ownership acquisition.

7.10.1 Supplied arguments

a) Object instance designator

b) Set of attribute designators

7.10.2 Returned arguments

a) None

7.10.3 Preconditions

a) The federation execution exists.

b) The federate is joined to that federation execution.

c) The joined federate knows about the object instance with the specified designator.

d) The joined federate had requested ownership acquisition (if available) for the specified instance attributes.

e) The joined federate does not own the specified instance attributes.

7.10.4 Postconditions

a) The joined federate has been informed that the specified instance attributes were not available for ownership acquisition.

b) The joined federate may stop publishing the corresponding class attributes at the known class of the specified object instance.

7.10.5 Exceptions

a) The object instance is not known.

b) The attribute designator is not recognized.

c) The joined federate already owns the instance attribute.

d) The joined federate had not requested ownership acquisition (if available) for the instance attribute.

e) Federate internal error.

7.10.6 Reference state charts

a) Figure 15: Establishing Ownership of Instance Attribute (i, k, j)

7.11 Request Attribute Ownership Release †

The *Request Attribute Ownership Release †* service shall request that the joined federate release ownership of the specified instance attributes of the specified object instance. The *Request Attribute Ownership Release †* service shall provide an object instance designator and set of attribute designators and shall be invoked only as the result of an *Attribute Ownership Acquisition* service invocation by some other joined federate. The joined federate may return the subset of the supplied instance attributes for which it is willing to release ownership via the *Attribute Ownership Divestiture If Wanted* service, the *Unconditional Attribute Ownership Divestiture* service, or the *Negotiated Attribute Ownership Divestiture* service.

7.11.1 Supplied arguments

a) Object instance designator

b) Set of attribute designators

c) User-supplied tag

7.11.2 Returned arguments

a) None

7.11.3 Preconditions

a) The federation execution exists.

b) The federate is joined to that federation execution.

c) The joined federate knows about the object instance with the specified designator.

d) The joined federate owns the specified instance attributes.

7.11.4 Postconditions

a) The joined federate has been informed of the set of instance attributes for which the RTI is requesting the joined federate to release ownership.

7.11.5 Exceptions

a) The object instance is not known

b) The attribute designator is not recognized

c) The joined federate does not own the instance attribute

d) Federate internal error

7.11.6 Reference state charts

a) Figure 15: Establishing ownership of instance attribute (i, k, j)

7.12 Attribute Ownership Divestiture If Wanted

The *Attribute Ownership Divestiture If Wanted* service shall notify the RTI that the joined federate is willing to divest itself of ownership of the specified instance attributes if another joined federate is attempting to acquire ownership of them. The joined federate may use this service to provide an answer to the question posed as a result of the invocation of *Request Attribute Ownership Release †*, or it may use this service at any other time. The returned argument shall indicate the instance attributes for which ownership was actually divested, and it shall be viewed as the conclusion of this attempted divestiture for all supplied instance attributes that are not in the returned argument. Any instance attributes in the returned argument shall be a subset of those instance attributes in the supplied argument.

7.12.1 Supplied arguments

a) Object instance designator

b) Set of attribute designators for which the joined federate is willing to divest ownership

7.12.2 Returned arguments

a) Set of attribute designators for which ownership has actually been divested.

7.12.3 Preconditions

a) The federation execution exists.

b) The federate is joined to that federation execution.

c) An object instance with the specified designator exists.

d) The joined federate knows about the object instance with the specified designator.

e) The joined federate owns the specified instance attributes.

f) Save not in progress.

g) Restore not in progress.

7.12.4 Postconditions

a) Ownership is divested for the instance attributes in the returned argument set.

7.12.5 Exceptions

a) The object instance is not known.

b) The class attribute is not available at the known class of the object instance.

c) The joined federate does not own the instance attribute.

d) The federate is not a federation execution member.

e) Save in progress.

f) Restore in progress.

g) RTI internal error.

7.12.6 Reference state charts

a) Figure 15: Establishing ownership of instance attribute (i, k, j)

7.13 Cancel Negotiated Attribute Ownership Divestiture

The *Cancel Negotiated Attribute Ownership Divestiture* service shall notify the RTI that the joined federate no longer wants to divest ownership of the specified instance attributes.

7.13.1 Supplied arguments

a) Object instance designator

b) Set of attribute designators

7.13.2 Returned arguments

a) None

7.13.3 Preconditions

a) The federation execution exists.

b) The federate is joined to that federation execution.

c) An object instance with the specified designator exists.

d) The joined federate knows about the object instance with the specified designator.

e) The joined federate owns the specified instance attributes.

f) The specified instance attributes were candidates for divestiture.

g) Save not in progress.

h) Restore not in progress.

7.13.4 Postconditions

a) The specified instance attributes are unavailable for divestiture.

7.13.5 Exceptions

a) The object instance is not known.

b) The class attribute is not available at the known class of the object instance.

c) The joined federate does not own the instance attribute.

d) The instance attribute was not a candidate for divestiture.

e) The federate is not a federation execution member.

f) Save in progress.

g) Restore in progress.

h) RTI internal error.

7.13.6 Reference state charts

a) Figure 15: Establishing ownership of instance attribute (i, k, j)

7.14 Cancel Attribute Ownership Acquisition

The *Cancel Attribute Ownership Acquisition* service shall notify the RTI that the joined federate no longer wants to acquire ownership of the specified instance attributes. This service shall always receive one of two replies from the RTI. The first form of reply, *Confirm Attribute Ownership Acquisition Cancellation*, shall indicate that the request to acquire ownership of the specified instance attributes has been successfully canceled. The second form of reply, *Attribute Ownership Acquisition Notification †*, shall indicate that the request to acquire ownership of the specified instance attributes was not canceled in time and that the joined federate has acquired ownership of the instance attributes. The joined federate may receive both forms of reply in response to a single *Cancel Attribute Ownership Acquisition* service invocation since the cancellation may succeed for some of the supplied instance attributes and fail for others. This service shall be used only to cancel requests to acquire ownership of instance attributes that were made via the *Attribute Ownership Acquisition* service. Requests made via the *Attribute Ownership Acquisition If Available* service shall not be explicitly canceled. They may, however, be overridden by an invocation of the *Attribute Ownership Acquisition* service.

7.14.1 Supplied arguments

a) Object instance designator

b) Set of attribute designators

7.14.2 Returned arguments

a) None

7.14.3 Preconditions

a) The federation execution exists.

b) The federate is joined to that federation execution.

c) An object instance with the specified designator exists.

d) The joined federate knows about the object instance with the specified designator.

e) The joined federate does not own the specified instance attributes.

f) The joined federate is attempting to acquire ownership of the specified instance attributes.

g) Save not in progress.

h) Restore not in progress.

7.14.4 Postconditions

a) The RTI has been notified that joined federate no longer wants to acquire ownership of the specified instance attributes.

7.14.5 Exceptions

a) The object instance is not known.

b) The class attribute is not available at the known class of the object instance.

c) The joined federate already owns the instance attribute.

d) The joined federate was not attempting to acquire ownership of the instance attribute.

e) The federate is not a federation execution member.

f) Save in progress.

g) Restore in progress.

h) RTI internal error.

7.14.6 Reference state charts

a) Figure 15: Establishing ownership of instance attribute (i, k, j)

7.15 Confirm Attribute Ownership Acquisition Cancellation †

The *Confirm Attribute Ownership Acquisition Cancellation †* service shall inform the joined federate that the specified instance attributes are no longer candidates for ownership acquisition.

7.15.1 Supplied arguments

a) Object instance designator

b) Set of attribute designators

7.15.2 Returned arguments

a) None

7.15.3 Preconditions

a) The federation execution exists.

b) The federate is joined to that federation execution.

c) The joined federate knows about the object instance with the specified designator.

d) The joined federate had attempted to cancel an ownership acquisition request for the specified instance attributes.

e) The joined federate does not own the specified instance attributes.

7.15.4 Postconditions

a) The specified instance attributes are no longer candidates for acquisition by the joined federate.

b) The joined federate may stop publishing the corresponding class attributes at the known class of the specified object instance.

7.15.5 Exceptions

a) The object instance is not known.

b) The attribute designator is not recognized.

c) The joined federate already owns the instance attribute.

d) The joined federate had not canceled an ownership acquisition request for the instance attribute.

e) Federate internal error.

7.15.6 Reference state charts

a) Figure 15: Establishing ownership of instance attribute (i, k, j)

7.16 Query Attribute Ownership

The *Query Attribute Ownership* service shall be used to determine the owner of the specified instance attribute. The RTI shall provide the instance attribute owner information via the *Inform Attribute Ownership †* service invocation.

7.16.1 Supplied arguments

a) Object instance designator

b) Attribute designator

7.16.2 Returned arguments

a) None

7.16.3 Preconditions

a) The federation execution exists.

b) The federate is joined to that federation execution.

c) An object instance with the specified designator exists.

d) The joined federate knows about the object instance with the specified designator.

e) The corresponding class attribute is an available attribute of the known class of the specified object instance.

f) Save not in progress.

g) Restore not in progress.

7.16.4 Postconditions

a) The request for instance attribute ownership information has been received by the RTI.

7.16.5 Exceptions

a) The object instance is not known.

b) The class attribute is not available at the known class of the object instance.

c) The federate is not a federation execution member.

d) Save in progress.

e) Restore in progress.

f) RTI internal error.

7.16.6 Reference state charts

a) None

7.17 Inform Attribute Ownership †

The *Inform Attribute Ownership †* service shall be used to provide ownership information for the specified instance attribute. This service shall be invoked by the RTI in response to a *Query Attribute Ownership* service invocation by a joined federate. This service shall provide the joined federate a designator of the instance attribute owner (if the instance attribute is owned) or an indication that the instance attribute is available for acquisition.

7.17.1 Supplied arguments

a) Object instance designator

b) Attribute designator

c) Ownership designator (could be a joined federate, RTI, or unowned)

7.17.2 Returned arguments

a) None

7.17.3 Preconditions

a) The federation execution exists.

b) The federate is joined to that federation execution.

c) The joined federate knows about the object instance with the specified designator.

d) The corresponding class attribute is an available attribute of the known class of the specified object instance.

e) The joined federate has previously invoked the *Query Attribute Ownership* service and has not yet received an *Inform Attribute Ownership †* service invocation in response.

7.17.4 Postconditions

a) The joined federate has been informed of the instance attribute ownership.

7.17.5 Exceptions

a) The object instance is not known.

b) The attribute designator is not recognized.

c) Federate internal error.

7.17.6 Reference state charts

a) None

7.18 Is Attribute Owned By Federate

The *Is Attribute Owned By Federate* service shall be used to determine if the specified instance attribute of the specified object instance designator is owned by the invoking joined federate. The service shall return a Boolean value indicating ownership status of the specified instance attribute.

7.18.1 Supplied arguments

a) Object instance designator

b) Attribute designator

7.18.2 Returned arguments

a) Instance attribute ownership indicator

7.18.3 Preconditions

a) The federation execution exists.

b) The federate is joined to that federation execution.

c) An object instance with the specified designator exists.

d) The joined federate knows about the object instance with the specified designator.

e) The corresponding class attribute is an available attribute of the known class of the specified object instance.

f) Save not in progress.

g) Restore not in progress.

7.18.4 Postconditions

a) The joined federate has the requested ownership information.

7.18.5 Exceptions

a) The object instance is not known.

b) The class attribute is not available at the known class of the object instance.

c) The federate is not a federation execution member.

d) Save in progress.

e) Restore in progress.

f) RTI internal error.

7.18.6 Reference state charts

a) None

8. Time management

8.1 Overview

The time management services and associated mechanisms provide a federation execution with the means to order the delivery of messages throughout the federation execution. Use of these mechanisms permits messages sent by different joined federates to be delivered in a consistent order to any joined federate in the federation execution that is to receive those messages.

Time in the system being modeled shall be represented in the federation as points along the HLA time axis. A joined federate can associate both itself and some of its activities with points on the HLA time axis. A joined federate's association with the HLA time axis is referred to as the logical time of the joined federate. An association of a joined federate's activities with the HLA time axis is denoted by assigning time stamps to the messages representing those activities. Time stamps and logical times are both represented by the same datatype and so can be compared, but they are referenced using different terms in order to help clarify whether a given time value applies to a joined federate or to a joined federate's activities.

Each joined federate may advance along the HLA time axis during the course of the execution. Such joined federate logical time advances may be constrained by the progress of other joined federates or they may be unconstrained.

Time management is concerned with the mechanisms for controlling the advancement of each joined federate along the HLA time axis. In general, logical time advances shall be coordinated with object management services so that information is delivered to joined federates in a causally correct and ordered fashion.

A joined federate that becomes time-regulating may associate some of its activities (e.g., updating instance attribute values and sending interactions) with points on the HLA time axis. It shall do so by assigning time stamps to the messages representing its activities; these time stamps correspond to the points on the HLA time axis with which the activities are associated. A joined federate that is time-constrained is interested in receiving the messages representing these activities (e.g., reflecting instance attribute values and receiving interactions) in a federation-wide time-stamped order. Use of the time management services allows this type of coordination among time-regulating and time-constrained joined federates in an execution. The coordination shall be achieved by various constraints on joined federate activities described in this chapter.

The activities of joined federates that are neither time-regulating nor time-constrained (the default state of all joined federates upon joining a federation execution) shall not be coordinated with other joined federates by the RTI, and such joined federates need not make use of any of the time management services. Such joined federates may, however, optionally place time stamps on the messages representing their activities (e.g., updating instance attribute values and sending interactions) to indicate when these actions occur during the federation execution. In this case, the RTI simply passes the specified time stamp, unchanged, to joined federate(s) receiving the message.

The representation of logical times and time stamps shall be specified to the RTI via the TRADT and should be documented in the time representation table of the federation's FOM. Federations requiring multiple time representations within a single execution (e.g., logical time and wallclock time values) shall realize this functionality explicitly within their TRADT definition, e.g., by using a "time type" field to differentiate between different types of time values.

8.1.1 Messages

The manner in which HLA services are coordinated with the HLA time axis shall be through the concept of messages as follows:

— Invocation of the *Update Attribute Values* service, *Send Interaction* service, *Send Interaction with Regions* service, or *Delete Object Instance* service by a joined federate shall be called sending a message. (Invocation of the *Update Attribute Values* service may actually result in the sending of multiple messages as each passel resulting from the invocation shall be considered a separate message.)

— Invocation of the *Reflect Attribute Values †* service, *Receive Interaction †* service, or *Remove Object Instance †* service at a joined federate shall be called receiving a message.

Messages sent by one joined federate typically result in one or more other joined federates receiving a corresponding message. The mapping from one sent message to one or more received messages shall follow the descriptions in 6.6, *Update Attribute Values*, 6.7, *Reflect Attribute Values †*, 6.8, *Send Interaction*, 6.9, *Receive Interaction †*, 6.10, *Delete Object Instance*, 6.11, *Remove Object Instance †*, and 9.12, *Send Interaction With Regions*. For example, a sent message representing an *Update Attribute Values* service invocation shall result only in received messages representing *Reflect Attribute Values †* service invocations at the appropriate joined federates depending on the normal ownership/subscription properties.

Each message, sent or received, shall be either a TSO message or an RO message (see 8.1.8). The order type of a message shall be determined by several factors:

— Preferred order type: The preferred order type of a message shall be the same as the preferred order type of the data contained in the message (instance attribute values or interactions). Each class attribute and interaction class shall be provided with a preferred order type in the FDD that indicates the order type (TSO or RO) that should be used when sending messages carrying values for instances of these classes. In the case of sent messages representing a *Delete Object Instance* service invocation, the preferred order type of the message shall be based on the preferred order type of the **HLAprivilegeToDeleteObject** attribute of the specified object instance. Joined federates may use the *Change Attribute Order Type* service to change the preferred order type of instance attributes; the preferred order type of class attributes may not be changed during a federation execution. Joined federates may use the *Change Interaction Order Type* service to change the preferred order type of interaction classes.

— Presence of a time stamp: Each of the services that corresponds to sending a message should include a time stamp argument if the preferred order type of any instance attribute or interaction class specified in the message is TSO, and the joined federate is time-regulating. This time stamp argument is always optional, but it is recommended in such cases as the occurrence of a preferred order type of TSO and a time-regulating joined federate indicate an intent for TSO messages to be sent.

— Joined federate's time status: Whether or not a joined federate is time-regulating shall determine whether or not a joined federate can send TSO messages. Similarly, whether or not a joined federate is time-constrained shall determine whether or not the joined federate can receive TSO messages.

— Sent message order type: The order type of a received message shall depend on the order type of the corresponding sent message.

These factors shall be considered together when determining if a given message is sent or received as a TSO message or as an RO message.

The order type of a sent message shall be determined by the preferred order type of the message at the sending joined federate, whether or not that joined federate is time-regulating, and whether or not a time stamp was supplied in the service invocation that sends the message. Table 1 illustrates how the order type of a sent message shall be determined.

Table 1—Order type of a sent message

Preferred order type?	Is the sending joined federate time-regulating?	Was a time stamp supplied?	Order type of sent message	Message retraction designator returned
RO	No	No	RO	No
RO	No	Yes	RO	No
RO	Yes	No	RO	No
RO	Yes	Yes	RO	No
TSO	No	No	RO	No
TSO	No	Yes	RO	No
TSO	Yes	No	RO	No
TSO	Yes	Yes	TSO	Yes

Table 2 illustrates how the order type of a received message shall be determined. The order type of a received message shall be determined by whether or not that joined federate is time-constrained and by the order type of the corresponding sent message. As shown in these two tables, a message retraction designator is provided to both the sender and the receiver if the sent message order type is TSO, regardless of the order type of the received message.

Table 2—Order type of a received message

Is the receiving joined federate time-constrained?	Order type of corresponding sent message?	Order type of received message?	Message retraction designator provided
No	RO	RO	No
No	TSO	RO	Yes
Yes	RO	RO	No
Yes	TSO	TSO	Yes

Because of the above rule defining the order type of a received message, the RTI will sometimes convert a sent TSO message to a received RO message at some receiving joined federates. The need for such conversions shall be considered on a per-joined federate basis. Thus, the received messages at different joined federates that correspond to the same sent message may be of different order types. Sent RO messages shall never be converted to received TSO messages.

Except when the FQR service is used, messages that are received as TSO messages shall be received only by a given joined federate in TSO, regardless of the joined federates from which the messages originate and regardless of the sequence in which the messages were sent. Thus, two TSO messages with different time stamps shall always be received by each joined federate in the same order. Multiple TSO messages having the same time stamp shall be received in an indeterminate order.

Messages that are received as RO messages shall be received in an arbitrary order. A time stamp shall be provided with the received message if one was specified when the message was sent.

8.1.2 Logical time

Each joined federate, upon joining an execution, shall be assigned a logical time. A joined federate's logical time shall initially be set to the initial time on the HLA time axis (the initial time of the TRADT). The logical time of a joined federate shall only advance; thus, a joined federate may request to advance only to a logical time that is greater than or equal to its current logical time. In order for a joined federate to advance its logical time, it shall request an advance explicitly. A joined federate shall request to advance its logical time only by invoking one of the following services:

— *Time Advance Request*
— *Time Advance Request Available*
— *Next Message Request*
— *Next Message Request Available*
— *Flush Queue Request*

The advance shall not occur until the RTI issues a grant via the *Time Advance Grant †* service. In general, at any instant during an execution, different joined federates may be at different logical times.

Joined federates also may become time-regulating and/or time-constrained. The logical times of joined federates that are time-regulating shall be used to constrain the advancement of the logical times of joined federates that are time-constrained.

8.1.3 Time-regulating joined federates

Only time-regulating joined federates may send TSO messages. A joined federate shall request to become time-regulating by invoking the *Enable Time Regulation* service. The RTI shall subsequently make the joined federate time-regulating by invoking the *Time Regulation Enabled †* service at that joined federate. A joined federate shall cease to be time-regulating whenever it invokes the *Disable Time Regulation* service.

Each time-regulating joined federate shall provide a lookahead when becoming time-regulating. Lookahead shall be a non-negative value that establishes the lowest value of time stamps that can be sent in TSO messages by the joined federate. Specifically, a time-regulating joined federate shall not send a TSO message that contains a time stamp less than its current logical time plus its current lookahead. Once established, changes to a joined federate's lookahead may only be requested using the *Modify Lookahead* service.

A time-regulating joined federate with a lookahead of zero shall be subject to an additional restriction. If such a joined federate has advanced its logical time by use of TAR or NMR, it shall not send TSO messages that contain time stamps less than or equal to its logical time (plus its lookahead, which is zero), rather than the usual less-than restriction. Subsequent use of a time advancement service that moves the joined federate's logical time forward lifts this additional restriction (and imposes the restriction of the subsequent advancement service). For example, if a zero lookahead joined federate were to invoke TAR (t_1) and to follow this with an invocation of TARA (t_1), that joined federate would still have the additional restriction. After the TARA is granted, it still may not send any TSO messages with a time stamp less than or equal to t_1 (the TAR restriction) because the second advance did not advance the joined federate's logical time.

A time-regulating joined federate need not send TSO messages in TSO, but all TSO messages that it sends shall be received by other joined federates in time-stamped order (if they are received as TSO messages) except when the FQR service is used.

8.1.4 Time-constrained joined federates

Only time-constrained joined federates can receive messages as TSO messages. A joined federate shall request to become time-constrained by invoking the *Enable Time Constrained* service. The RTI shall subsequently make the joined federate time-constrained by invoking the *Time Constrained Enabled †* service at that joined federate. A joined federate shall cease to be time-constrained whenever it invokes the *Disable Time Constrained* service.

In order to ensure that a time-constrained joined federate shall never receive a TSO message with a time stamp less than its logical time, a bound is placed on each time-constrained joined federate that limits how far it can advance its logical time. The bound ensures that a time-constrained joined federate cannot advance its logical time past a point at which TSO messages could still be sent by another joined federate. Should a time-constrained joined federate request to advance its logical time beyond its bound, the time advance shall not be granted until its bound has increased beyond the logical time to be granted.

This bound on advancement is expressed in terms of a value called the GALT. Each joined federate shall have a GALT that expresses the greatest logical time to which the RTI guarantees it can grant an advance without having to wait for other joined federates to advance. In the case of some time advance services (TAR and NMR) the joined federate shall be granted only to logical times that are less than its GALT. In the case of other time advance services (TARA, NMRA, and FQR), the joined federate shall be granted only to logical times that are less than or equal to its GALT.

A joined federate’s GALT is calculated by the RTI and is based on factors such as the logical time, lookahead, and requests to advance the logical time of time-regulating joined federates. If there are no other time-regulating joined federates in the execution, however, the joined federate's GALT is undefined, indicating that the bound is such that the joined federate may advance to any point without having to wait for another joined federate to advance.

While GALT is only used by the RTI to bound the advancement of time-constrained joined federates, nonconstrained joined federates also have a GALT. For a non-constrained joined federate, GALT expresses the bound that would apply to that joined federate if it were to become time-constrained.

Each joined federate shall have another value that builds on the idea of GALT called LITS. A joined federate’s LITS shall express the smallest time stamp that the joined federate could (but not necessarily will) receive in the future in a TSO message. A joined federate’s LITS is calculated by the RTI and is based on the joined federate’s GALT and any queued TSO messages that may later be received by the joined federate. If the joined federate’s GALT is undefined and there are no queued TSO messages that the joined federate could receive, the joined federate’s LITS shall also be undefined.

LITS is more useful for time-constrained joined federates, but it applies to all joined federates[7]. LITS is useful for joined federates wishing to know the time stamp of the next TSO message that they may have to process.

8.1.5 Advancing logical time

A joined federate is permitted to advance its logical time only by requesting a time advancement from the RTI (as indicated in 8.1.2). Its logical time shall not actually be advanced until the RTI responds with a *Time Advance Grant †* service invocation at that joined federate. The interval between these service invocations is defined to be the Time Advancing state; this is shown in the statechart in Figure 16.

[7]In the case of a nonconstrained joined federate, messages that may be received by that joined federate and that have a sent message order type of TSO may or may not be included in the calculation of the joined federate’s LITS.

Each service shall take a specific logical time t_R (time requested) as an argument, shall request slightly different coordination from the RTI, and is further elaborated on in the service descriptions and summarized in Table 3. The first column represents the form of time advancement that is requested. The second column describes what time stamps a joined federate cannot assign to messages it sends while in the Time Advancing state (i.e., after requesting an advance to t_R, but before the advance is granted). *ts* is short for the time stamp of a message. For example, the entry for TAR with zero lookahead reads "Can't send $ts \leq t_R$." This means that the joined federate shall not send any TSO messages that have a time stamp less than or equal to the time requested by the joined federate. The third column summarizes what messages the RTI shall guarantee to deliver to the advancing joined federate before granting the joined federate's request for time advancement; only messages described in this column shall be delivered. The fourth column is similar to the second, but it shows what constraints apply when the joined federate is in the Time Granted state (i.e., after an advance has been granted) and has it reached the Time Granted state by means of the specified form of time advancement.[8] Constraints in the fourth column are expressed in terms of the logical time (*LT*) of the joined federate. Lookahead in the second and fourth columns refers to the joined federate's current value of lookahead when in the respective states. Since a joined federate's lookahead can change while it is in the Time Granted state, the constraints expressed in this table change accordingly.

The *Time Advance Grant †* service shall be used to grant an advance regardless of which form of request was made to advance logical time. This service shall take a logical time as an argument, and this shall be the joined federate's new logical time (*LT*). The guarantee that the RTI makes about message delivery relative to the provided logical time shall depend on the type of request to advance time; the specific guarantees shall be provided in the service descriptions. Note that in some cases (i.e., when using NMR, NMRA, or FQR), the RTI may advance a joined federate to a logical time that is less than the logical time that the joined federate requested (t_R). In other cases (i.e., TAR and TARA), the RTI shall only advance a joined federate to the logical time that was requested.

The RTI shall grant an advance to logical time *LT* only when it can guarantee that all TSO messages with time stamps less than *LT* (or in some cases, less than or equal to *LT*) have been delivered to the joined federate. This guarantee enables the joined federate to model the behavior of the entities it represents up to logical time *LT* without concern for receiving new messages with time stamps less than *LT*. Note that in some cases, providing this guarantee shall require the RTI to wait for a significant period of wall-clock time to elapse before it can grant a time advancement to a time-constrained joined federate. However, in the case of joined federates that are not time-constrained (and, thus, cannot receive TSO messages), the guarantee is trivially true and the advance can be granted almost immediately.

The advancement of logical time by time-regulating joined federates is important because it acts as their guarantee not to send any TSO messages with time stamps less than some specified time. In general, when time-regulating joined federates move their logical times forward, time-constrained joined federates can move forward as well.

Joined federates that are not time-regulating need not advance their logical time, but they may do so. Such advancements shall have no effect on other joined federates' time advancement unless the advancing joined federate subsequently becomes time-regulating (at which point the advancing joined federate may begin to have an effect on the advancement of time-constrained joined federates).

[8]When time-regulation becomes enabled at a joined federate, that joined federate is treated as if it had advanced its logical time via the TARA service.

Table 3—Service descriptions

Time service	Constraint when in Time Advancing state (t_R is the requested logical time; lookahead refers to the actual lookahead the federate would have if it were granted to t_R)	Messages delivered before advance is granted	Constraint when in the Time Granted state (LT is the logical time of the joined federate; lookahead refers to the actual lookahead of the federate while in the Time Granted state)
TAR	Can't send $ts < t_R$ + lookahead	All queued RO messages All TSO messages with $ts \le t_R$	Can't send $ts < LT$ + lookahead
TAR (zero lookahead)	Can't send $ts \le t_R$	All queued RO messages All TSO messages with $ts \le t_R$	Can't send $ts \le LT$
TARA	Can't send $ts < t_R$ + lookahead	All queued RO messages All TSO messages with $ts < t_R$ All queued TSO messages with $ts = t_R$	Can't send $ts < LT$ + lookahead
NMR	Can't send $ts < t_R$ + lookahead	All queued RO messages TSO message with the smallest time stamp that will ever be received in a TSO message and for which $ts \le t_R$ All other TSO messages with the same ts	Can't send $ts < LT$ + lookahead
NMR (zero lookahead)	Can't send $ts \le t_R$	All queued RO messages TSO message with the smallest time stamp that will ever be received in a TSO message and for which $ts \le t_R$ All other TSO messages with the same ts	Can't send $ts \le LT$
NMRA	Can't send $ts < t_R$ + lookahead	All queued RO messages TSO message with the smallest time stamp that will ever be received in a TSO message and for which $ts \le t_R$ All other queued TSO messages with the same ts	Can't send $ts < LT$ + lookahead
FQR	Can't send $ts < t_R$ + lookahead	All queued RO messages All queued TSO messages	Can't send $ts < LT$ + lookahead

8.1.6 Putting it all together

The state chart shown in Figure 16 illustrates when a joined federate may become time-regulating and time-constrained, when time advances may be requested, how a joined federate enables or disables asynchronous message delivery, the effect these activities have on determining sent and received message order types, and when messages may be sent and received.

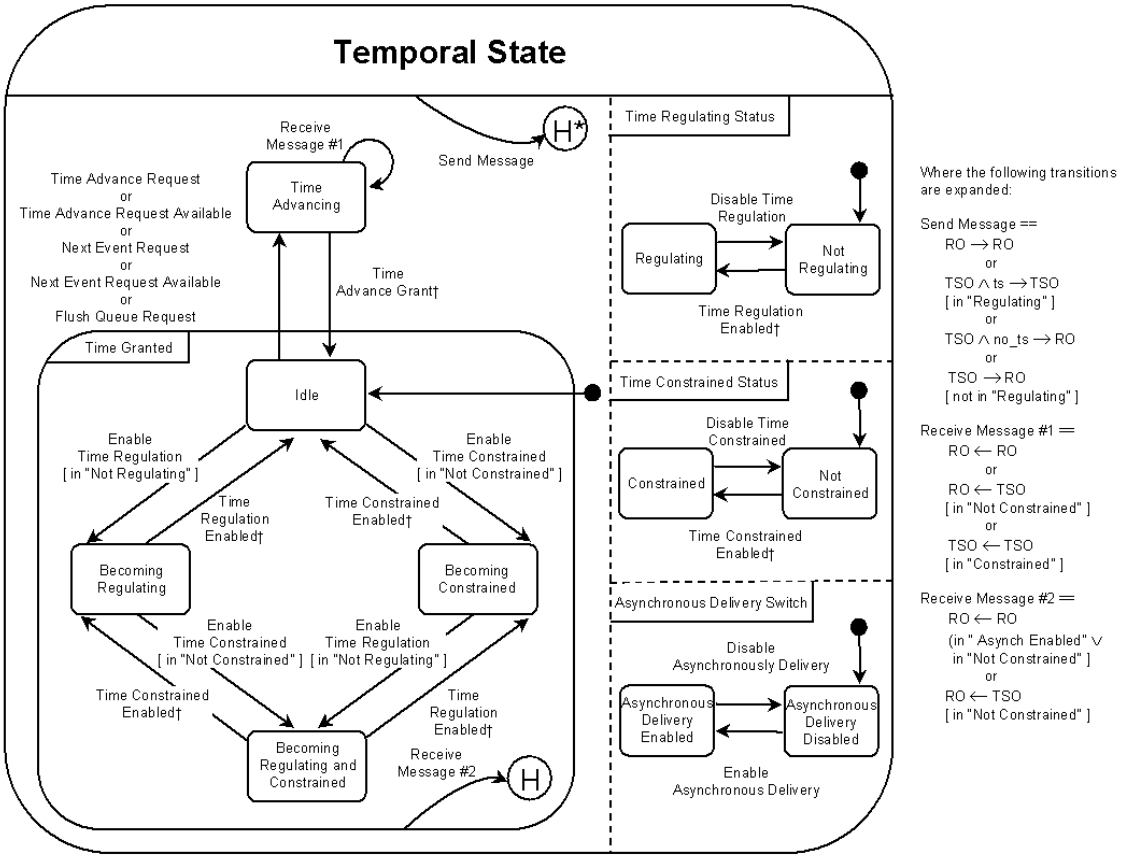

Figure 16—Temporal state

The transition labeled “Send Message” shall represent any service invocation that is called sending a message. As represented in the state chart, such a transition can occur in any state and shall result in the joined federate returning to whatever state it was in before the transition. The column to the right of the state chart elaborates on how the order type of the sent message is determined. Each part of the definition of “Send Message” shall be composed of a conversion rule (denoted as two terms separated by an arrow) and an optional Boolean guard (denoted in square braces, just as in state charts). The term to the left of the arrow in each conversion rule shall represent the preferred order type of the message and whether or not a time stamp was provided by the invoking joined federate. The term to the right of the arrow shall represent the order type of the sent message. The guard shall represent under what circumstances the conversion rule applies. So each part of the definition shall be read as follows: if the preferred order type of the message is as indicated to the left of the arrow, the usage of a time stamp is as described to the left of the arrow, and the Boolean guard (if present) is true, then the order type of the sent message is as indicated to the right of the arrow. The conversion rules provided in the statechart are the same as the results contained in Table 1 and Table 2.

The transitions labeled “Receive Message #1” and “Receive Message #2” shall be read similarly with one exception: The conversion rules shall be slightly different. The term to the left of the arrow shall represent the order type of the received message. The term to the right of the arrow shall represent the order type of the corresponding sent message.

A joined federate may send messages while in any state in this diagram. If the joined federate is time-regulating and sending a TSO message, the time stamp of that message shall be constrained as described in 8.1.3 with one exception: When a joined federate is in the Time Advancing state, the time stamp of sent TSO messages is constrained by the joined federate’s requested logical time (plus its lookahead) rather than by the joined federate’s current logical time (plus its lookahead). If the joined federate is granted to a logical time that is less than its requested logical time (i.e., the request must have used the NMR, NMRA, or FQR service), the constraints shall ease upon leaving the Time Advancing state. The constraints on advancing and nonadvancing joined federates are summarized in Table 3.

When joined federates are eligible to receive messages is dependent on several factors. If the joined federate is not time-constrained, it may receive messages while in any state (although only RO messages may be received). If the joined federate is time-constrained, it shall normally receive messages only when in the Time Advancing state. However, joined federates may enable asynchronous message delivery (via the *Enable Asynchronous Delivery* service), which permits them to receive RO messages (but not TSO messages) when not in the Time Advancing state.

Which RO messages will be received when a joined federate is eligible to receive RO messages shall depend only on which messages have been sent that will be received as RO messages by that joined federate. In general, if a joined federate is eligible to receive RO messages, it may receive all RO messages that it has not yet received.

Which TSO messages will be received when a joined federate is eligible to receive TSO messages shall depend on which TSO messages have been sent that will be received as TSO messages, what time stamps the messages have, and what form of time advancement was requested. Precisely which TSO messages will be received shall be defined in each of the different time advancement services.

Because messages are not always eligible for delivery, the RTI shall internally queue pending messages for each joined federate. The RTI shall queue all messages that the joined federate will receive as TSO or RO messages. When messages are finally delivered to the joined federate, they shall be removed from the queue.

8.1.7 Warnings on partial use of time management

When there are some joined federates in a federation execution that use time management services and other joined federates that do not, there are some complications that federation developers should be aware of. The complications stem from the fact that some joined federates are making use of TSO ordering of messages, while others are accepting RO ordering of these same messages. The two basic scenarios (which can, of course, occur independently or together in the same federation execution) are as follows:

a) A time-regulating joined federate sends TSO messages that are received by a nonconstrained joined federate in RO.

b) A nonregulating joined federate sends RO messages to a joined federate that expects to receive messages in TSO.

The former tends to be more of a problem because, among other reasons, the time-regulating joined federate need not send its messages in TSO. They may (for example) be received by the nonconstrained joined federate in the same order as sent, which is not equivalent to the order in which the sender meant them to be received.

The latter is not as much of a problem because the nonregulating joined federate will send its messages in the order it means for them to be received. The time-constrained joined federate will receive them in this order (barring factors like differing transportation types among the sent messages).

Another factor to consider is that joined federates do not need to be time-regulating with respect to all messages. A time-regulating joined federate can send some messages TSO and send other messages RO. This can also lead to complications if the decision on order type is not carefully considered.

Some specific examples of odd results when mixing the use of TSO and RO messages are presented below:

— A time-regulating joined federate invokes the *Delete Object Instance* service and provides a time stamp far in its future (much greater than its logical time). Subsequently, it sends several update messages, modeling attributes of the object instance in the interval prior to its deletion, providing a time stamp to each update that is less than the time stamp of the deletion. Nonconstrained joined federates may receive these messages in the order in which they were sent. This means that a nonconstrained joined federate may receive the delete message and never receive any of the update messages.

— A time-regulating joined federate invokes the *Delete Object Instance* service and provides a time stamp. This TSO message will be received as RO by any joined federates that are not time-constrained. This may result in nonconstrained joined federates receiving this message much sooner in wallclock time than time-constrained joined federates if the time stamp provided with the delete was greater than the logical time of most time-constrained joined federates. Even more problematic is if the nonconstrained joined federate later decides to become time-constrained. If it becomes time-constrained at a time less than the time stamp of the delete, it may now discover the object instance for which it just received a delete message. However, any messages regarding this object instance may have been dequeued for this joined federate when it received the delete message, and subsequent behavior may not be what is expected.

— A time-regulating joined federate that is sending both TSO and RO messages can in fact send both kinds of messages in a single *Update Attribute Values* service invocation. The message retraction designator that is returned on this service invocation, however, only applies to the resulting sent TSO messages, not to the whole service invocation. Consequently, any instance attribute values that resulted in sent RO messages shall not be retracted by invocations of *Retract* that include the returned message retraction designator.

8.1.8 Order types

The following order types shall be the only ones supported by the RTI

— TSO: (legal name shall be “TimeStamp”)

— RO: (legal name shall be “Receive”)

8.2 Enable time regulation

The *Enable Time Regulation* service shall enable time-regulation for the joined federate invoking the service, thereby enabling the joined federate to send TSO messages. The joined federate shall request that its lookahead be set to the specified value. The RTI shall indicate the logical time assigned to the joined federate through the *Time Regulation Enabled †* service. The logical time provided when time-regulation is enabled shall be the smallest possible logical time that is greater than or equal to the joined federate’s current logical time and for which all other constraints necessary to ensure TSO message delivery are satisfied. In general, this means that the logical time that the joined federate will be given (plus the requested lookahead) must be greater than or equal to the maximum logical time of all time-constrained joined federates (with the possibility of being of equal value depending on what form of time advancement was used by each time-constrained joined federate).

Upon the RTI's invocation of the corresponding *Time Regulation Enabled †* service, the invoking joined federate may begin sending TSO messages that have a time stamp greater than or equal to the joined federate's logical time plus the joined federate's lookahead. Zero lookahead joined federates are not subject to additional restrictions when time-regulation is first enabled.

Because the invocation of this service may require the RTI to advance the invoking joined federate's logical time, this service has an additional meaning for time-constrained joined federates. Since the advancing logical time for a time-constrained joined federate is synonymous with a guarantee that all TSO messages with time stamps less than the new logical time have been delivered, the invocation of this service shall be considered an implicit TARA service invocation. The subsequent invocation of *Time Regulation Enabled †* shall be considered an implicit *Time Advance Grant †* service invocation. Thus, if a time-constrained joined federate attempts to become time-regulating, it may receive RO and TSO messages between its invocation of *Enable Time Regulation* and the RTI's invocation of *Time Regulation Enabled †* at the joined federate. This special case is not illustrated in the state chart in Figure 16.

8.2.1 Supplied arguments

a) Lookahead

8.2.2 Returned arguments

a) None

8.2.3 Preconditions

a) The federation execution exists.

b) The federate is joined to that federation execution.

c) The joined federate is not in the time advancing state.

d) The joined federate does not have a request to enable time-regulation pending.

e) Time-regulation is not enabled in the joined federate.

f) Save not in progress.

g) Restore not in progress.

8.2.4 Postconditions

a) The RTI is informed of the joined federate's request to enable time-regulation (the request is now pending).

8.2.5 Exceptions

a) Time-regulation is already enabled.

b) Invalid lookahead.

c) Joined federate in time advancing state.

d) Request for time-regulation pending.

e) The federate is not a federation execution member.

f) Save in progress.

g) Restore in progress.

h) RTI internal error.

8.2.6 Reference state charts

a) Figure 16: Temporal state

8.3 Time Regulation Enabled †

Invocation of the *Time Regulation Enabled †* service shall indicate that a prior request to enable time-regulation has been honored. The value of this service's argument shall indicate that the logical time of the joined federate has been set to the specified value.

8.3.1 Supplied arguments

a) Current logical time of the joined federate

8.3.2 Returned arguments

a) None

8.3.3 Preconditions

a) The federation execution exists.

b) The federate is joined to that federation execution.

c) The joined federate has a request to enable time-regulation pending.

8.3.4 Postconditions

a) Time-regulation is enabled (the request is no longer pending), and the joined federate may now send TSO messages. The joined federate's logical time shall be set to the value specified as the argument to this service. The joined federate's actual lookahead shall be set to that specified in the corresponding *Enable Time Regulation* request.

b) If the joined federate is time-constrained, no additional TSO messages shall be delivered with time stamps less than the provided time.

8.3.5 Exceptions

a) Invalid logical time

b) No request to enable time-regulation pending

c) Federate internal error

8.3.6 Reference state charts

a) Figure 16: Temporal state

8.4 Disable Time Regulation

Invocation of the *Disable Time Regulation* service shall indicate that the joined federate is disabling time-regulation. Subsequent messages sent by the joined federate shall be sent automatically as RO messages.

8.4.1 Supplied arguments

a) None

8.4.2 Returned arguments

a) None

8.4.3 Preconditions

a) The federation execution exists.

b) The federate is joined to that federation execution.

c) Time-regulation is enabled at the joined federate.

d) Save not in progress.

e) Restore not in progress.

8.4.4 Postconditions

a) The joined federate may no longer send TSO messages.

8.4.5 Exceptions

a) Time-regulation was not enabled.

b) The federate is not a federation execution member.

c) Save in progress.

d) Restore in progress.

e) RTI internal error.

8.4.6 Reference state charts

a) Figure 16: Temporal state

8.5 Enable Time Constrained

The *Enable Time Constrained* service shall request that the joined federate invoking the service become time-constrained. The RTI shall indicate that the joined federate is time-constrained by invoking the *Time Constrained Enabled †* service.

8.5.1 Supplied arguments

a) None

8.5.2 Returned arguments

a) None

8.5.3 Preconditions

a) The federation execution exists.

b) The federate is joined to that federation execution.

c) The joined federate is not in the time advancing state.

d) The joined federate does not have a request to enable time-constrained pending.

e) The joined federate is not already time-constrained.

f) Save not in progress.

g) Restore not in progress.

8.5.4 Postconditions

a) The RTI is informed of the joined federate's request to become time-constrained (the request is now pending).

8.5.5 Exceptions

a) Time-constrained is already enabled.

b) Joined federate in time advancing state.

c) Request for time-constrained pending.

d) The federate is not a federation execution member.

e) Save in progress.

f) Restore in progress.

g) RTI internal error.

8.5.6 Reference state charts

a) Figure 16: Temporal state

8.6 Time Constrained Enabled †

Invocation of the *Time Constrained Enabled †* service shall indicate that a prior request to become time-constrained has been honored. The value of this service's argument shall indicate the new logical time of the joined federate. If the joined federate is time-regulating, the argument shall equal the joined federate's current logical time. The RTI shall provide the joined federate with the smallest possible logical time that is greater than or equal to the logical time of the joined federate and for which all other constraints necessary to ensure TSO message delivery are satisfied. In general, this means that the provided time must be less than or equal to the joined federate's GALT. If the joined federate's current logical time is greater than its GALT, then the joined federate shall not become time-constrained until the joined federate's GALT advances beyond its logical time. See 8.1.5 for more explanation on this point.

When a joined federate changes to be time-constrained, TSO messages stored in the RTI's internal queues that have time stamps greater than or equal to the joined federate's logical time shall be delivered in TSO.

TSO messages delivered to the joined federate before it becomes time-constrained, possibly including messages with time stamps greater than or equal to the joined federate's current logical time, shall be delivered as RO messages.

Joined federates that are time-constrained may receive messages only when in the Time Advancing state unless asynchronous message delivery is enabled (by use of the *Enable Asynchronous Delivery †* service). If asynchronous message delivery is enabled, the time-constrained joined federate may receive RO messages when not in the Time Advancing state, but TSO messages may still be received only when in the Time Advancing state.

8.6.1 Supplied arguments

a) Current logical time of the joined federate

8.6.2 Returned arguments

a) None

8.6.3 Preconditions

a) The federation execution exists.

b) The federate is joined to that federation execution.

c) The joined federate has a request to enable time-constrained pending.

8.6.4 Postconditions

a) The joined federate may now receive TSO messages (the request is no longer pending), and its logical time advances are constrained so that the joined federate's logical time shall never exceed the GALT computed by the RTI for the joined federate. The joined federate's logical time shall be set to the value specified as the argument to this service.

8.6.5 Exceptions

a) The logical time is invalid.

b) No request to enable time-constrained pending.

c) Federate internal error.

8.6.6 Reference state charts

a) Figure 16: Temporal state

8.7 Disable Time Constrained

Invocation of the *Disable Time Constrained* service shall indicate that the joined federate is no longer time-constrained. All enqueued and subsequent TSO messages shall be delivered to the joined federate as RO messages.

8.7.1 Supplied arguments

a) None

8.7.2 Returned arguments

a) None

8.7.3 Preconditions

a) The federation execution exists.

b) The federate is joined to that federation execution.

c) The joined federate is time-constrained.

d) Save not in progress.

e) Restore not in progress.

8.7.4 Postconditions

a) The joined federate is no longer time-constrained and can no longer receive TSO messages.

8.7.5 Exceptions

a) Time-constrained is not enabled.

b) The federate is not a federation execution member.

c) Save in progress.

d) Restore in progress.

e) RTI internal error.

8.7.6 Reference state charts

a) Figure 16: Temporal state

8.8 Time Advance Request

The TAR service shall request an advance of the joined federate's logical time and release zero or more messages for delivery to the joined federate.

Invocation of this service shall cause the following set of messages to be delivered to the joined federate:

— All messages queued in the RTI that the joined federate will receive as RO messages

— All messages that the joined federate will receive as TSO messages that have time stamps less than or equal to the specified logical time

After invoking TAR, the messages shall be passed to the joined federate by the RTI invoking the *Receive Interaction †*, *Reflect Attribute Values †*, and *Remove Object Instance †* services.

By invoking TAR with the specified logical time, the joined federate is guaranteeing that it will not generate a TSO message at any point in the future with a time stamp less than or equal to the specified logical time, even if the joined federate's lookahead is zero. Further, the joined federate may not generate any TSO messages in the future with time stamps less than the specified logical time plus the actual lookahead the joined federate would have if it were granted to the specified logical time.

A *Time Advance Grant †* shall complete this request and indicate to the joined federate that it has advanced to the specified logical time, and that no additional TSO messages will be delivered to the joined federate in the future with time stamps less than or equal to the logical time of the grant. For time-constrained joined federates, requests for which the specified logical time is less than GALT can be granted without waiting for other joined federates to advance.

8.8.1 Supplied arguments

a) Logical time

8.8.2 Returned arguments

a) None

8.8.3 Preconditions

a) The federation execution exists.

b) The federate is joined to that federation execution.

c) The specified logical time is greater than or equal to the joined federate's logical time.

d) The joined federate is not in the Time Advancing state.

e) The joined federate does not have a request to enable time-regulation pending.

f) The joined federate does not have a request to enable time-constrained pending.

g) Save not in progress.

h) Restore not in progress.

8.8.4 Postconditions

a) The joined federate may not send any TSO messages with time stamps less than the specified logical time plus its actual lookahead.

b) If the joined federate's actual lookahead is zero, it may not send any TSO messages with time stamps less than or equal to the specified logical time.

c) The RTI is informed of the joined federate's request to advance its logical time.

8.8.5 Exceptions

a) The logical time is invalid.

b) Logical time already passed.

c) Joined federate in time advancing state.

d) Request for time-regulation pending.

e) Request for time-constrained pending.

f) The federate is not a federation execution member.

g) Save in progress.

h) Restore in progress.

i) RTI internal error.

8.8.6 Reference state charts

a) Figure 16: Temporal state

8.9 Time Advance Request Available

The TARA service shall request an advance of the joined federate's logical time. It is similar to TAR to logical time *T,* except

— The RTI shall not guarantee delivery of all messages with time stamps equal to T when a *Time Advance Grant †* to logical time T is issued

— After the joined federate receives a *Time Advance Grant †* to logical time T, it can send additional messages with time stamps equal to T if the joined federate's actual lookahead is zero

Invocation of this service shall cause the following set of messages to be delivered to the joined federate:

— All messages queued in the RTI that the joined federate will receive as RO messages

— All messages that the joined federate will receive as TSO messages that have time stamps less than the specified logical time

— Any messages queued in the RTI that the joined federate will receive as TSO messages that have time stamps equal to the specified logical time

After invoking TARA, the messages shall be passed to the joined federate by the RTI invoking the *Receive Interaction †, Reflect Attribute Values †,* and *Remove Object Instance †* services.

By invoking TARA with the specified logical time, the joined federate is guaranteeing that it will not generate a TSO message at any point in the future with a time stamp less than the specified logical time plus the actual lookahead the joined federate would have if it were granted to the specified logical time.

A *Time Advance Grant †* shall complete this request and indicate to the joined federate that it has advanced to the specified logical time, and no additional TSO messages shall be delivered to the joined federate in the future with time stamps less than the logical time of the grant. Additional messages with time stamps equal to the logical time of the grant can arrive in the future. For time-constrained joined federates, requests for which the specified logical time is less than or equal to GALT can be granted without waiting for other joined federates to advance.

8.9.1 Supplied arguments

a) Logical time

8.9.2 Returned arguments

a) None

8.9.3 Preconditions

a) The federation execution exists.

b) The federate is joined to that federation execution.

c) The specified logical time is greater than or equal to the joined federate's logical time.

d) The joined federate is not in the time advancing state.

e) The joined federate does not have a request to enable time-regulation pending.

f) The joined federate does not have a request to enable time-constrained pending.

g) Save not in progress.

h) Restore not in progress.

8.9.4 Postconditions

a) The joined federate may not send any TSO messages with time stamps less than the specified logical time plus its actual lookahead.

b) The RTI is informed of the joined federate's request to advance logical time.

8.9.5 Exceptions

a) The logical time is invalid.

b) Logical time has already passed.

c) Joined federate in time advancing state.

d) Request for time-regulation pending.

e) Request for time-constrained pending.

f) The federate is not a federation execution member.

g) Save in progress.

h) Restore in progress.

i) RTI internal error.

8.9.6 Reference state charts

a) Figure 16: Temporal state

8.10 Next Message Request

The NMR service shall request the logical time of the joined federate to be advanced to the time stamp of the next TSO message that will be delivered to the joined federate, provided that message has a time stamp no greater than the logical time specified in the request.

Invocation of this service shall cause the following set of messages to be delivered to the joined federate:

— All messages queued in the RTI that the joined federate will receive as RO messages

— The smallest time-stamped message that will ever be received by the joined federate as a TSO message with a time stamp less than or equal to the specified logical time, and all other messages containing the same time stamp that the joined federate will receive as TSO messages

After invocation of NMR, the messages shall be passed to the joined federate by the RTI invoking the *Receive Interaction †, Reflect Attribute Values †,* and *Remove Object Instance †* services.

By invoking NMR with the specified logical time, the joined federate is guaranteeing that it will not generate a TSO message before the next *Time Advance Grant †* invocation with a time stamp less than or equal to the specified logical time (or less than the specified logical time plus the actual lookahead the joined federate would have if it were granted to the specified logical time if its lookahead is not zero).

If it does not receive any TSO messages before the *Time Advance Grant †* invocation, the joined federate shall guarantee that it will not generate a TSO message at any point in the future with a time stamp less than or equal to the specified logical time (or less than the specified logical time plus its actual lookahead if its lookahead is not zero).

If it does receive any TSO messages before the *Time Advance Grant †* invocation, the joined federate shall guarantee that it will not generate a TSO message at any point in the future with a time stamp less than or equal to the logical time of the grant (or less than the logical time of the grant plus its actual lookahead if its lookahead is not zero).

A *Time Advance Grant †* shall complete this request and indicate to the joined federate that it has advanced its logical time to the time stamp of the TSO messages that are delivered, if any, or to the specified logical time if no TSO messages were delivered. It shall also indicate that no TSO messages will be delivered to the joined federate in the future with time stamps less than or equal to the logical time of the grant. For time-constrained joined federates, requests for which either LITS or the specified logical time is less than GALT can be granted without waiting for other joined federates to advance.

8.10.1 Supplied arguments

a) Logical time

8.10.2 Returned arguments

a) None

8.10.3 Preconditions

a) The federation execution exists.

b) The federate is joined to that federation execution.

c) The specified logical time is greater than or equal to the joined federate's logical time.

d) The joined federate is not in the time advancing state.

e) The joined federate does not have a request to enable time-regulation pending.

f) The joined federate does not have a request to enable time-constrained pending.

g) Save not in progress.

h) Restore not in progress.

8.10.4 Postconditions

a) The joined federate may not send any TSO messages with time stamps less than the specified logical time plus its actual lookahead.

b) If the joined federate's lookahead is zero, it may not send any TSO messages with time stamps less than or equal to the specified logical time.

c) The RTI is informed of the joined federate's request to advance logical time.

8.10.5 Exceptions

a) The logical time is invalid.

b) Logical time has already passed.

c) Joined federate in time advancing state.

d) Request for time-regulation pending.

e) Request for time-constrained pending.

f) The federate is not a federation execution member.

g) Save in progress.

h) Restore in progress.

i) RTI internal error.

8.10.6 Reference state charts

a) Figure 16: Temporal state

8.11 Next Message Request Available

The *Next Message Request Available* service shall request the logical time of the joined federate to be advanced to the time stamp of the next TSO message that will be delivered to the joined federate, provided that message has a time stamp no greater than the logical time specified in the request. It is similar to *Next Message Request* except

— The RTI shall not guarantee delivery of all messages with time stamps equal to T when a *Time Advance Grant †* to logical time T is issued, and

— After the joined federate receives a *Time Advance Grant †* to logical time T, it can send additional messages with time stamps equal to T if the joined federate's lookahead is zero.

Invocation of this service shall cause the following set of messages to be delivered to the joined federate:

— All messages queued in the RTI that the joined federate will receive as RO messages

— The smallest time-stamped message that will ever be received by the joined federate as a TSO message with a time stamp less than or equal to the specified logical time, and any other messages queued in the RTI that the joined federate will receive as TSO messages and that have the same time stamp.

After invoking *Next Message Request Available*, the messages shall be passed to the joined federate by the RTI invoking the *Receive Interaction †, Reflect Attribute Values †,* and *Remove Object Instance †* services.

By invoking *Next Message Request Available* with the specified logical time, the joined federate is guaranteeing that it will not generate a TSO message before the next *Time Advance Grant †* invocation with a time stamp less than the specified logical time plus the actual lookahead the joined federate would have if it were granted to the specified logical time.

If it does not receive any TSO messages before the *Time Advance Grant †* invocation, the joined federate shall guarantee that it will not generate a TSO message at any point in the future with a time stamp less than the specified logical time plus its actual lookahead.

If it does receive any TSO messages before the *Time Advance Grant †* invocation, the joined federate shall guarantee that it will not generate a TSO message at any point in the future with a time stamp less than the logical time of the grant plus its actual lookahead.

A *Time Advance Grant †* shall complete this request and indicate to the joined federate that it has advanced its logical time to the time stamp of the TSO messages that are delivered, if any, or to the specified logical time if no TSO messages were delivered. Even if no TSO messages were delivered, it is still possible that TSO messages may be delivered in the future with time stamps equal to the logical time of the grant. A *Time Advance Grant †* shall also indicate that no TSO messages will be delivered to the joined federate in the future with time stamps less than the logical time of the grant. For time-constrained joined federates, requests for which LITS or the specified logical time is less than or equal to GALT can be granted without waiting for other joined federates to advance.

8.11.1 Supplied arguments

a) Logical time

8.11.2 Returned arguments

a) None

8.11.3 Preconditions

a) The federation execution exists.

b) The federate is joined to that federation execution.

c) The specified logical time is greater than or equal to the joined federate's logical time.

d) The joined federate is not in the Time Advancing state.

e) The joined federate does not have a request to enable time-regulation pending.

f) The joined federate does not have a request to enable time-constrained pending.

g) Save not in progress.

h) Restore not in progress.

8.11.4 Postconditions

a) The joined federate may not send TSO messages with time stamps less than the specified logical time plus its actual lookahead.

b) The RTI is informed of the joined federate's request to advance logical time.

8.11.5 Exceptions

a) The logical time is invalid.

b) Logical time has already passed.

c) Joined federate in time advancing state.

d) Request for time-regulation pending.

e) Request for time-constrained pending.

f) The federate is not a federation execution member.

g) Save in progress.

h) Restore in progress.

i) RTI internal error.

8.11.6 Reference state charts

a) Figure 16: Temporal state

8.12 Flush Queue Request

The FQR service shall request that all messages queued in the RTI that the joined federate will receive as TSO messages be delivered now. The RTI shall deliver all such messages as soon as possible, despite the fact that it may not be able to guarantee that no future messages containing smaller time stamps could arrive. The RTI shall advance the joined federate's logical time to the smallest of the following:

— The specified logical time

— The joined federate's GALT value

— The smallest time stamp of all TSO messages delivered by the RTI in response to this invocation of the FQR service

If the joined federate will not receive any additional TSO messages with time stamps less than the specified logical time, the joined federate shall be advanced to the specified logical time. Otherwise, the RTI shall advance the joined federate's logical time as far as possible (i.e., to the joined federate's GALT).

Invocation of this service shall cause the following set of messages to be delivered to the joined federate:

— All messages queued in the RTI that the joined federate will receive as RO messages

— All messages queued in the RTI that the joined federate will receive as TSO messages

After invoking *Flush Queue Request*, the messages shall be passed to the joined federate by the RTI invoking the *Receive Interaction †, Reflect Attribute Values †,* and *Remove Object Instance †* services.

By invoking *Flush Queue Request* with the specified logical time, the joined federate is guaranteeing that it will not generate a TSO message before the next *Time Advance Grant †* invocation with a time stamp less than the specified logical time plus the actual lookahead the joined federate would have if it were granted to the specified logical time.

After the *Time Advance Grant †* invocation, the joined federate shall guarantee that it shall not generate a TSO message at any point in the future with a time stamp less than the logical time of the grant plus its actual lookahead.

A *Time Advance Grant †* shall complete this request and indicate to the joined federate that it has advanced to the logical time of the grant, and no additional TSO messages shall be delivered to the joined federate in the future with time stamps less than the logical time of the grant. An FQR can always be granted without waiting for other joined federates to advance.

Since FQR results in the delivery of all queued TSO messages, not just those the RTI can guarantee are safe to deliver, there are two consequences the receiving joined federate must be prepared to handle. First, messages may later be received that have time stamps that are smaller than those of some messages received now, thus violating time-stamped ordering. Second, some messages delivered may still be eligible for retraction by their originators, and so the receiving joined federate must be capable of dealing with *Request Retraction †* service invocations for these messages.

8.12.1 Supplied arguments

a) Logical time

8.12.2 Returned arguments

a) None

8.12.3 Preconditions

a) The federation execution exists.

b) The federate is joined to that federation execution.

c) The specified logical time is greater than or equal to the joined federate's logical time.

d) The joined federate is not in the Time Advancing state.

e) The joined federate does not have a request to enable time-regulation pending.

f) The joined federate does not have a request to enable time-constrained pending.

g) Save not in progress.

h) Restore not in progress.

8.12.4 Postconditions

a) The joined federate may not send any TSO messages with time stamps less than the specified logical time plus its actual lookahead.

b) The RTI is informed of the joined federate's request to advance logical time.

8.12.5 Exceptions

a) The logical time is invalid.

b) Logical time has already passed.

c) Joined federate in time advancing state.

d) Request for time-regulation pending.

e) Request for time-constrained pending.

f) The federate is not a federation execution member.

g) Save in progress.

h) Restore in progress.

i) RTI internal error.

8.12.6 Reference state charts

a) Figure 16: Temporal state

8.13 Time Advance Grant †

Invocation of the *Time Advance Grant †* service shall indicate that a prior request to advance the joined federate's logical time has been honored. The argument of this service shall indicate that the logical time for the joined federate has been advanced to this value.

If the grant is issued in response to invocation of NMR or TAR, the RTI shall guarantee that no additional TSO messages shall be delivered in the future with time stamps less than or equal to this value.

If the grant is in response to an invocation of TARA, NMRA, or FQR, the RTI shall guarantee that no additional TSO messages shall be delivered in the future with time stamps less than the value of the grant.

8.13.1 Supplied arguments

a) Logical time

8.13.2 Returned arguments

a) None

8.13.3 Preconditions

a) The federation execution exists.

b) The federate is joined to that federation execution.

c) The joined federate is in the time advancing state.

8.13.4 Postconditions

a) If the joined federate has a change to its lookahead pending, its new actual lookahead shall be equal to the maximum of its requested lookahead and its old actual lookahead less the amount of logical time advanced (the joined federate's old logical time less the supplied logical time).

b) The joined federate may not send TSO messages with time stamps less than the supplied logical time plus its actual lookahead.

c) If this service is invoked in response to a TAR or NMR the joined federate's actual lookahead is zero, and the joined federate may not send TSO messages with time stamps less than or equal to the supplied logical time.

d) If this service is invoked in response to a TAR or NMR, no additional TSO messages shall be delivered with time stamps less than or equal to the supplied logical time.

e) If this service is invoked in response to a TARA, NMRA, or FQR, no additional TSO messages shall be delivered with time stamps less than the supplied logical time.

8.13.5 Exceptions

a) The logical time is invalid

b) Joined federate is not in time advancing state

c) Federate internal error

8.13.6 Reference state charts

a) Figure 16: Temporal state

8.14 Enable Asynchronous Delivery

Invocations of the *Enable Asynchronous Delivery* service shall instruct the RTI to deliver received RO messages to the invoking joined federate when it is in either the Time Advancing or Time Granted state.

8.14.1 Supplied arguments

a) None

8.14.2 Returned arguments

a) None

8.14.3 Preconditions

a) The federation execution exists.

b) The federate is joined to that federation execution.

c) Asynchronous delivery is disabled at the joined federate.

d) Save not in progress.

e) Restore not in progress.

8.14.4 Postconditions

a) Asynchronous delivery is enabled at the joined federate.

8.14.5 Exceptions

a) Asynchronous delivery is already enabled.

b) The federate is not a federation execution member.

c) Save in progress.

d) Restore in progress.

e) RTI internal error.

8.14.6 Reference state charts

a) Figure 16: Temporal state

8.15 Disable Asynchronous Delivery

Invocations of the *Disable Asynchronous Delivery* service shall, for a time-constrained federate, instruct the RTI to deliver RO messages to the invoking federate only when it is in the Time Advancing state.

8.15.1 Supplied arguments

a) None

8.15.2 Returned arguments

a) None

8.15.3 Preconditions

a) The federation execution exists.

b) The federate is joined to that federation execution.

c) Asynchronous delivery is enabled at the joined federate.

d) Save not in progress.

e) Restore not in progress.

8.15.4 Postconditions

a) Asynchronous delivery is disabled at the joined federate.

8.15.5 Exceptions

a) Asynchronous delivery is already disabled.

b) The federate is not a federation execution member.

c) Save in progress.

d) Restore in progress.

e) RTI internal error.

8.15.6 Reference state charts

a) Figure 16: Temporal state

8.16 Query GALT

The *Query GALT* service shall request the invoking joined federate's current GALT. The first returned argument shall indicate whether or not GALT is defined for the joined federate. If the argument indicates that GALT is defined for the joined federate, then the optional returned argument shall be supplied to indicate the joined federate's current GALT.

8.16.1 Supplied arguments

a) None

8.16.2 Returned arguments

a) GALT definition indicator

b) Optional current value of invoking joined federate's GALT

8.16.3 Preconditions

a) The federation execution exists.

b) The federate is joined to that federation execution.

c) Save not in progress.

d) Restore not in progress.

8.16.4 Postconditions

a) The joined federate receives the current value of its GALT, if defined.

8.16.5 Exceptions

a) The federate is not a federation execution member.

b) Save in progress.

c) Restore in progress.

d) RTI internal error.

8.16.6 Reference state charts

a) None

8.17 Query Logical Time

The *Query Logical Time* service shall request the current logical time of the invoking joined federate.

8.17.1 Supplied arguments

b) None

8.17.2 Returned arguments

a) The invoking joined federate’s current logical time

8.17.3 Preconditions

a) The federation execution exists.

b) The federate is joined to that federation execution.

c) Save not in progress.

d) Restore not in progress.

8.17.4 Postconditions

a) The joined federate receives the current value of its logical time.

8.17.5 Exceptions

a) The federate is not a federation execution member.

b) Save in progress.

c) Restore in progress.

d) RTI internal error.

8.17.6 Reference state charts

a) None

8.18 Query LITS

The *Query LITS* service shall request the invoking joined federate's current LITS. The first returned argument shall indicate whether or not LITS is defined for the joined federate. If the argument indicates that LITS is defined for the joined federate, then the optional returned argument shall be supplied to indicate the joined federate's current LITS.

8.18.1 Supplied arguments

a) None

8.18.2 Returned arguments

a) LITS definition indicator

b) Optional current value of invoking joined federate's LITS

8.18.3 Preconditions

a) The federation execution exists.

b) The federate is joined to that federation execution.

c) Save not in progress.

d) Restore not in progress.

8.18.4 Postconditions

a) The joined federate receives its LITS, if defined.

8.18.5 Exceptions

a) The federate is not a federation execution member.

b) Save in progress.

c) Restore in progress.

d) RTI internal error.

8.18.6 Reference state charts

a) None

8.19 Modify Lookahead

The *Modify Lookahead* service shall request a change to the joined federate's lookahead. A joined federate may not request a change to its lookahead while it is in the Time Advancing state. The specified lookahead shall be greater than or equal to zero. If the requested value is greater than or equal to the joined federate's actual lookahead, the change shall take effect immediately and the requested lookahead shall become the actual lookahead. If the requested value is less than the joined federate's actual lookahead, the change shall take effect gradually as the joined federate advances its logical time and the actual lookahead is initially unchanged. Specifically, the joined federate's actual lookahead shall decrease by T units each time logical time advances T units until the requested lookahead is reached.

8.19.1 Supplied arguments

a) Requested lookahead

8.19.2 Returned arguments

a) None

8.19.3 Preconditions

a) The federation execution exists.
b) The federate is joined to that federation execution.
c) Time-regulation is enabled in the joined federate.
d) The joined federate is not in the time advancing state.
e) Save not in progress.
f) Restore not in progress.

8.19.4 Postconditions

a) If the requested lookahead is greater than or equal to the joined federate's actual lookahead, the joined federate's actual lookahead shall be set to the requested value.
b) If the requested lookahead is less than the joined federate's actual lookahead, the RTI shall be informed of the joined federate's requested lookahead value.

8.19.5 Exceptions

a) Time-regulation was not enabled.
b) Invalid lookahead.
c) Joined federate in Time Advancing state.
d) The federate is not a federation execution member.
e) Save in progress.
f) Restore in progress.
g) RTI internal error.

8.19.6 Reference state charts

a) None

8.20 Query Lookahead

The *Query Lookahead* service shall query the RTI for the joined federate's current actual lookahead. The current value of actual lookahead may differ temporarily from the requested lookahead given in the *Modify Lookahead* service if the joined federate is attempting to reduce its actual lookahead.

8.20.1 Supplied arguments

a) None

8.20.2 Returned arguments

a) The invoking joined federate's current actual lookahead

8.20.3 Preconditions

a) The federation execution exists.

b) The federate is joined to that federation execution.

c) Time-regulation is enabled in the joined federate.

d) Save not in progress.

e) Restore not in progress.

8.20.4 Postconditions

a) The joined federate receives the current value of its actual lookahead.

8.20.5 Exceptions

a) Time-regulation was not enabled.

b) The federate is not a federation execution member.

c) Save in progress.

d) Restore in progress.

e) RTI internal error.

8.20.6 Reference state charts

a) None

8.21 Retract

The *Retract* service shall be used by a joined federate to notify the federation execution that a message previously sent by the joined federate is to be retracted. The *Update Attribute Values*, *Send Interaction*, *Send Interaction with Regions*, and *Delete Object Instance* services shall return a message retraction designator that is used to specify the message that is to be retracted. Retracting a message shall cause the invocation of the *Request Retraction †* service in all joined federates that received the original message.

Retracting a *Delete Object Instance* message shall result in the reconstitution of the corresponding object instance. This shall cause the ownership reassumption of the attributes of the affected object instance by the joined federates that owned them when the *Delete Object Instance* service was invoked.

A message may only be retracted if

a) The retracting joined federate sent the message

b) The message had a sent order type of TSO (i.e., a message retraction designator was returned by the RTI)

c) The joined federate is time-regulating, and either

1) The joined federate is in the Time Granted state and the message associated with the specified retraction designator contained a time stamp that is larger than the joined federate's current logical time plus its actual lookahead, or

2) The joined federate is in the Time Advancing state and the message associated with the specified retraction designator contained a time stamp that is larger than the logical time specified in the joined federate's most recent advance request plus the joined federate's actual lookahead

8.21.1 Supplied arguments

a) Message retraction designator

8.21.2 Returned arguments

a) None

8.21.3 Preconditions

a) The federation execution exists.

b) The federate is joined to that federation execution.

c) The joined federate is time-regulating.

d) The joined federate has issued *Update Attribute Values*, *Send Interaction*, *Send Interaction with Regions*, or *Delete Object Instance* service invocations previously and obtained the message retraction designator.

e) If the joined federate is in the Time Granted state, the message associated with the specified retraction designator contained a time stamp that is larger than the joined federate's current logical time plus its actual lookahead.

f) If the joined federate is in the Time Advancing state, the message associated with the specified retraction designator contained a time stamp that is larger than the logical time specified in the joined federate's most recent advance request plus its actual lookahead.

g) Save not in progress.

h) Restore not in progress.

8.21.4 Postconditions

a) The RTI is informed that the joined federate requests to retract the specified message.

8.21.5 Exceptions

a) The message retraction designator is invalid.

b) The federate is not time-regulating.

c) The retraction designator is associated with a message that can no longer be retracted.

d) The federate is not a federation execution member.

e) Save in progress.

f) Restore in progress.

g) RTI internal error.

8.21.6 Reference state charts

a) None

8.22 Request Retraction †

If the RTI receives a legal *Retract* service invocation for a message that has already been delivered to a joined federate, the *Request Retraction †* service shall be invoked on that joined federate. If the message in question is queued in the RTI and has not been delivered to the joined federate, the message shall be dequeued from the RTI and will never be delivered to the joined federate.

Time-constrained joined federates that do not use the FQR service are not subject to invocation of this service on any received TSO message because they will never receive TSO messages that are eligible for retraction. Nonconstrained joined federates, however, must be prepared to deal with invocations of this service because any message received that was sent as a TSO message may be eligible for retraction.

Since the joined federate must save the message retraction designator in order to perform a retraction, the RTI is not required to ensure the message was actually delivered to the federate. The RTI is only required to ensure that a message is not delivered to a joined federate after a retraction for that message has been requested.

8.22.1 Supplied arguments

a) Message retraction designator

8.22.2 Returned arguments

a) None

8.22.3 Preconditions

a) The federation execution exists.

b) The federate is joined to that federation execution.

8.22.4 Postconditions

a) The joined federate has been directed to retract the specified message.

8.22.5 Exceptions

a) Federate internal error

8.22.6 Reference state charts

a) None

8.23 Change Attribute Order Type

The preferred order type for each attribute of an object instance shall be initialized from the object class description in the FDD. A joined federate may choose to change the preferred order type during execution. Invoking the *Change Attribute Order Type* service shall change the order type for all future *Update Attribute*

Values service invocations for the specified instance attributes. When the ownership of an instance attribute is changed, the preferred order type shall revert to that defined in the FDD.

8.23.1 Supplied arguments

a) Object instance designator

b) Set of attribute designators

c) Order type

8.23.2 Returned arguments

a) None

8.23.3 Preconditions

a) The federation execution exists.

b) The federate is joined to that federation execution.

c) An object instance with the specified designator exists.

d) The joined federate knows about the object instance with the specified designator.

e) The specified class attributes are available attributes of the object instance's known class.

f) The attributes are defined in the FDD.

g) The joined federate owns the instance attributes.

h) The order type is valid (see 8.1.8).

i) Save not in progress.

j) Restore not in progress.

8.23.4 Postconditions

a) The order type is changed for the specified instance attributes.

8.23.5 Exceptions

a) The object instance is not known.

b) The specified class attributes are not available attributes of the known object class.

c) The joined federate does not own the specified instance attributes.

d) The order type is invalid.

e) The federate is not a federation execution member.

f) Save in progress.

g) Restore in progress.

h) RTI internal error.

8.23.6 Reference state charts

a) None

8.24 Change Interaction Order Type

The preferred order type of each interaction shall be initialized from the interaction class description in the FDD. A joined federate may choose to change the preferred order type during execution. Invoking the *Change Interaction Order Type* service shall change the order type for all future *Send Interaction* and *Send Interaction with Regions* service invocations for the specified interaction class for the invoking joined federate only.

8.24.1 Supplied arguments

a) Interaction class designator

b) Order type

8.24.2 Returned arguments

a) None

8.24.3 Preconditions

a) The federation execution exists.

b) The federate is joined to that federation execution.

c) The interaction class is defined in the FDD.

d) The joined federate is publishing the interaction class.

e) The order type is valid (see 8.1.8).

f) Save not in progress.

g) Restore not in progress.

8.24.4 Postconditions

a) The preferred order type is changed for the specified interaction class.

8.24.5 Exceptions

a) The interaction class is not defined in the FDD.

b) The joined federate is not publishing the interaction class.

c) The order type is invalid.

d) The federate is not a federation execution member.

e) Save in progress.

f) Restore in progress.

g) RTI internal error.

8.24.6 Reference state charts

a) None

9. Data distribution management

9.1 Overview

DDM services may be used by joined federates to reduce both the transmission and the reception of irrelevant data. Whereas DM services provide information on data relevance at the class attribute level, DDM services add the capability to further refine the data requirements at the instance attribute level. Similarly, whereas DM services provide information on data relevance at the interaction class level, DDM services add the capability to further refine the data requirements at the specific interaction level. Producers and consumers of data communicated between joined federates may use the DDM services to bound the relevance of such communicated data, enabling the RTI to recognize the irrelevant data and prevent its delivery to consumers. The relevance of communicating interactions or instance attribute updates is expressed by giving a bounding for relevant communication in a space of user-defined dimensions. Both consumers and producers specify the upper and lower bounds for the relevant portion of that space, and the overlap of producers' and consumers' region sets is used to bound relevant communication.

9.1.1 Definitions for DDM

The DDM services shall be based on the following concepts and terms:

a) A **dimension** shall be a named interval of non-negative integers. The interval shall be defined by an ordered pair of values. The first value is 0 (zero) for every dimension. The second value, which can vary for each dimension, is that dimension's upper bound as specified in the FDD.

b) A **range** shall be a continuous semi-open interval on a dimension. The interval is defined by an ordered pair of values. The first component of the pair shall be called **range lower bound**, and the second component shall be called **range upper bound**. The range upper bound shall be strictly greater than the range lower bound. The minimum possible difference between a range lower bound and a range upper bound shall be 1 (one).

c) A **region specification** shall be a set of ranges. The dimensions contained in a region specification shall be the dimensions of the ranges that are included in the region specification. A region specification shall contain at most one range for any given dimension. Each range in a region specification shall be defined in terms of the bounds [0, the corresponding dimension's upper bound).

d) A **region realization** shall be a region specification that is associated with an instance attribute for update, with a sent interaction, or with a class attribute or interaction class for subscription. The generic term **region** may be used in circumstances where a region specification, a region realization, or both apply.

e) The RTI shall provide the **default region**, which is defined as having the range [0, the range's dimension's upper bound) for each dimension found in the FDD. The default region shall have all dimensions found in the FDD, regardless of the class attribute or interaction class with which it is associated. The default region realization shall not have a region specification. There shall be no way for a joined federate to refer to the default region. If a joined federate creates a region that has as its dimensions all dimensions found in the FDD and has the range [0, the range's dimension's upper bound) for each dimension, this region shall be equivalent to the default region, but it is not the default region.

f) A collection of attribute designator set and region designator set pairs shall be interpreted as a set of (attribute designator, region designator) pairs. The interpreted form can be viewed as if constructed in the following manner:

    1) The interpreted form begins as an empty set of (attribute designator, region designator) pairs.

2) For each pair (of sets) in the attribute designator set and region designator set pair collection, create the cross product of the attribute designator set with its region designator set, resulting in a collection of (attribute designator, region designator) pairs. The union of all such (attribute designator, region designator) pairs shall be the interpreted form.

9.1.2 Relating region specifications and region realizations

The following relationships, established in the FDD, shall pertain to dimensions:

a) The available dimensions of a class attribute shall be the dimensions associated with that class attribute in the FDD. The available dimensions of an instance attribute shall be the available dimensions of the corresponding class attribute.

b) The available dimensions of an interaction class shall be the dimensions associated with that class in the FDD. The available dimensions of a sent interaction shall be the available dimensions of the interaction class specified in the *Send Interaction With Regions* service invocation.

c) DDM services shall not be invoked for class attributes, instance attributes, interaction classes, or sent interactions that do not have available dimensions.

d) Each dimension in the FDD shall have either a default range specified in terms of [0, the dimension's upper bound) or shall have an **excluded indicator** specified in the Value When Unspecified Field of the FDD Dimension Table. Where a default range is specified for a dimension in the FDD and the dimension is not mentioned in a region specification, it is implicitly added, with the specified default range, when creating a region realization.

9.1.3 Using regions

9.1.3.1 Region relationships

The following relationships, established through DDM services, shall pertain to regions:

a) A region specification may be created using the *Create Region* service. Such a region specification may be deleted using the *Delete Region* service. Invoking the *Commit Region Modifications* service for a region specification shall notify the RTI about modifications to that region specification.

b) The specified dimensions of the region specification shall be the dimensions that are explicitly provided when the region specification is created or modified.

c) The unspecified dimensions of a region realization shall be the available dimensions of the class attribute, instance attribute, interaction class, or sent interaction with which the region realization is associated, less the specified dimensions of the region specification from which the region realization is **derived**.

d) Region realizations can be created from region specifications via the *Register Object Instance with Regions*, *Associate Regions for Updates*, *Subscribe Object Class Attributes With Regions*, *Subscribe Interaction Class with Regions*, *Send Interaction With Regions*, *Request Attribute Value Update With Regions*, or *Commit Region Modifications* services.

e) When a region realization is created by associating a region specification with an instance attribute, a class attribute, an interaction class, or a sent interaction, additions may be made to the region realization by the RTI to account for default dimension values indicated in the FDD as follows:

1) All specified dimensions of the region specification shall be included in the region realization, and the range associated with each specified dimension in the region realization shall be the range found in the region specification for the same dimension.

2) For each unspecified dimension of the region realization, if the FDD indicates a default range other than the excluded indicator, this default range shall be added to the region realization.

3) When a federate modifies a region specification from which region realizations have been derived, the RTI shall fill in unspecified dimensions in this same manner.

f) A region specification may contain an empty set of ranges. The region realization derived from such a region specification shall contain only those dimensions for which the FDD defines a default range and that are available dimensions of the instance attribute, class attribute, interaction class, or sent interaction with which the region specification is being associated. This may result in a region realization that contains no dimensions if all available dimensions have the excluded indicator specified in the FDD; such a region realization shall not overlap with any other region realization, even the default region.

9.1.3.2 Instance attribute relationships

The following relationships, established through DDM services, shall pertain to object classes, class attributes, object instances, and instance attributes:

a) A region realization other than the default region shall be used for update of an instance attribute if the joined federate owns the instance attribute and has used the instance attribute and region specification from which the region realization was derived as arguments either in the *Register Object Instance With Regions* service or in the *Associate Regions For Updates* service.

The default region shall be used for update of an instance attribute if and only if the corresponding class attribute has available dimensions and there is no other region realization used for update for that instance attribute. This may occur if the *Register Object Instance* service is used to register an instance of an object class that has attributes with available dimensions, or if a region association for update is removed via *Unassociate Regions for Updates*.

Subsequently invoking the *Unassociate Regions For Updates* service for the same (object instance, region, attribute) triple shall cause that region not to be used for update of that instance attribute. Invoking the *Associate Regions For Updates* service or the *Unassociate Regions For Updates* service with an empty set of regions shall make no changes to the set of associations of the instance attribute.

Any region specifications used to generate region realizations associated with an instance attribute for update shall only contain dimensions that are a subset of the available dimensions of that instance attribute. The resulting region realizations shall have the same property.

A joined federate shall use a region for update of an instance attribute to assert properties of that instance attribute. These properties shall be used to limit reflection of the instance attributes when the joined federate invokes the *Update Attribute Values* service. If a region other than the default region is used for update of a particular instance attribute by a joined federate and the joined federate loses ownership of that instance attribute, that region shall no longer be used for update of that instance attribute.

A joined federate may create and modify update region associations with an instance attribute that is owned by another joined federate or that is unowned. These associations shall not take effect until the joined federate making the associations has acquired ownership of the specified instance attribute. These associations shall take effect immediately upon acquiring ownership of the specified instance attributes.

In all cases, if a joined federate divests ownership of an instance attribute with which an update region is associated, that association shall be lost, as if the *Unassociate Regions For Updates* service had been invoked for that (object instance, region, attribute) triple. Update region associations that are lost will not come into effect if the joined federate subsequently acquires ownership of the instance attribute unless the joined federate explicitly recreates the association.

b) A region realization other than the default region shall be used for subscription of a class attribute if the joined federate has used the class attribute and a given object class and the region specification from which the region realization was derived as arguments in the *Subscribe Object Class Attributes With Regions* service. Subsequently invoking the *Unsubscribe Object Class Attributes With Regions* service for the same (object class, region, attribute) triple shall cause the region not to be used for subscription of that class attribute. Invoking the *Subscribe Object Class Attributes With Regions* service or the *Unsubscribe Object Class Attributes With Regions* service with an empty set of regions shall make no changes to the set of subscriptions with region of the class attribute.

Any region specifications used to generate region realizations associated with a class attribute for subscription shall only contain dimensions that are a subset of the available dimensions of that class attribute. The resulting region realizations shall have the same property.

The default region shall be used for subscription of a class attribute if and only if that class attribute has available dimensions, there is no other region realization used for subscription of that class attribute, and the joined federate is subscribed to that class attribute. Subsequently invoking the *Unsubscribe Object Class Attributes* service for the same object class and attribute shall cause the default region not to be used for subscription of that class attribute.

A joined federate shall use a region for subscription of a class attribute to specify requirements for reflecting values of that class attribute's corresponding instance attributes.

9.1.3.3 Interaction relationships

The following relationships, established through DDM services, shall pertain to interaction classes, parameters, and interactions:

a) A region realization other than the default region shall be used for sending an interaction if the joined federate has used the interaction and a region specification from which the region realization may be derived, as arguments to an invocation of the *Send Interaction With Regions* service.

Any region associated with a sent interaction shall only contain dimensions that are a subset of the available dimensions of the sent interaction. The default region shall be used for sending an interaction of such a class during an invocation of the *Send Interaction* service.

A joined federate shall use a region for sending an interaction to assert properties of that interaction when the *Send Interaction With Regions* service is invoked.

b) A region realization other than the default region shall be used for subscription of an interaction class if the joined federate has used the interaction class and the region specification from which the region realization was derived as arguments in the *Subscribe Interaction Class With Regions* service. Subsequently invoking the *Unsubscribe Interaction Class With Regions* service for the same (interaction class, region) pair shall cause the region not to be used for subscription of that interaction class. Invoking the *Subscribe Interaction Class With Regions* service or the *Unsubscribe Interaction Class With Regions* service with an empty set of regions shall make no changes to the set of subscriptions with region of the interaction class.

Any region associated with an interaction class for subscription shall only contain dimensions that are a subset of the available dimensions of that interaction class.

The default region shall be used for subscription of that interaction class if and only if that interaction class has available dimensions, there is no other region realization used for subscription of that interaction class, and the joined federate is subscribed to that interaction class. Subsequently invoking the *Unsubscribe Interaction Class* service for the same interaction class shall cause the default region not to be used for subscription of that interaction class.

A joined federate shall use a region for subscription of an interaction class to establish requirements for receiving interactions of that class.

A set of region realizations used for update of instance attributes or for sending interactions shall be called an **update region set**.

A set of region realizations used for subscription of either class attributes or interaction classes shall be called a **subscription region set**.

9.1.4 Calculating region overlaps

An update region set and a subscription region set shall overlap if and only if a region exists in each set such that the two regions overlap. Two regions overlap if and only if all ranges of dimensions that are contained in both regions overlap pairwise. If two regions do not have any dimensions in common, they shall not overlap. Two ranges in the same dimension A = [a_{lower}, a_{upper}) and B = [b_{lower}, b_{upper}) shall overlap, if and only if either $a_{lower} = b_{lower}$ or ($a_{lower} < b_{upper}$ and $b_{lower} < a_{upper}$). The default region shall overlap with all nonempty region realizations.

The normalization of federation data to [0, a particular dimension's upper bound) for use with DDM services shall be left to the federation. The effects of DDM services shall be independent of the HLA time axis and HLA time management services.

Figure 17 depicts a region with two dimensions.

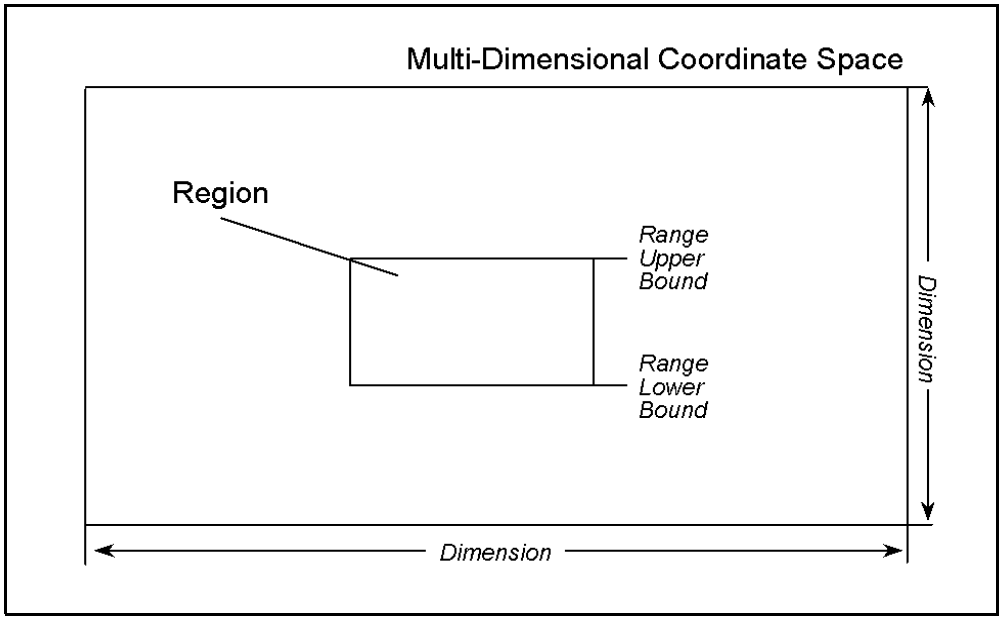

Figure 17—Region of two dimensions

9.1.5 Reinterpretation of selected DM services

Some DDM services can be used to perform similar functions to what is accomplished with DM services. When a joined federate uses DDM services, some of the DM definitions, constraints and services described in Clause 5 shall be extended to encompass the expanded interpretation of how DM services work when used in conjunction with DDM services by a joined federate, from the perspective of that joined federate.

A joined federate that is using DDM services shall interpret all uses of the following four DM services by any joined federate (including itself) in the federation execution:

— *Subscribe Object Class Attributes*

— *Unsubscribe Object Class Attributes*

— *Subscribe Interaction Class*

— *Unsubscribe Interaction Class*

as special cases of the following DDM services, respectively:

— *Subscribe Object Class Attributes With Regions*

— *Unsubscribe Object Class Attributes With Regions*

— *Subscribe Interaction Class With Regions*

— *Unsubscribe Interaction Class With Regions*

From the perspective of the joined federate that is using DDM services, each of the four DM services listed above shall be defined to be equivalent to the corresponding DDM service when invoked with a region argument of the default region.

In practice, because there shall be no way to refer to the default region, there shall be no way to substitute a DDM service for its corresponding DM service. Furthermore, a given joined federate may invoke both the DM services listed above and their corresponding DDM services using the same object class and class attribute designators or interaction class designators as arguments and there shall be no interaction between the subscription effects that result from the DM service invocations and those that result from the DDM service invocations.

For a joined federate that is using DDM services, the following expanded definitions and constraints shall replace the correspondingly numbered DM definitions and constraints that appear in 5.1.2 and 5.1.3.

Expanded definitions and constraints replace corresponding items in 5.1.2 as follows:

a) A class attribute may be used as an argument to *Subscribe Object Class Attributes*, *Subscribe Object Class Attributes With Regions, Unsubscribe Object Class Attributes, Unsubscribe Object Class Attributes With Regions, Publish Object Class Attributes*, and *Unpublish Object Class Attributes* service invocations for a particular object class if and only if the attribute is an available attribute of that object class.

b) From a joined federate's perspective, the subscribed attributes of an object class shall be all of the class attributes that have been arguments to *Subscribe Object Class Attributes* service invocations by that joined federate for that object class, and that have not subsequently been unsubscribed, either individually or by unsubscribing the whole object class. *Subscribe Object Class Attributes* and *Unsubscribe Object Class Attributes* service invocations for one object class shall have no effect on the subscribed attributes of any other object class.

   From a joined federate's perspective, the subscribed attributes of an object class with region shall be the class attributes that have been arguments to *Subscribe Object Class Attributes With Regions* service invocations by that joined federate for that object class and any region, assuming the joined federate did not subsequently invoke the *Unsubscribe Object Class Attributes With Regions* service for that (object class, region, class attribute) triple.

Subscribe Object Class Attributes With Regions and *Unsubscribe Object Class Attributes With Regions* service invocations for one (object class, region, class attribute) triple shall have no effect on the subscribed attributes of any other (object class, region, class attribute) triples.

Subscribe Object Class Attributes and *Unsubscribe Object Class Attributes* service invocations shall have no effect on any class attribute with region subscriptions that were established via the *Subscribe Object Class Attributes With Regions* service.

Subscribe Object Class Attributes With Regions and *Unsubscribe Object Class Attributes With Regions* service invocations shall have no effect on any class attribute subscriptions that were established via the *Subscribe Object Class Attributes* service.

c) If a class attribute is a subscribed attribute of an object class, the joined federate shall be subscribed to that class attribute either actively or passively, but not both. If a class attribute is a subscribed attribute of an object class with region, the joined federate shall be subscribed to that class attribute at a given object class and region either actively or passively, but not both.

h) From a joined federate's perspective, an object class shall be subscribed if and only if

1) The joined federate is subscribed to at least one attribute of the object class, or

2) The joined federate is subscribed with region to at least one attribute of the object class.

j) Joined federates may invoke the *Register Object Instance* and the *Register Object Instance With Regions* services only with a published object class as an argument.

k) The registered class of an object instance shall be the object class that was an argument to either the *Register Object Instance* or the *Register Object Instance With Regions* service invocation for that object instance.

s) An update to an instance attribute by the joined federate that owns that instance attribute may be reflected only by other joined federates that are either

1) Subscribed to the instance attribute's corresponding class attribute at the known class of the object instance that contains that instance attribute at the subscribing joined federate, or

2) Using a subscription region set for the instance attribute's corresponding class attribute at the known class of the object instance at the subscribing joined federate, and this set overlaps the update region set of the specified instance attribute at the time that the *Update Attribute Values* service was invoked. An update shall be reflected at the subscribing joined federate at most once, regardless of another subscription to the class attribute using *Subscribe Object Class Attributes*.

Expanded definitions and constraints replace corresponding items in 5.1.3 as follows:

a) From a joined federate's perspective, an interaction class shall be subscribed if and only if

1) It was an argument to a *Subscribe Interaction Class* service invocation by that joined federate that was not subsequently followed by an *Unsubscribe Interaction Class* service invocation for that interaction class, or

2) There is at least one region such that the interaction class and region were arguments to a *Subscribe Interaction Class With Regions* service invocation by that joined federate that was not subsequently followed by an *Unsubscribe Interaction Class With Regions* service invocation for that interaction class and region.

b) If an interaction class is subscribed, the joined federate shall be subscribed to that interaction class either actively or passively, but not both. If an interaction class is subscribed with region, the joined federate shall be subscribed to that interaction class with a given region either actively or passively, but not both.

d) Joined federates may invoke the *Send Interaction* and the *Send Interaction With Regions* services only with a published interaction class as an argument.

e) The sent class of an interaction shall be the interaction class that was an argument to the *Send Interaction* or the *Send Interaction With Regions* service invocation for that interaction.

g) The *Receive Interaction †* service shall be invoked at a joined federate only with a subscribed interaction class as an argument. An interaction shall be received at the joined federate at most once, regardless of another subscription to the interaction class using *Subscribe Interaction Class*.

j) Only the available parameters of an interaction class may be used in a *Send Interaction* and *Send Interaction With Regions* service invocation with that interaction class as an argument.

k) The sent parameters of an interaction shall be the parameters that were arguments to the *Send Interaction* or *Send Interaction With Regions* service invocation for that interaction.

9.1.6 Reinterpretation of selected object management services

Some DDM services can be used to perform similar functions to what is accomplished with object management services. When a joined federate uses DDM services, three of the object management services described in Clause 6 shall be extended to encompass the expanded interpretation of how object management services work when used in conjunction with DDM services by a joined federate, from the perspective of that joined federate.

A joined federate using DDM services shall interpret all uses of the following three DM services by any joined federate in the federation execution (including itself):

— *Register Object Instance*

— *Send Interaction*

— *Request Attribute Value Update*

as special cases of the following DDM services, respectively:

— *Register Object Instance With Regions*

— *Send Interaction With Regions*

— *Request Attribute Value Update With Regions*

From the perspective of the joined federate that is using DDM services, each of the three object management services listed above shall be defined to be equivalent to the corresponding DDM service when invoked with a region argument of the default region.

9.1.7 Convey region designator sets switch

The federation execution-wide Convey Region Designator Sets Switch indicates whether the RTI should provide the optional Set of Sent Region Designators argument with invocations of *Reflect Attribute Values †* and *Receive Interaction †* services.

The initial setting for this switch shall come from the FDD at federation execution creation time. This switch setting shall be modifiable during a federation execution via the HLAmanager.HLAfederation.HLAadjust.HLAsetSwitches MOM interaction.

Whenever this switch is Enabled, the optional Set of Sent Region Designators (region realizations) argument shall be provided (as appropriate) with all *Reflect Attribute Values †* and *Receive Interaction †* service invocations at all joined federates.

Whenever this switch is Enabled, the optional Set of Sent Region Designators argument shall be provided (as appropriate) with all *Reflect Attribute Values †* and *Receive Interaction †* service invocations at all joined federates. For *Reflect Attribute Values †*, if the specified instance attributes have available dimensions, the Set of Sent Region Designators argument shall contain the respective update region set. For *Receive Interaction †*, if the specified interaction has available dimensions, the Set of Sent Region Designators argument shall contain the respective update region set.

If the specified interaction or corresponding class attributes has available dimensions, this switch is Enabled, and no region designator set is supplied, then the default region shall have been used on the sending side. In this case, the Convey Region set argument shall be present and empty.

A joined federate shall use conveyed region sets only in the *Get Range Bounds* and *Get Dimension Handle Set* service invocations.

9.2 Create Region

The *Create Region* service shall create a region that has the specified dimensions. The region may be used for either update or subscription.

9.2.1 Supplied arguments

a) Set of dimension designators

9.2.2 Returned arguments

a) Region designator

9.2.3 Preconditions

a) The federation execution exists.

b) The federate is joined to that federation execution.

c) The specified dimension designator exists.

d) Save not in progress.

e) Restore not in progress.

9.2.4 Postconditions

a) A region has been created with the specified dimensions.

9.2.5 Exceptions

a) Invalid dimension designator.

b) The federate is not a federation execution member.

c) Save in progress.

d) Restore in progress.

e) RTI internal error.

9.2.6 Reference state charts

a) None

9.3 Commit Region Modifications

The *Commit Region Modifications* service shall inform the RTI about changes to the ranges of the regions.

9.3.1 Supplied arguments

a) Set of region designators

9.3.2 Returned arguments

a) None

9.3.3 Preconditions

a) The federation execution exists.

b) The federate is joined to that federation execution.

c) The regions exist.

d) The regions were created by the invoking joined federate using the *Create Region* service.

e) Save not in progress.

f) Restore not in progress.

9.3.4 Postconditions

a) The specified regions are modified.

b) All update region realizations derived from the region specifications are modified.

c) The RTI has been informed of the joined federate's requested change to all subscription region realizations derived from the region specifications.

9.3.5 Exceptions

a) Invalid region.

b) The region was not created by the joined federate using the *Create Region* service.

c) The federate is not a federation execution member.

d) Save in progress.

e) Restore in progress.

f) RTI internal error.

9.3.6 Reference state charts

a) None

9.4 Delete Region

The *Delete Region* service shall delete the specified region. A region in use for subscription or update should not be deleted.

9.4.1 Supplied arguments

a) Region designator

9.4.2 Returned arguments

a) None

9.4.3 Preconditions

a) The federation execution exists.

b) The federate is joined to that federation execution.

c) The region exists.

d) The region was created by the invoking joined federate using the *Create Region* service.

e) The region is not in use for update or subscription.

f) Save not in progress.

g) Restore not in progress.

9.4.4 Postconditions

a) The region no longer exists.

9.4.5 Exceptions

a) Invalid region.

b) The region was not created by the joined federate using the *Create Region* service.

c) The region is in use for update or subscription.

d) The federate is not a federation execution member.

e) Save in progress.

f) Restore in progress.

g) RTI internal error.

9.4.6 Reference state charts

a) None

9.5 Register Object Instance With Regions

The *Register Object Instance With Regions* service shall create a unique object instance designator and link it with an object instance of the supplied object class. All instance attributes of the object instance for which the corresponding class attributes are currently published by the registering joined federate shall be set as owned by the registering joined federate.

This service shall be used to create an object instance and simultaneously associate update regions with instance attributes of that object instance. This service shall be an atomic operation that can be used in place of *Register Object Instance* followed by *Associate Regions For Updates* when the atomicity is required. Those instance attributes whose corresponding class attributes are currently published and that have available dimensions, but are not supplied in the service invocation, shall be associated with the default region.

If the region set paired with an instance attribute is empty, and the corresponding class attribute of the instance attribute has available dimensions, that instance attribute shall be associated with the default region.

If a joined federate loses ownership of an instance attribute that it had associated with an update region, the association shall be lost. If the joined federate subsequently acquires ownership of that instance attribute, the original association shall not be restored.

If the optional object instance name argument is supplied, that name shall have been successfully reserved as indicated by the *Object Instance Name Reserved†* service and shall be coadunated with the object instance. If the optional object instance name argument is not supplied, the RTI shall create one when needed (*Get Object Instance Name* service).

9.5.1 Supplied arguments

a) Object class designator

b) Collection of attribute designator set and region designator set pairs

c) Optional object instance name

9.5.2 Returned arguments

a) Object instance designator

9.5.3 Preconditions

a) The federation execution exists.

b) The federate is joined to that federation execution.

c) The object class is defined in the FDD.

d) The joined federate is publishing the object class.

e) The class attributes are available at the specified object class.

f) The joined federate is publishing the specified class attributes of the specified object class.

g) The regions exist.

h) The regions were created by the invoking joined federate using the *Create Region* service.

i) For each class attribute/region pair, the dimensions denoted by the region are a subset of the available dimensions of the specified class attributes in the FDD.

j) If the optional object instance name argument is supplied, that name is reserved and not already coadunated with another object instance.

k) Save not in progress.

l) Restore not in progress.

9.5.4 Postconditions

a) The returned object instance designator is associated with the object instance.

b) The joined federate owns the instance attributes that correspond to those class attributes that are published attributes at the specified object class.

c) The specified instance attributes are associated with the respective regions for future *Update Attribute Values* service invocations.

d) If the optional object instance name argument is supplied, that name is associated with the object instance.

9.5.5 Exceptions

a) The object class is not defined in the FDD.

b) The joined federate is not publishing the object class.

c) The class attribute is not available at the known class of the object instance.

d) The joined federate is not publishing the class attribute.

e) Invalid region.

f) The region was not created by the joined federate using the *Create Region* service.

g) The specified dimensions of the region are not a subset of the available dimensions of the specified class attributes in the FDD.

h) The object instance name was not reserved (if optional object instance name argument is supplied).

i) The object instance name was already coadunated with another object instance (if optional object instance name argument is supplied).

j) The federate is not a federation execution member.

k) Save in progress.

l) Restore in progress.

m) RTI internal error.

9.5.6 Reference state charts

a) Figure 9: Object class (i) {implicit reference}

b) Figure 14: Implications of ownership of instance attribute (i, k, j) {implicit reference}

c) Figure 15: Establishing ownership of instance attribute (i, k, j) {implicit reference}

9.6 Associate Regions For Updates

The *Associate Regions For Updates* service shall associate regions to be used for updates with instance attributes of a specific object instance.

A joined federate that has associated regions with an instance attribute for update should ensure that the regions adequately and correctly bound the portion of space defined by dimensions in which communication of updates to the instance attribute are relevant at the time when an *Update Attribute Values* service is invoked.

The association shall be used by the *Update Attribute Values* service to route data to subscribers whose subscription region sets overlap the specified update region set. This service shall add the specified regions to the set of associations of each specified instance attribute. No changes shall be made to the association set if the specified regions are already in the set of associations of the specified instance attributes.

If the region set paired with an instance attribute is empty, no associations shall be added for that instance attribute, not even the default region.

If a joined federate loses ownership of an instance attribute that it had associated with an update region, the association shall be lost. If the joined federate subsequently acquires ownership of that instance attribute, the original association shall not be restored.

9.6.1 Supplied arguments

a) Object instance designator

b) Collection of attribute designator set and region designator set pairs

9.6.2 Returned arguments

a) None

9.6.3 Preconditions

a) The federation execution exists.

b) The federate is joined to that federation execution.

c) An object instance with the specified designator exists.

d) The joined federate knows about the object instance with the specified designator.

e) The class attributes are available at the specified object class.

f) The regions exist.

g) The regions were created by the invoking joined federate using the *Create Region* service.

h) For each instance attribute/region pair, the dimensions denoted by the region are a subset of the available dimensions of the corresponding class attributes in the FDD.

i) Save not in progress.

j) Restore not in progress.

9.6.4 Postconditions

a) The specified instance attributes are associated with the respective regions for future *Update Attribute Values* service invocations.

9.6.5 Exceptions

a) The object instance is not known.

b) The class attribute is not available.

c) Invalid region.

d) The region was not created by the joined federate using the *Create Region* service.

e) The dimensions denoted by the region are not a subset of the available dimensions of the specified class attributes in the FDD.

f) The federate is not a federation execution member.

g) Save in progress.

h) Restore in progress.

i) RTI internal error.

9.6.6 Reference state charts

a) None

9.7 Unassociate Regions For Updates

The *Unassociate Regions For Updates* service shall remove the association between the regions and the specified instance attributes. No changes shall be made to the association set if the specified regions are not in the set of associations of the specified instance attributes.

Each of the instance attributes that are unassociated by the invocation of this service shall be associated with the default region if and only if they are not associated with any other update region.

If the region set paired with an instance attribute is empty, no associations shall be removed for that instance attribute.

9.7.1 Supplied arguments

a) Object instance designator

b) Collection of attribute designator set and region designator set pairs

9.7.2 Returned arguments

a) None

9.7.3 Preconditions

a) The federation execution exists.

b) The federate is joined to that federation execution.

c) An object instance with the specified designator exists.

d) The joined federate knows about the object instance with the specified designator.

e) The corresponding class attributes are available at the known class of the object instance at the invoking federate.

f) The regions exist.

g) The regions were created by the invoking joined federate using the *Create Region* service.

h) Save not in progress.

i) Restore not in progress.

9.7.4 Postconditions

a) The regions are no longer associated with the specified instance attributes.

b) All affected instance attributes that no longer have regions associated with them shall now be associated with the default region.

9.7.5 Exceptions

a) The object instance is not known.

b) The class attribute is not available.

c) Invalid region.

d) The region was not created by the joined federate using the *Create Region* service.

e) The federate is not a federation execution member.

f) Save in progress.

g) Restore in progress.

h) RTI internal error.

9.7.6 Reference state charts

a) None

9.8 Subscribe Object Class Attributes With Regions

The *Subscribe Object Class Attributes With Regions* service shall specify an object class for which the RTI shall begin notifying the joined federate of discovery of instantiated object instances when at least one of that object instance's instance attributes are in scope. This service and subsequent related RTI operations shall behave analogously to the *Subscribe Object Class Attributes* service as described in 5.6 and its subsequent related RTI operations. This service shall provide additional functionality in that the overlap of the relevant subscription and update region sets affects the subsequent RTI operations, as described in 9.1.5.

This service shall add the specified regions to the set of subscriptions of the specified class attributes. No changes shall be made to the subscription set if the specified regions are already in the set of subscriptions of the specified class attributes.

Invocations of the *Subscribe Object Class Attributes With Regions* service shall have no affect on any object class or class attribute subscriptions that were established via the *Subscribe Object Class Attributes* service. Subscriptions that are established via the *Subscribe Object Class Attributes With Regions* service shall not be affected by invocations of either the *Subscribe Object Class Attributes* service or the *Unsubscribe Object Class Attributes* service.

If the region set paired with a class attribute is empty, no subscriptions shall be added for that class attribute, not even the default region. Use of an empty set is not equivalent to invocation of the *Subscribe Object Class Attributes* service.

If the optional passive subscription indicator indicates that this is a passive subscription, than the following apply:

a) The invocation of this service shall not cause the *Start Registration For Object Class †* service or the *Turn Updates On For Object Instance †* service to be invoked at any other joined federate, and

b) If a specified region was used in a previous subscription that was active rather than passive, this invocation shall change the previous subscription to passive. Invocation of this service may cause the *Stop Registration for Object Class †* service or the *Turn Updates Off For Object Instance †* service to be invoked at one or more other joined federates.

If the optional passive subscription indicator is not present or indicates that this is an active subscription, the invocation of this service may cause the *Start Registration For Object Class †* service or the *Turn Updates On For Object Instance †* service to be invoked at one or more other joined federates.

9.8.1 Supplied arguments

a) Object class designator

b) Collection of attribute designator set and region designator set pairs

c) Optional passive subscription indicator

9.8.2 Returned arguments

a) None

9.8.3 Preconditions

a) The federation execution exists.

b) The federate is joined to that federation execution.

c) The object class is defined in the FDD.

d) The class attributes are available at the specified object class.

e) The regions exist.

f) The regions were created by the invoking joined federate using the *Create Region* service.

g) For each class attribute/region pair, the dimensions denoted by the region are a subset of the available dimensions of the specified class attributes in the FDD.

h) Save not in progress.

i) Restore not in progress.

9.8.4 Postconditions

a) The RTI has been informed of the joined federate’s requested subscription.

9.8.5 Exceptions

a) The object class is not defined in the FDD.

b) The class attribute is not available at the specified object class.

c) Invalid region.

d) The region was not created by the joined federate using the *Create Region* service.

e) The specified dimensions of the region are not a subset of the available dimensions of the specified class attributes in the FDD.

f) Invalid passive subscription indicator (if optional passive subscription indicator argument is supplied).

g) The federate is not a federation execution member.

h) Save in progress.

i) Restore in progress.

j) RTI internal error.

9.8.6 Reference state charts

a) Figure 10: Class attribute (i, j) {implicit reference}

b) Figure 11: Class attribute (i, HLAprivilegeToDeleteObject) {implicit reference}

9.9 Unsubscribe Object Class Attributes With Regions

The *Unsubscribe Object Class Attributes With Regions* service shall inform the RTI that it shall stop notifying the joined federate of object instance discoveries and attribute updates for instance attributes of the specified object class in the specified region. The unsubscribe shall be confined to only the specified class attribute subscriptions using the specified region. No changes shall be made to the subscription set of the specified class attributes if the specified regions are not in the set of subscriptions of the specified class attributes.

If the region set paired with a class attribute is empty, no subscriptions shall be removed for that class attribute. Use of an empty set is not equivalent to invocation of the *Unsubscribe Object Class Attributes* service.

Invocation of this service, subject to other conditions, may cause an implicit out-of-scope situation for affected instance attributes, i.e., invoking joined federate will NOT be notified via an invocation of *Attributes Out of Scope †* (6.14) service invocation when the instance attribute goes out-of-scope.

9.9.1 Supplied arguments

a) Object class designator

b) Collection of attribute designator set and region designator set pairs

9.9.2 Returned arguments

a) None

9.9.3 Preconditions

a) The federation execution exists.

b) The federate is joined to that federation execution.

c) The object class is defined in the FDD.

d) The class attributes are available at the specified object class.

e) The regions exist.

f) The regions were created by the joined federate using the *Create Region* service.

g) Save not in progress.

h) Restore not in progress.

9.9.4 Postconditions

a) The joined federate is not subscribed to the specified class attributes with the specified regions.

b) May cause an implicit out-of-scope situation for affected instance attributes at invoking joined federate.

9.9.5 Exceptions

a) The object class is not defined in the FDD.

b) The class attribute is not available at the specified object class.

c) Invalid region.

d) The region was not created by the invoking joined federate using the *Create Region* service.

e) The federate is not a federation execution member.

f) Save in progress.

g) Restore in progress.

h) RTI internal error.

9.9.6 Reference state charts

a) Figure 10: Class attribute (i, j) {implicit reference}

b) Figure 11: Class attribute (i, HLAprivilegeToDeleteObject) {implicit reference}

9.10 Subscribe Interaction Class With Regions

The *Subscribe Interaction Class With Regions* service shall specify the class of interactions that shall be delivered to the joined federate, taking the region into account. This service and subsequent related RTI operations shall behave analogously to the *Subscribe Interaction Class* service as described in 5.8. This service shall provide additional functionality in that the overlap of the region set used for subscription of the interaction and the region set used for sending the interaction affects the subsequent RTI operations, as described in 9.1.5.

This service shall add the specified regions to the set of regions associated with the specified interaction class subscription. No changes shall be made to the subscription set if the specified regions are already in the set of subscriptions of the specified interaction class.

Invocations of the *Subscribe Interaction Class With Regions* service shall have no affect on any interaction class subscriptions that were established via the *Subscribe Interaction Class* service. Subscriptions that are established via the *Subscribe Interaction Class With Regions* service shall not be affected by invocations of either the *Subscribe Interaction Class* service or the *Unsubscribe Interaction Class* service.

If the region set provided is empty, no subscription to the interaction class shall be made, not even the default region. Use of an empty set is not equivalent to invocation of the *Subscribe Interaction Class* service.

If the optional passive subscription indicator indicates that this is a passive subscription, than the following apply:

a) The invocation of this service shall not cause the *Turn Interactions On †* service to be invoked at any other joined federate, and

b) If a specified region was used in a previous subscription that was active rather than passive, this service invocation shall change the previous subscription to passive. Invocation of this service may cause the *Turn Interactions Off †* service to be invoked at one or more other joined federates.

If the optional passive subscription indicator is not present or indicates that this is an active subscription, the invocation of this service may cause the *Turn Interactions On †* service to be invoked at one or more other joined federates.

9.10.1 Supplied arguments

a) Interaction class designator

b) Set of region designators

c) Optional passive subscription indicator

9.10.2 Returned arguments

a) None

9.10.3 Preconditions

a) The federation execution exists.

b) The federate is joined to that federation execution.

c) The interaction class is defined in the FDD.

d) The regions exist.

e) The regions were created by the invoking joined federate using the *Create Region* service.

f) The specified dimensions of the regions are a subset of the available dimensions of the specified interaction class in the FDD.

g) If the specified interaction class designator is the MOM interaction class HLAmanager.HLAfederate.HLAreport.HLAreportServiceInvocation, the federate's service invocations are not being reported via MOM interactions.

h) Save not in progress.

i) Restore not in progress.

9.10.4 Postconditions

a) The RTI has been informed of the joined federate's requested subscription.

9.10.5 Exceptions

a) The interaction class is not defined in the FDD.

b) Invalid region.

c) The region was not created by the joined federate using the *Create Region* service.

d) The specified dimensions of the region are not a subset of the available dimensions of the specified interaction class in the FDD.

e) Invalid passive subscription indicator (if optional passive subscription indicator argument is supplied).

f) Federate service invocations are being reported via MOM interactions.

g) The federate is not a federation execution member.

h) Save in progress.

i) Restore in progress.

j) RTI internal error.

9.10.6 Reference state charts

a) Figure 12: Interaction class (m) {implicit reference}

9.11 Unsubscribe Interaction Class With Regions

The *Unsubscribe Interaction Class With Regions* service shall inform the RTI that it shall no longer notify the joined federate of interactions of the specified class that are sent into the specified region. No changes shall be made to the subscription set of the specified interaction class if the specified regions are not in the set of subscriptions of the specified interaction class.

If the region set provided is empty, no subscription to the interaction class shall not be removed. Use of an empty set is not equivalent to invocation of the *Unsubscribe Interaction Class* service.

9.11.1 Supplied arguments

a) Interaction class designator

b) Set of region designators

9.11.2 Returned arguments

a) None

9.11.3 Preconditions

a) The federation execution exists.

b) The federate is joined to that federation execution.

c) The interaction class is defined in the FDD.

d) The regions exist.

e) The regions were created by the invoking joined federate using the *Create Region* service.

f) Save not in progress.

g) Restore not in progress.

9.11.4 Postconditions

a) The joined federate is not subscribed to the specified interaction class with the specified regions.

9.11.5 Exceptions

a) The interaction class is not defined in the FDD.

b) Invalid region.

c) The region was not created by the joined federate using the *Create Region* service.

d) The federate is not a federation execution member.

e) Save in progress.

f) Restore in progress.

g) RTI internal error.

9.11.6 Reference state charts

a) Figure 12: Interaction class (m) {implicit reference}

9.12 Send Interaction With Regions

The *Send Interaction With Regions* service shall send an interaction into the federation. The interaction parameters shall only be those in the specified class and all superclasses, as defined in the FDD. The regions shall be used to limit the scope of potential receivers of the interaction.

If preferred order type (see 8.1.1) of the interaction is TSO, the joined federate invoking this service is time-regulating, and the time stamp argument is present, this service shall return a federation-unique message retraction designator. Otherwise, no message retraction designator shall be returned. A time stamp value should be provided if the preferred order type of the interaction is TSO and the joined federate invoking this service is time-regulating.

Following the invocation of this service, if at any time when the RTI has not yet invoked a corresponding *Receive Interaction †* service at a potentially receiving joined federate, that joined federate either is

— Unsubscribed to the specified interaction class, or

— Using a subscription region set for the specified interaction class and this region set does not overlap the update region set used for sending the interaction

The RTI may not invoke the corresponding *Receive Interaction †* service at that joined federate.

If at any time between the invocation of this service for a particular interaction class and what would be the invocation of the corresponding *Receive Interaction †* service at a potential receiving joined federate, that joined federate either is

— Subscribed to the specified interaction class, or

— Using a subscription region set for the specified interaction class, and this region set overlaps the update region set used for sending the interaction.

The RTI may invoke the corresponding *Receive Interaction †* service at that joined federate, subject to the preconditions of the *Receive Interaction †* service and the rules of time management.

If the region set provided is empty, the interaction shall not be sent. Use of an empty set is not equivalent to invocation of the *Send Interaction* service.

9.12.1 Supplied arguments

a) Interaction class designator

b) Constrained set of parameter designator and value pairs

c) Set of region designators

d) User-supplied tag

e) Optional time stamp

9.12.2 Returned arguments

a) Optional message retraction designator

9.12.3 Preconditions

a) The federation execution exists.

b) The federate is joined to that federation execution.

c) The interaction class is defined in the FDD.

d) The joined federate is publishing the interaction class.

e) The interaction parameters are available.

f) The regions exist.

g) The regions were created by the invoking joined federate using the *Create Region* service.

h) The specified dimensions of the regions are a subset of the available dimensions of the specified interaction class in the FDD.

i) The time stamp value shall be in accordance with the constraints stated in Clause 8 (if optional time stamp argument is supplied).

j) Save not in progress.

k) Restore not in progress.

9.12.4 Postconditions

a) The RTI has received the interaction.

9.12.5 Exceptions

a) The interaction class is not defined in the FDD.

b) The joined federate is not publishing the specified interaction class.

c) The interaction parameter is not available at the specified interaction class.

d) Invalid region.

e) The region was not created by the joined federate using the *Create Region* service.

f) The specified dimensions of the region are not a subset of the available dimensions of the specified interaction class in the FDD.

g) The time stamp is invalid (if optional time stamp argument is supplied).

h) The federate is not a federation execution member.

i) Save in progress.

j) Restore in progress.

k) RTI internal error.

9.12.6 Reference state charts

a) None

9.13 Request Attribute Value Update With Regions

The *Request Attribute Value Update With Regions* service shall be used to stimulate the update of values of specified attributes. The RTI shall solicit the values of the specified instance attributes, for all object instances registered at that class and all subclasses of that class, from their owners using the *Provide Attribute Value Update †* service. The resulting *Provide Attribute Value Update †* service invocations issued by the RTI shall be consistent with the region sets provided to this service. An invocation shall be consistent with the provided region sets if the instance attributes in an updating joined federate are associated with an update region set that overlaps with the corresponding region set.

The time stamp of any resulting *Reflect Attribute Values †* service invocations shall be determined by the updating joined federate.

If the federate invoking this service also owns some of the instance attributes whose values are being requested, it will not have the *Provide Attribute Value Update†* service invoked on it by the RTI. The request is taken as an implicit invocation of the provide service at the requesting federate for all qualified instance attributes that it owns.

This service does not allow the use of object instances. If a federate wants to request the update of instance attributes of a single object instance, use of *Request Attribute Value Update* is sufficient and the use of DDM provides no additional benefit.

If the region set paired with a class attribute is empty, no request for update shall be issued for the class attribute, not even with the default region. Use of an empty set is not equivalent to invocation of the *Request Attribute Value Update* service.

The user-supplied tag argument shall be provided with all corresponding *Provide Attribute Value Update †* service invocations.

9.13.1 Supplied arguments

a) Object class designator

b) Collection of attribute designator set and region designator set pairs

c) User supplied tag

9.13.2 Returned arguments

a) None

9.13.3 Preconditions

a) The federation execution exists.

b) The federate is joined to that federation execution.

c) The object class is defined in the FDD.

d) The class attributes are available at the specified object class.

e) The regions exist.

f) The regions were created by the invoking joined federate using the *Create Region* service.

g) The specified dimensions of the regions are a subset of the available dimensions of the specified class attributes in the FDD.

h) Save not in progress.

i) Restore not in progress.

9.13.4 Postconditions

a) The request for the updated attribute values has been received by the RTI.

9.13.5 Exceptions

a) The object class is not defined in the FDD

b) The class attribute is not available.

c) Invalid region.

d) The region was not created by the joined federate using the *Create Region* service.

e) The specified dimensions of the region are not a subset of the available dimensions of the specified class attributes in the FDD.

f) The joined federate is not a federation execution member.

g) Save in progress.

h) Restore in progress.

i) RTI internal error.

9.13.6 Reference state charts

a) None

10. Support services

10.1 Overview

This clause describes miscellaneous services utilized by joined federates for performing such actions as

— Name-to-handle and handle-to-name transformation

— Setting advisory switches

— Manipulating regions

— RTI start-up and shutdown

10.1.1 Names

All class name arguments shall be completely specified, including all superclass names, with the exception of the HLA root classes. The inclusion of HLAobjectRoot or HLAinteractionRoot shall be optional.

Each name in an FDD (object classes, interactions, attributes, parameters, dimensions, transportation types, order types) shall have a unique, and unpredictable, federation execution-wide handle.

All names shall be case sensitive.

10.1.2 Advisory switches

The settings of the following advisory switches can be modified by services in Clause 10:

— Class Relevance Advisory Switch

— Attribute Relevance Advisory Switch

— Attribute Scope Advisory Switch

— Interaction Relevance Advisory Switch

The federation-wide initial setting of each advisory switch shall be as indicated in the FDD used at federation execution creation time. Subsequent invocation of the advisory switch services in this clause shall affect a switch setting for the invoking federate only.

The enabling of an advisory switch directs that the RTI shall inform the joined federate, via the appropriate advisory notifications, whenever the conditions covered by that advisory change. The disabling of an advisory switch directs the RTI to cease informing the joined federate of changes in the conditions covered by that advisory. When an advisory switch is disabled, all relevant notifications that would have been delivered to the joined federate shall be lost, i.e., the RTI shall not save the relevant actions and send them to the joined federate when a switch is subsequently enabled.

There is, however, one case in which the RTI shall not inform a joined federate when specific conditions covered by an advisory change, even if that advisory switch is enabled. Specifically, even if the Attribute Scope Advisory Switch is enabled, there are some actions a joined federate could take that would cause an instance attribute to go out-of-scope with respect to that joined federate. Since these changes are a result of the joined federate’s own action and the joined federate knows that the action will cause the instance attribute to go out-of-scope, no out-of-scope advisory notification will be given the joined federate. The actions that could cause an instance attribute to go out-of-scope for the acting joined federate are as follows:

— The joined federate acquires ownership of the instance attribute.

— The joined federate unsubscribes to the instance attribute’s corresponding class attribute, and it is not subscribed to the instance attribute’s corresponding class attribute with region.

— The joined federate unsubscribes with region to the instance attribute's corresponding class attribute such that it is no longer subscribed to the class attribute (with or without region).

These cases are referred to as implicit out-of-scope situations because no out-of-scope advisory notification is generated.

Note that if a joined federate unsubscribes with region to an instance attribute's corresponding class attribute, but is still subscribed to that class attribute with some other region, it may or may not have caused the instance attribute to go out-of-scope (assuming it was previously in-scope). Since the joined federate does not know if its removal has caused overlap to be lost with the update region set, it must in this case rely on the RTI to inform it whether or not the instance attribute has gone out-of-scope. Only if the joined federate completely removes all subscriptions (DM and DDM based subscriptions) can it know with certainty that it is causing an implicit out-of-scope situation.

10.2 Get Object Class Handle

The *Get Object Class Handle* service shall return the object class handle associated with the supplied object class name.

10.2.1 Supplied arguments

a) Object class name

10.2.2 Returned arguments

a) Object class handle

10.2.3 Preconditions

a) The federation execution exists.

b) The federate is joined to that federation execution.

c) The specified object class is defined in the FDD.

10.2.4 Postconditions

a) The joined federate has the requested object class handle.

10.2.5 Exceptions

a) The object class is not defined in the FDD.

b) The federate is not a federation execution member.

c) RTI internal error.

10.2.6 Reference state charts

a) None

10.3 Get Object Class Name

The *Get Object Class Name* service shall return the object class name associated with the supplied object class handle.

10.3.1 Supplied arguments

a) Object class handle

10.3.2 Returned arguments

a) Object class name

10.3.3 Preconditions

a) The federation execution exists.

b) The federate is joined to that federation execution.

c) The specified object class handle exists.

10.3.4 Postconditions

a) The joined federate has the requested object class name.

10.3.5 Exceptions

a) Invalid object class handle.

b) The federate is not a federation execution member.

c) RTI internal error.

10.3.6 Reference state charts

a) None

10.4 Get Attribute Handle

The *Get Attribute Handle* service shall return the class attribute handle associated with the supplied class attribute name and object class.

10.4.1 Supplied arguments

a) Object class handle

b) Class attribute name

10.4.2 Returned arguments

a) Class attribute handle

10.4.3 Preconditions

a) The federation execution exists.

b) The federate is joined to that federation execution.

c) The specified object class handle exists.

d) The specified class attribute is an available attribute of the specified object class.

10.4.4 Postconditions

a) The joined federate has the requested class attribute handle.

10.4.5 Exceptions

b) Invalid object class handle.

c) The object class attribute is not an available attribute of the object class.

d) The federate is not a federation execution member.

e) RTI internal error.

10.4.6 Reference state charts

a) None

10.5 Get Attribute Name

The *Get Attribute Name* service shall return the class attribute name associated with the supplied class attribute handle and object class.

10.5.1 Supplied arguments

a) Object class handle

b) Class attribute handle

10.5.2 Returned arguments

a) Class attribute name

10.5.3 Preconditions

a) The federation execution exists.

b) The federate is joined to that federation execution.

c) The specified object class handle exists.

d) The specified class attribute handle exists.

e) The specified class attribute is an available attribute of the specified object class.

10.5.4 Postconditions

a) The joined federate has the requested class attribute name.

10.5.5 Exceptions

a) Invalid object class handle.

b) Invalid class attribute handle.

c) The object class attribute is not an available attribute of the object class.

d) The federate is not a federation execution member.

e) RTI internal error.

10.5.6 Reference state charts

a) None

10.6 Get Interaction Class Handle

The *Get Interaction Class Handle* service shall return the interaction class handle associated with the supplied interaction class name.

10.6.1 Supplied arguments

a) Interaction class name

10.6.2 Returned arguments

a) Interaction class handle

10.6.3 Preconditions

a) The federation execution exists.

b) The federate is joined to that federation execution.

c) The specified interaction class is defined in the FDD.

10.6.4 Postconditions

a) The joined federate has the requested interaction class handle.

10.6.5 Exceptions

a) The interaction class is not defined in the FDD.

b) The federate is not a federation execution member.

c) RTI internal error.

10.6.6 Reference state charts

a) None

10.7 Get Interaction Class Name

The *Get Interaction Class Name* service shall return the interaction class name associated with the supplied interaction class handle.

10.7.1 Supplied arguments

a) Interaction class handle

10.7.2 Returned arguments

a) Interaction class name

10.7.3 Preconditions

a) The federation execution exists.

b) The federate is joined to that federation execution.

c) The specified interaction class handle exists.

10.7.4 Postconditions

a) The joined federate has the requested interaction class name.

10.7.5 Exceptions

a) Invalid interaction class handle.

b) The federate is not a federation execution member.

c) RTI internal error.

10.7.6 Reference state charts

a) None

10.8 Get Parameter Handle

The *Get Parameter Handle* service shall return the interaction parameter handle associated with the supplied parameter name and interaction class.

10.8.1 Supplied arguments

a) Interaction class handle

b) Parameter name

10.8.2 Returned arguments

a) Parameter handle

10.8.3 Preconditions

a) The federation execution exists.

b) The federate is joined to that federation execution.

c) The specified interaction class handle exists.

d) The specified parameter is an available parameter of the specified interaction class.

10.8.4 Postconditions

a) The joined federate has the requested parameter handle.

10.8.5 Exceptions

a) Invalid interaction class handle.

b) The parameter is not an available parameter of the interaction class.

c) The federate is not a federation execution member.

d) RTI internal error.

10.8.6 Reference state charts

a) None

10.9 Get Parameter Name

The *Get Parameter Name* service shall return the interaction parameter name associated with the supplied parameter handle and interaction class.

10.9.1 Supplied arguments

a) Interaction class handle

b) Parameter handle

10.9.2 Returned arguments

a) Parameter name

10.9.3 Preconditions

a) The federation execution exists.

b) The federate is joined to that federation execution.

c) The specified interaction class handle exists.

d) The specified parameter handle exists.

e) The specified parameter is an available parameter of the specified interaction class.

10.9.4 Postconditions

a) The joined federate has the requested parameter name.

10.9.5 Exceptions

a) Invalid interaction class handle.

b) Invalid parameter handle.

c) The parameter is an available parameter of the interaction class.

d) The federate is not a federation execution member.

e) RTI internal error.

10.9.6 Reference state charts

a) None

10.10 Get Object Instance Handle

The *Get Object Instance Handle* service shall return the handle of the object instance with the supplied name.

10.10.1 Supplied arguments

a) Object instance name

10.10.2 Returned arguments

a) Object instance handle

10.10.3 Preconditions

a) The federation execution exists.

b) The federate is joined to that federation execution.

c) The object instance with the specified name exists.

d) The joined federate knows about the object instance with the specified name.

10.10.4 Postconditions

a) The joined federate has the requested object instance handle.

10.10.5 Exceptions

a) The object instance is not known.

b) The federate is not a federation execution member.

c) RTI internal error.

10.10.6 Reference state charts

a) None

10.11 Get Object Instance Name

The *Get Object Instance Name* service shall return the name of the object instance with the supplied handle.

10.11.1 Supplied arguments

a) Object instance handle

10.11.2 Returned arguments

a) Object instance name

10.11.3 Preconditions

a) The federation execution exists.

b) The federate is joined to that federation execution.

c) The object instance with the specified handle exists.

d) The joined federate knows about the object instance with the specified handle.

10.11.4 Postconditions

a) The joined federate has the requested object instance name.

10.11.5 Exceptions

a) The object instance is not known.

b) The federate is not a federation execution member.

c) RTI internal error.

10.11.6 Reference state charts

a) None

10.12 Get Dimension Handle

The *Get Dimension Handle* service shall return the dimension handle associated with the supplied dimension name.

10.12.1 Supplied arguments

a) Dimension name

10.12.2 Returned arguments

a) Dimension handle

10.12.3 Preconditions

a) The federation execution exists.

b) The federate is joined to that federation execution.

c) The specified dimension is defined in the FDD.

10.12.4 Postconditions

a) The joined federate has the requested dimension handle.

10.12.5 Exceptions

a) The dimension is not defined in the FDD.

b) The federate is not a federation execution member.

c) RTI internal error.

10.12.6 Reference state charts

a) None

10.13 Get Dimension Name

The *Get Dimension Name* service shall return the dimension name associated with the supplied dimension handle.

10.13.1 Supplied arguments

a) Dimension handle

10.13.2 Returned arguments

a) Dimension name

10.13.3 Preconditions

a) The federation execution exists.

b) The federate is joined to that federation execution.

c) The specified dimension handle exists.

10.13.4 Postconditions

a) The joined federate has the requested dimension name.

10.13.5 Exceptions

a) Invalid dimension handle.

b) The federate is not a federation execution member.

c) RTI internal error.

10.13.6 Reference state charts

a) None

10.14 Get Dimension Upper Bound

The *Get Dimension Upper Bound* service shall return the positive integer that is the upper bound for the specified dimension, as specified in the FDD.

10.14.1 Supplied arguments

a) Dimension handle

10.14.2 Returned arguments

a) Dimension upper bound

10.14.3 Preconditions

a) The federation execution exists.

b) The federate is joined to that federation execution.

c) The specified dimension handle exists.

10.14.4 Postconditions

a) The joined federate has the requested dimension upper bound.

10.14.5 Exceptions

a) Invalid dimension handle.

b) The federate is not a federation execution member.

c) RTI internal error.

10.14.6 Reference state charts

a) None

10.15 Get Available Dimensions For Class Attribute

The *Get Available Dimensions For Class Attribute* service shall return the available dimensions of the supplied class attribute and object class. If the supplied class attribute does not have available dimensions, an empty set shall be returned.

10.15.1 Supplied arguments

a) Object class handle

b) Class attribute handle

10.15.2 Returned arguments

a) A set of dimension handles

10.15.3 Preconditions

a) The federation execution exists.

b) The federate is joined to that federation execution.

c) The specified object class handle exists.

d) The specified class attribute handle exists.

e) The specified class attribute is an available attribute of the specified object class.

10.15.4 Postconditions

a) The joined federate has the dimension handle set.

10.15.5 Exceptions

a) Invalid object class handle.

b) Invalid class attribute handle.

c) The object class attribute is not an available attribute of the object class.

d) The federate is not a federation execution member.

e) RTI internal error.

10.15.6 Reference state charts

a) None

10.16 Get Known Object Class Handle

The *Get Known Object Class Handle* service shall return the known object class of the supplied object instance.

10.16.1 Supplied arguments

a) Object instance handle

10.16.2 Returned arguments

a) Object class handle

10.16.3 Preconditions

a) The federation execution exists.

b) The federate is joined to that federation execution.

c) An object instance with the specified handle exists.

d) The joined federate knows about the object instance with the specified handle.

10.16.4 Postconditions

a) The joined federate has the known object class handle of the specified object instance.

10.16.5 Exceptions

a) The object instance is not known.

b) The federate is not a federation execution member.

c) RTI internal error.

10.16.6 Reference state charts

a) None

10.17 Get Available Dimensions For Interaction Class

The *Get Available Dimensions For Interaction Class* service shall return the available dimensions of the supplied interaction class. If the supplied interaction class does not have available dimensions, an empty set shall be returned.

10.17.1 Supplied arguments

a) Interaction class handle

10.17.2 Returned arguments

a) A set of dimension handles

10.17.3 Preconditions

a) The federation execution exists.

b) The federate is joined to that federation execution.

c) The specified interaction class handle exists.

10.17.4 Postconditions

a) The joined federate has the requested dimension handle set.

10.17.5 Exceptions

a) Invalid interaction class handle.

b) The federate is not a federation execution member.

c) RTI internal error.

10.17.6 Reference state charts

a) None

10.18 Get Transportation Type

The *Get Transportation Type* service shall return the transportation type associated with the supplied transportation name, as defined in the FDD Transportation Table.

10.18.1 Supplied arguments

a) Transportation name

10.18.2 Returned arguments

a) Transportation type

10.18.3 Preconditions

a) The federation execution exists.

b) The federate is joined to that federation execution.

c) The transportation name is defined in the FDD transportation type table.

10.18.4 Postconditions

a) The joined federate has the requested transportation type.

10.18.5 Exceptions

a) Invalid transportation name.

b) The federate is not a federation execution member.

c) RTI internal error.

10.18.6 Reference state charts

a) None

10.19 Get Transportation Name

The *Get Transportation Name* service shall return the transportation name associated with the supplied transportation type, as defined in the FDD Transportation Table.

10.19.1 Supplied arguments

a) Transportation type

10.19.2 Returned arguments

a) Transportation name

10.19.3 Preconditions

a) The federation execution exists.

b) The federate is joined to that federation execution.

c) The transportation type has a corresponding name in the FDD transportation type table.

10.19.4 Postconditions

a) The joined federate has the requested transportation name.

10.19.5 Exceptions

a) Invalid transportation type.

b) The federate is not a federation execution member.

c) RTI internal error.

10.19.6 Reference state charts

a) None

10.20 Get Order Type

The *Get Order Type* service shall return the order type associated with the supplied order name, as defined in 8.1.8.

10.20.1 Supplied arguments

a) Order name

10.20.2 Returned arguments

a) Order type

10.20.3 Preconditions

a) The federation execution exists.

b) The federate is joined to that federation execution.

c) The order name is valid (see 8.1.8).

10.20.4 Postconditions

a) The joined federate has the requested order type.

10.20.5 Exceptions

a) Invalid order name.

b) The federate is not a federation execution member.

c) RTI internal error.

10.20.6 Reference state charts

a) None

10.21 Get order name

The *Get Order Name* service shall return the order name, as defined in 8.1.8, associated with the supplied order type.

10.21.1 Supplied arguments

a) Order type

10.21.2 Returned arguments

a) Order name

10.21.3 Preconditions

a) The federation execution exists.

b) The federate is joined to that federation execution.

c) The order type is valid (8.1.8).

10.21.4 Postconditions

a) The joined federate has the requested order name.

10.21.5 Exceptions

a) Invalid order type.

b) The federate is not a federation execution member.

c) RTI internal error.

10.21.6 Reference state charts

a) None

10.22 Enable Object Class Relevance Advisory Switch

The *Enable Class Relevance Advisory Switch* service shall set the Object Class Relevance Advisory Switch on.

10.22.1 Supplied arguments

a) None

10.22.2 Returned arguments

a) None

10.22.3 Preconditions

a) The federation execution exists.

b) The federate is joined to that federation execution.

c) The Object Class Relevance Advisory Switch is off.

d) Save not in progress.

e) Restore not in progress.

10.22.4 Postconditions

a) The Class Relevance Advisory Switch is turned on.

10.22.5 Exceptions

a) Object Class Relevance Advisory Switch is on.

b) The federate is not a federation execution member.

c) Save in progress.

d) Restore in progress.

e) RTI internal error.

10.22.6 Reference state charts

a) None

10.23 Disable Object Class Relevance Advisory Switch

The *Disable Class Relevance Advisory Switch* service shall set the Object Class Relevance Advisory Switch off.

10.23.1 Supplied arguments

a) None

10.23.2 Returned arguments

a) None

10.23.3 Preconditions

a) The federation execution exists.

b) The federate is joined to that federation execution.

c) The object class relevance advisory switch is on.

d) Save not in progress.

e) Restore not in progress.

10.23.4 Postconditions

a) The Class Relevance Advisory Switch is turned off.

10.23.5 Exceptions

a) Object Class Relevance Advisory Switch is off.

b) The federate is not a federation execution member.

c) Save in progress.

d) Restore in progress.

e) RTI internal error.

10.23.6 Reference state charts

a) None

10.24 Enable Attribute Relevance Advisory Switch

The *Enable Attribute Relevance Advisory Switch* service shall set the Attribute Relevance Advisory Switch on.

10.24.1 Supplied arguments

a) None

10.24.2 Returned arguments

a) None

10.24.3 Preconditions

a) The federation execution exists.

b) The federate is joined to that federation execution.

c) The Attribute Relevance Advisory Switch is off.

d) Save not in progress.

e) Restore not in progress.

10.24.4 Postconditions

a) The Attribute Relevance Advisory switch is turned on.

10.24.5 Exceptions

a) Attribute Relevance Advisory Switch is on.

b) The federate is not a federation execution member.

c) Save in progress.

d) Restore in progress.

e) RTI internal error.

10.24.6 Reference state charts

a) Figure 14: Implications of ownership of instance attribute (i, k, j)

10.25 Disable Attribute Relevance Advisory Switch

The *Disable Attribute Relevance Advisory Switch* service shall set the Attribute Relevance Advisory Switch off.

10.25.1 Supplied arguments

a) None

10.25.2 Returned arguments

a) None

10.25.3 Preconditions

a) The federation execution exists.

b) The federate is joined to that federation execution.

c) The Attribute Relevance Advisory Switch is on.

d) Save not in progress.

e) Restore not in progress.

10.25.4 Postconditions

a) The Attribute Relevance Advisory switch is turned off.

10.25.5 Exceptions

a) Attribute Relevance Advisory Switch is off.

b) The federate is not a federation execution member.

c) Save in progress.

d) Restore in progress.

e) RTI internal error.

10.25.6 Reference state charts

a) Figure 14: Implications of ownership of instance attribute (i, k, j)

10.26 Enable Attribute Scope Advisory Switch

The *Enable Attribute Scope Advisory Switch* service shall set the Attribute Scope Advisory Switch on.

10.26.1 Supplied arguments

a) None

10.26.2 Returned arguments

a) None

10.26.3 Preconditions

a) The federation execution exists.

b) The federate is joined to that federation execution.

c) The Attribute Scope Advisory Switch is off.

d) Save not in progress.

e) Restore not in progress.

10.26.4 Postconditions

a) The Attribute Scope Advisory switch is turned on.

10.26.5 Exceptions

a) Attribute Scope Advisory Switch is on.

b) The federate is not a federation execution member.

c) Save in progress.

d) Restore in progress.

e) RTI internal error.

10.26.6 Reference state charts

a) Figure 14: Implications of ownership of instance attribute (i, k, j)

10.27 Disable Attribute Scope Advisory Switch

The *Disable Attribute Scope Advisory Switch* service shall set the Attribute Scope Advisory Switch off.

10.27.1 Supplied arguments

a) None

10.27.2 Returned arguments

a) None

10.27.3 Preconditions

a) The federation execution exists.

b) The federate is joined to that federation execution.

c) The Attribute Scope Advisory Switch is on.

d) Save not in progress.

e) Restore not in progress.

10.27.4 Postconditions

a) The Attribute Scope Advisory switch is turned off.

10.27.5 Exceptions

a) Attribute Scope Advisory Switch is off.

b) The federate is not a federation execution member.

c) Save in progress.

d) Restore in progress.

e) RTI internal error.

10.27.6 Reference state charts

a) Figure 14: Implications of ownership of instance attribute (i, k, j)

10.28 Enable Interaction Relevance Advisory Switch

The *Enable Interaction Relevance Advisory Switch* service shall set the Interaction Relevance Advisory Switch on.

10.28.1 Supplied arguments

a) None

10.28.2 Returned arguments

a) None

10.28.3 Preconditions

a) The federation execution exists.

b) The federate is joined to that federation execution.

c) The Interaction Relevance Advisory Switch is off.

d) Save not in progress.

e) Restore not in progress.

10.28.4 Postconditions

a) The Interaction Relevance Advisory switch is turned on.

10.28.5 Exceptions

a) Interaction Relevance Advisory Switch is on.

b) The federate is not a federation execution member.

c) Save in progress.

d) Restore in progress.

e) RTI internal error.

10.28.6 Reference state charts

a) Figure 12: Interaction class (m)

10.29 Disable Interaction Relevance Advisory Switch

The *Disable Interaction Relevance Advisory Switch* service shall set the Interaction Relevance Advisory Switch off.

10.29.1 Supplied arguments

a) None

10.29.2 Returned arguments

a) None

10.29.3 Preconditions

a) The federation execution exists.

b) The federate is joined to that federation execution.

c) The Interaction Relevance Advisory Switch is on.

d) Save not in progress.

e) Restore not in progress.

10.29.4 Postconditions

a) The Interaction Relevance Advisory switch is turned off.

10.29.5 Exceptions

a) Interaction Relevance Advisory Switch is off.

b) The federate is not a federation execution member.

c) Save in progress.

d) Restore in progress.

e) RTI internal error.

10.29.6 Reference state charts

a) Figure 12: Interaction class (m)

10.30 Get Dimension Handle Set

The *Get Dimension Handle Set* service shall return the dimensions of the specified region.

10.30.1 Supplied arguments

a) Region handle

10.30.2 Returned arguments

a) A set of dimensions

10.30.3 Preconditions

a) The federation execution exists.

b) The federate is joined to that federation execution.

c) The region exists.

10.30.4 Postconditions

a) The joined federate has the dimensions of the region.

10.30.5 Exceptions

a) Invalid region.

b) The federate is not a federation execution member.

c) Save in progress.

d) Restore in progress.

e) RTI internal error.

10.30.6 Reference state charts

a) None

10.31 Get Range Bounds

The *Get Range Bounds* service shall return the current lower and upper bounds of the range for the specified dimension in the specified region.

10.31.1 Supplied arguments

a) Region handle

b) Dimension handle

10.31.2 Returned arguments

a) Range lower bound

b) Range upper bound

10.31.3 Preconditions

a) The federation execution exists.

b) The federate is joined to that federation execution.

c) The region exists.

d) The region contains the specified dimension.

e) Save not in progress.

f) Restore not in progress.

10.31.4 Postconditions

a) The joined federate has the requested range bounds.

10.31.5 Exceptions

a) Invalid region.

b) The region does not contain the specified dimension.

c) The federate is not a federation execution member.

d) Save in progress.

e) Restore in progress.

f) RTI internal error.

10.31.6 Reference state charts

a) None

10.32 Set Range Bounds

The *Set Range Bounds* service shall set the current lower and upper bounds of the range for the specified dimension in the specified region.

10.32.1 Supplied arguments

a) Region handle

b) Dimension handle

c) Range lower bound

d) Range upper bound

10.32.2 Returned arguments

a) None

10.32.3 Preconditions

a) The federation execution exists.

b) The federate is joined to that federation execution.

c) The region exists.

d) The region was created by the joined federate using the *Create Region* service.

e) The region contains the specified dimension.

f) The range lower bound is greater than or equal to zero.

g) The range upper bound is less than or equal to the specified dimension’s upper bound.

h) The range lower bound is less than the range upper bound.

i) Save not in progress.

j) Restore not in progress.

10.32.4 Postconditions

a) The specified range is set to the specified values.

10.32.5 Exceptions

a) Invalid region

b) The region was not created by the joined federate using the *Create Region* service.

c) The region does not contain the specified dimension.

d) Invalid range bound.

e) The federate is not a federation execution member.

f) Save in progress.

g) Restore in progress.

h) RTI internal error.

10.32.6 Reference state charts

a) None

10.33 Normalize Federate Handle

The *Normalize Federate Handle* service shall return a normalized value of the provided federate handle that can be used in constructing a region with the dimension Federates.

10.33.1 Supplied arguments

a) Federate handle

10.33.2 Returned arguments

a) Normalized value

10.33.3 Preconditions

a) The federation execution exists.

b) The invoking federate is joined to that federation execution.

c) The specified federate is joined to that federation execution.

10.33.4 Postconditions

a) The normalized value corresponds to the provided federate handle.

10.33.5 Exceptions

a) Invalid federate handle.

b) The federate is not a federation execution member.

c) RTI internal error.

10.33.6 Reference state charts

a) None

10.34 Normalize Service Group

The *Normalize Service Group* service shall return a normalized value of the provided service group indicator that can be used in constructing a region with the dimension ServiceGroups.

10.34.1 Supplied arguments

a) Service group indicator

10.34.2 Returned arguments

a) Normalized value

10.34.3 Preconditions

a) The federation execution exists.

b) The invoking federate is joined to that federation execution.

10.34.4 Postconditions

a) The normalized value corresponds to the provided service group indicator.

10.34.5 Exceptions

a) Invalid service group.

b) The federate is not a federation execution member.

c) RTI internal error.

10.34.6 Reference state charts

a) None

10.35 Initialize RTI

The *Initialize RTI* service shall be used to initialize the RTI and shall be invoked before any other service. The RTI implementation shall use the set of strings argument for initialization and return those strings that are not recognized by the RTI implementation.

10.35.1 Supplied arguments

a) Set of strings

10.35.2 Returned arguments

a) Set of strings containing those strings that were in the supplied argument and unused by initialization

10.35.3 Preconditions

a) The *Initialize RTI* service was not previously invoked, or the *Finalize RTI* service was previously invoked.

10.35.4 Postconditions

a) The RTI is ready for other service invocations.

10.35.5 Exceptions

a) Initialize previously invoked

b) Bad initialization parameter

c) RTI internal error

10.35.6 Reference state charts

a) None

10.36 Finalize RTI

The *Finalize RTI* service shall be used to terminate service invocations with the RTI. Upon return from this service, the RTI shall no longer accept service invocations other than *Initialize RTI*.

10.36.1 Supplied arguments

a) None

10.36.2 Returned arguments

a) None

10.36.3 Preconditions

a) The *Initialize RTI* service was previously invoked.

b) The federate is not joined to any federation execution.

10.36.4 Postconditions

a) The RTI shall no longer accept service invocations other than *Initialize RTI.*

10.36.5 Exceptions

a) Initialize never invoked.

b) Some federate is joined to an execution.

c) RTI internal error.

10.36.6 Reference state charts

a) None

10.37 Evoke Callback

The *Evoke Callback* service instructs the RTI that the invoking federate is prepared to receive a single federate callback. If there are no pending callbacks to be delivered to the federate, this service invocation shall wait for the duration indicated by the supplied argument. The supplied argument shall have a precision of at least 1 ms. This service shall result in either the timeout expiring or a single callback to be invoked on the federate. The pending callback indicator argument shall be a Boolean value: TRUE shall indicate that pending callbacks exist, and FALSE shall indicate that no pending callbacks exist.

10.37.1 Supplied arguments

a) Minimum amount of wall-clock time the service will wait for a callback

10.37.2 Returned arguments

a) Pending callback indicator

10.37.3 Preconditions

a) The federation execution exists.

b) The federate is joined to that federation execution.

10.37.4 Postconditions

a) The RTI is provided the opportunity to invoke callbacks on the federate.

10.37.5 Exceptions

a) The federate is not a federation execution member.

b) RTI internal error.

10.37.6 Reference state charts

a) None

10.38 Evoke Multiple Callbacks

The *Evoke Multiple Callbacks* service instructs the RTI that the invoking federate is prepared to receive multiple federate callbacks. The service shall continue to process available callbacks until the minimum specified wall-clock time. At that wall-clock time, if there are no additional callbacks to be delivered to the federate, the service shall complete. If after the minimum specified wall-clock time there continues to be callbacks, the RTI shall continue to deliver those callbacks until the maximum specified wall-clock time is exceeded. Both supplied wall-clock time arguments shall have a precision of at least one millisecond. The pending callback indicator argument shall be a boolean value: TRUE shall indicate that pending callbacks exist and FALSE shall indicate that no pending callbacks exist.

10.38.1 Supplied arguments

a) Minimum amount of wall-clock time

b) Maximum amount of wall-clock time

10.38.2 Returned arguments

a) Pending callback indicator

10.38.3 Preconditions

a) The federation execution exists.

b) The federate is joined to that federation execution.

10.38.4 Postconditions

a) The RTI is provided the opportunity to invoke callbacks on the federate.

10.38.5 Exceptions

a) The federate is not a federation execution member.

b) RTI internal error.

10.38.6 Reference state charts

a) None

10.39 Enable Callbacks

The *Enable Callbacks* service instructs the RTI to deliver any available callbacks processed through the *Evoke Callback* and *Evoke Multiple Callbacks* services. The default state for a federate is that callbacks are enabled.

10.39.1 Supplied arguments

a) None

10.39.2 Returned arguments

a) None

10.39.3 Preconditions

a) The federation execution exists.

b) The federate is joined to that federation execution.

c) Save not in progress.

d) Restore not in progress.

10.39.4 Postconditions

a) The RTI will process any pending callbacks on the federate when either the *Evoke Callback* or *Evoke Multiple Callbacks* service is invoked.

10.39.5 Exceptions

a) The federate is not a federation execution member.

b) Save in progress.

c) Restore in progress.

d) RTI internal error.

10.39.6 Reference state charts

a) None

10.40 Disable Callbacks

The *Disable Callbacks* service instructs the RTI to defer the delivery of any available callbacks. This service can be used to implement a guard mechanism to control when callbacks are delivered to the federate.

10.40.1 Supplied arguments

a) None

10.40.2 Returned arguments

a) None

10.40.3 Preconditions

a) The federation execution exists.

b) The federate is joined to that federation execution.

c) Save not in progress.

d) Restore not in progress.

10.40.4 Postconditions

a) The RTI will not process any pending callbacks on the federate for any existing or subsequent *Evoke Callback* or *Evoke Multiple Callbacks* invocations until the *Enable Callback* service is invoked.

10.40.5 Exceptions

a) The federate is not a federation execution member.

b) Save in progress.

c) Restore in progress.

d) RTI internal error.

10.40.6 Reference state charts

a) None

11. Management object model (MOM)

11.1 Overview

The management object model (MOM) provides facilities for access to RTI operating information during federation execution. These MOM facilities can be used by joined federates to gain insight into the operations of the federation execution and to control the functioning of the RTI, the federation execution, and individual joined federates. The ability to monitor and control elements of a federation is required for proper functioning of a federation execution.

Access and interchange of RTI information elements included in the MOM shall be accomplished by utilizing predefined HLA constructs, objects, and interactions in the same way that participating federates exchange information with other federates. The MOM shall use the OMT format (IEEE P1516.2) and syntax for the definition of these information and control elements. The RTI publishes object classes, registers object instances, and updates values of attributes of those object instances; subscribes to and receives some interactions; and publishes and sends other interaction. A joined federate charged with controlling a federation execution can subscribe to some or all of the object classes, reflect the updates, publish and send some interactions, and subscribe to and receive other interactions. Thus, while a FOM for a federation shall include all elements of the MOM, a federation may choose to use all, part, or none of the MOM standard classes and associated information elements. The MOM shall satisfy these requirements by utilizing predefined HLA constructs: object instances and interactions. The RTI shall publish object classes and shall register object instances and update values of attributes of those object instances; shall subscribe to and receive some interaction classes; and shall publish and send other interaction classes. A joined federate charged with controlling a federation execution can subscribe to the object classes, reflect the updates, publish and send some interaction classes, and subscribe to and receive other interaction classes.

All MOM object classes, interaction classes, attributes, and parameters shall be predefined in the FDD. These definitions may not be revised. MOM definitions may be extended, however; they may be augmented with additional subclasses, class attributes, or parameters. These new elements shall not be acted upon directly by the RTI; they may be acted on by federates in the federation.

11.2 MOM object classes

The high-level MOM object class structure is depicted in Figure 18. The MOM object classes shall be defined as follows:

a) Object class *HLAmanager.HLAfederate*: contains attributes that describe the state of a federate. The RTI shall publish the class and register one object instance of this class for each federate in the federation. The RTI shall update the information periodically, based on timing data provided in *HLAmanager.HLAfederate.HLAadjust* interactions. Information shall be contained in an object instance that includes identifying information about the federate, measures of the federate's state, and the status of the queued messages maintained by the RTI for the federate.

b) Object class *HLAmanager.HLAfederation*: contains attributes that describe the state of the federation execution. The RTI shall publish the class and register one object instance of this class for the federation.

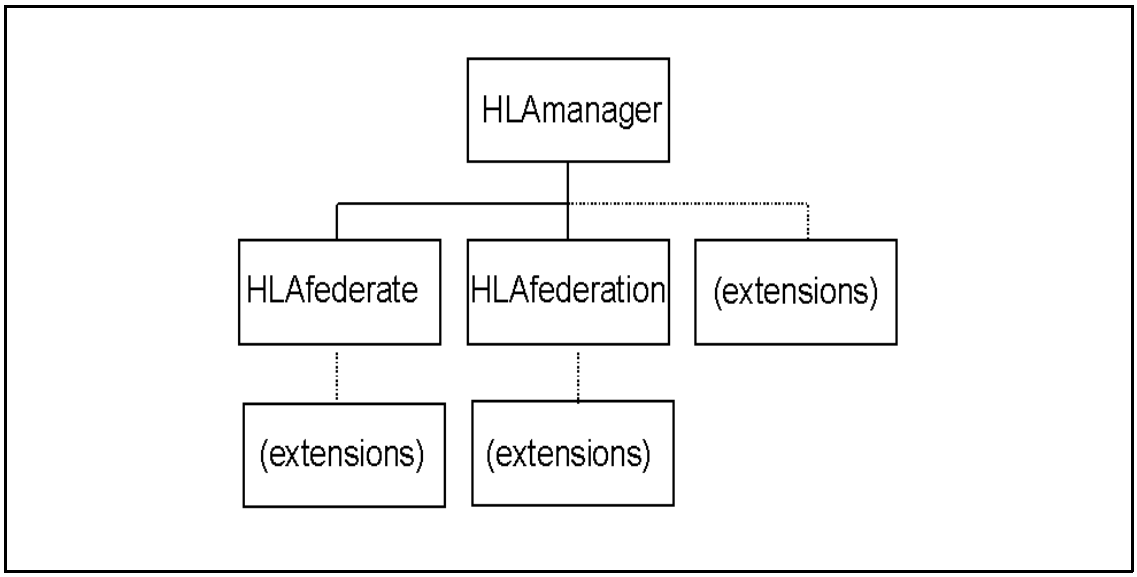

Figure 18—MOM object class structure

11.2.1 Extension of MOM object classes

The MOM object classes may be extended by adding subclasses or class attributes. Without extensions, the RTI shall publish *HLAmanager.HLAfederate* and *HLAmanager.HLAfederation* classes with predefined MOM class attributes, register an instance, and update the values of the predefined instance attributes.

The RTI shall not subscribe to any object class. Valid methods for extending the MOM object classes shall be as follows:

a) Subclasses may be added to any MOM object class. Here, the joined federate may publish the object class and its attributes, register an instance of the new class, and update values of instance attributes of the object instance according to the dictates of the federation execution. Note that the instance of the subclass shall be separate from the MOM object instance that is registered by the RTI. Therefore, instance attributes that are inherited by the extension subclass from the MOM predefined class shall not be updated by the RTI.

b) Attributes may be added to any MOM object class. Here, the joined federate may publish the object class with the new class attributes, and may subscribe to the object class and attributes in it, may discover and reflect updates to learn the object instance in question, and may update the values of the new instance attributes (after acquiring ownership, of course) using the discovered object instance designator. Note that the instance that the joined federate shall update with the new instance attributes shall be the same as the MOM object instance that is registered by the RTI. The RTI shall never acquire ownership of any new class attributes.

The RTI shall not divest ownership of MOM instance attributes that are defined in this major clause. The RTI shall respond to the invocation, by any federate, of the *Request Attribute Value Update* service for any MOM object class or instance attribute that is defined in this major clause by supplying values via the normal instance attribute update mechanism.

11.3 MOM interaction classes

The high-level MOM interaction class structure is depicted in Figure 19. The MOM interaction classes shall be defined as:

a) Interaction classes that are subclasses of *HLAmanager.HLAfederate.HLAadjust* and *HLAmanager.Hlafederation.HLAadjust* shall be acted on by the RTI. They shall permit a managing federate to adjust the way the RTI performs when responding to another federate and how it shall respond and report to the managing federate.

b) Interaction classes that are subclasses of *HLAmanager.HLAfederate.HLArequest* shall be acted on by the RTI. They shall cause the RTI to send subclasses of *HLAmanager.HLAfederate.HLAreport* interaction class.

c) Interaction classes that are subclasses of *HLAmanager.HLAfederate.HLAreport* shall be sent by the RTI. They shall respond to interaction classes that are subclasses of *HLAmanager.HLAfederate.HLArequest* class interactions. They shall describe some aspect of the federate, such as its object class subscription tree.

d) Interaction classes that are subclasses of *HLAmanager.HLAfederate.HLAservice* shall be acted on by the RTI. They shall invoke HLA services on behalf of another federate. These interactions shall cause the RTI to react as if the service was invoked by that other federate (for example, a managing federate could change the time-regulating state of another federate).

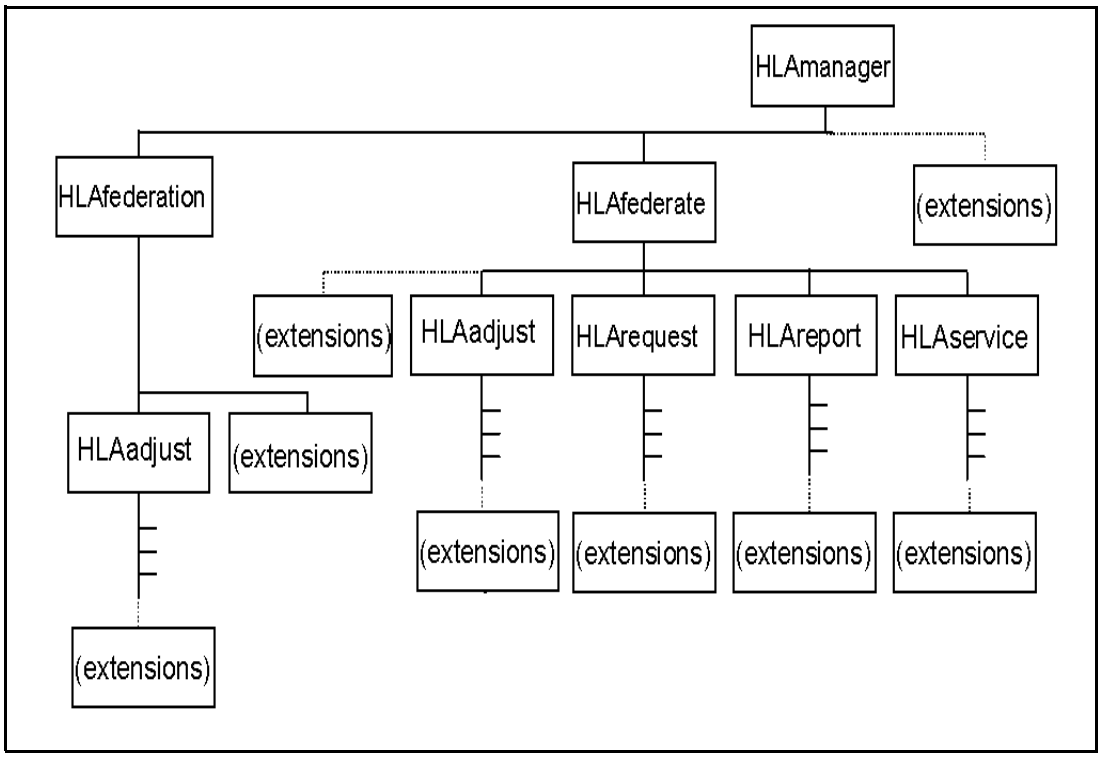

Figure 19—MOM interaction class structure

11.3.1 Extension of MOM interaction classes

The MOM interaction classes may be extended by adding subclasses or parameters. There shall be three categories of extension of MOM interaction classes as follows:

a) Classes of interaction that the RTI shall send (subclasses of *HLAmanager.HLAfederate.HLAreport*). The RTI shall publish at the MOM leaf-class level (e.g., *HLAmanager.HLAfederate.HLAreport.HLAreportException*). It shall send interactions containing all predefined parameters for that interaction class. Valid methods for extending this type of MOM interaction class shall be as follows:

   1) Subclasses may be added to these MOM interaction classes. The RTI shall not send interactions of these subclasses. If joined federates subscribe to the subclass, they shall receive the full interaction. If they subscribe to the class of which the extension is a subclass, the interaction shall be promoted to the subscribed class and any new parameters shall be lost.

   2) Parameters may be added to any MOM interaction class. Interactions of these classes that are sent by the RTI shall not contain the new parameters.

b) Classes of interaction that the RTI shall receive (subclasses of *HLAmanager.HLAfederate.HLAadjust*, *HLAmanager.HLAfederate.HLArequest*, *HLAmanager.HLAfederate.HLAservice,* and *HLAmanager.HLAfederation.HLAadjust*). The RTI shall subscribe at the MOM leaf-class level (e.g., *HLAmanager.HLAfederate.HLAadjust.HLAsetTiming*). It shall receive these interactions and process all predefined parameters for that interaction class (with exceptions as listed in the MOM Parameters Definition Table). Valid methods for extending this type of MOM interaction class shall be as follows:

   1) Subclasses may be added to any MOM interaction class. If a joined federate sends an interaction of this class, the RTI shall receive a promoted version that shall contain only the parameters of the predefined interaction class.

   2) Parameters may be added to any MOM interaction class. If a joined federate sends an interaction with extra parameters, the RTI shall receive the new parameters, but shall ignore them and process only the predefined parameters.

c) Classes of interaction that shall be neither sent nor received by the RTI. These classes of interaction shall be ignored by the RTI and may be formed in any way that is consistent with IEEE Std 1516.2-2000 FOM development.

11.4 MOM-related characteristics of the RTI

11.4.1 Normal RTI MOM administration

The RTI deals with MOM object classes and interaction classes in the following ways:

a) The RTI shall publish all leaf object classes in Table 4 that are designated with a “P.”

b) For each leaf object class in Table 4, the RTI shall publish all attributes in Table 6 for that object class that are designated with a “P,” and no more.

c) The RTI shall publish all leaf interaction classes in Table 5 that are designated with a “P.”

d) When sending an interactions of one of the leaf classes in Table 5, the RTI shall always supply all parameters listed in Table 7 for that interaction class, and no more.

e) The RTI shall be actively subscribed to all leaf interaction classes in Table 5 that are designated with an "S."

f) The RTI shall be neither time-regulating, nor time-constrained and shall never provide time stamps or user-supplied tags with instance attribute updates, sent interactions, or deletion of object instances.

g) The RTI shall update the values of all attributes of instances of HLAmanager.HLAfederate found in Table 4 using an update region set that contains a single region. The region used with each instance attribute shall contain only the Federate dimension as a specified dimension. The range for that dimension shall be a point range around the normalized value (using the *Normalize Federate Handle* service) of the HLAfederateHandle attribute of that object instance.

h) The RTI shall send all interactions of class HLAmanager.HLAfederate.HLAreport.HLAreportServiceInvocation using an update region set that contains a single region. The region used with each such interaction shall contain only the Federate and ServiceGroup dimensions as specified dimensions. The range used for the Federate dimension shall be a point range around the normalized value (using the *Normalize Federate Handle* service) of the HLAfederate parameter of that interaction. The range used for the ServiceGroup dimension shall be a point range around the normalized value (using the *Normalize Service Group* service) of the service group in which the service indicated in the HLAservice parameter is found.

i) The RTI shall send all other interactions that are of subclasses of HLAmanager.HLAfederate.HLAreport in Table 5 using an update region set that contains a single region. The region used with each such interaction shall contain only the Federate dimension as a specified dimension. The range for that dimension shall be a point range around the normalized value (using the *Normalize Federate Handle* service) of the HLAfederate parameter of that interaction.

j) Other than registering and deleting object instances, updating instance attributes, and sending and receiving interactions, the RTI shall not engage in any other activity with regard to MOM. This means that among other things, an RTI shall never

   1) Engage in ownership transfer of MOM instance attributes.

   2) Modify the transportation or order type of a MOM instance attribute or interaction class.

   3) Use DDM services with MOM instance attributes or interaction classes, except as noted above.

11.4.2 Sending MOM interactions by federates during save/restore

Though sending interactions is normally forbidden while saves and restores are in progress, the following deviations to this rule shall be allowed (assuming the correct publications are in place):

a) A joined federate shall be allowed to send a MOM HLAresignFederationExecution interaction when either a save or a restore is in progress.

b) A joined federate shall be allowed to send either a MOM HLAfederateSaveBegun or a MOM HLAfederateSaveComplete interaction when a save is in progress.

c) A joined federate shall be allowed to send a MOM HLAfederateRestoreComplete interaction when a restore is in progress.

11.5 Service-reporting

The *HLAmanager.HLAfederate.HLAreport.HLAreportServiceInvocation* interaction shall be sent by the RTI whenever an HLA service is invoked, either by a particular joined federate or by the RTI at a particular joined federate, and Service-Reporting is Enabled for that particular joined federate. The Service-Reporting value for a specific joined federate shall be determined in the following manner:

— If no *HLAmanager.HLAfederate.HLAadjust.HLAsetServiceReporting* interaction has set the *HLAreportingState* parameter HLAtrue for that joined federate, Service-Reporting for that joined federate shall be Enabled if the Service Reporting Switch in the federation's FDD is set to Enabled; otherwise, Service-Reporting for that joined federate shall be Disabled.

— If one or more *HLAmanager.HLAfederate.HLAadjust.HLAsetServiceReporting* interactions have set the *HLAreportingState* parameter for that joined federate, Service-Reporting for that joined federate shall be Enabled if the last *HLAmanager.HLAfederate.HLAadjust.HLAsetServiceReporting* interaction set the *HLAreportingState* parameter for that joined federate to HLAtrue; otherwise, Service-Reporting for that joined federate shall be Disabled.

If a joined federate is subscribed to the *HLAmanager.HLAfederate.HLAreport.HLAreportServiceInvocation* interaction, all attempts to enable Service-Reporting for that joined federate by sending an *HLAmanager.HLAfederate.HLAadjust.HLAsetServiceReporting* interaction with an *HLAreportingState* parameter value of HLAtrue shall fail and be reported via the normal MOM interaction failure means. If a joined federate has Service-Reporting enabled, that joined federate shall not be able to subscribe to the *HLAmanager.HLAfederate.HLAreport.HLAreportServiceInvocation* interaction class.

11.6 MOM OMT tables

The elements that comprise MOM are depicted in OMT format (IEEE Std 1516.2-2000) in Table 4 through Table 17. The MOM table information in this clause shall appear in all compliant FOMs (IEEE Std 1516-2000 and IEEE Std 1516.2-2000) and shall not be altered. Note that MOM does not use many of the OMT tables (e.g., time representation table, user-supplied tag table, synchronization table, transportation type table, switches table, and variant record table). They have been omitted from the depiction.

Table 4—MOM object class structure table

HLAobjectRoot (N)	HLAmanager (N)	HLAfederate (P)
		HLAfederation (P)

Table 5—MOM interaction class structure table

HLAinteractionRoot (N)	HLAmanager (N)	HLAfederate (N)	HLAadjust (N)	HLAsetTiming (S)
				HLAmodifyAttributeState (S)
				HLAsetServiceReporting (S)
				HLAsetExceptionReporting (S)
			HLArequest (N)	HLArequestPublications (S)
				HLArequestSubscriptions (S)
				HLArequestObjectInstancesThatCanBeDeleted (S)
				HLArequestObjectInstancesUpdated (S)
				HLArequestObjectInstancesReflected (S)
				HLArequestUpdatesSent (S)
				HLArequestInteractionsSent (S)
				HLArequestReflectionsReceived (S)
				HLArequestInteractionsReceived (S)
				HLArequestObjectInstanceInformation (S)
				HLArequestSynchronizationPoints (S)
				HLArequestSynchronizationPointStatus (S)
			HLAreport (N)	HLAreportObjectClassPublication (P)
				HLAreportObjectClassSubscription (P)
				HLAreportInteractionPublication (P)
				HLAreportInteractionSubscription (P)
				HLAreportObjectInstancesThatCanBeDeleted (P)
				HLAreportObjectInstancesUpdated (P)
				HLAreportObjectInstancesReflected (P)
				HLAreportUpdatesSent (P)
				HLAreportReflectionsReceived (P)
				HLAreportInteractionsSent (P)
				HLAreportInteractionsReceived (P)
				HLAreportObjectInstanceInformation (P)
				HLAreportException (P)
				HLAreportServiceInvocation (P)
				HLAreportMOMexception (P)
				HLAreportSynchronizationPoints (P)
				HLAreportSynchronizationPointStatus (P)
			HLAservice (N)	HLAresignFederationExecution (S)
				HLAsynchronizationPointAchieved (S)
				HLAfederateSaveBegun (S)
				HLAfederateSaveComplete (S)
				HLAfederateRestoreComplete (S)
				HLApublishObjectClassAttributes (S)
				HLAunpublishObjectClassAttributes (S)

Table 5—MOM interaction class structure table *(continued)*

HLAinteractionRoot (N)	HLAmanager (N)	HLAfederate (N)	HLAservice (N)	HLApublishInteractionClass (S)
				HLAunpublishInteractionClass (S)
				HLAsubscribeObjectClassAttributes (S)
				HLAunsubscribeObjectClassAttributes (S)
				HLAsubscribeInteractionClass (S)
				HLAunsubscribeInteractionClass (S)
				HLAdeleteObjectInstance (S)
				HLAlocalDeleteObjectInstance (S)
				HLAchangeAttributeTransportationType (S)
				HLAchangeInteractionTransportationType (S)
				HLAunconditionalAttributeOwershipDivestiture (S)
				HLAenableTimeRegulation (S)
				HLAdisableTimeRegulation (S)
				HLAenableTimeConstrained (S)
				HLAdisableTimeConstrained (S)
				HLAtimeAdvanceRequest (S)
				HLAtimeAdvanceRequestAvailable (S)
				HLAnextMessageRequest (S)
				HLAnextMessageRequestAvailable (S)
				HLAflushQueueRequest (S)
				HLAenableAsynchronousDelivery (S)
				HLAdisableAsynchronousDelivery (S)
				HLAmodifyLookahead (S)
				HLAchangeAttributeOrderType (S)
				HLAchangeInteractionOrderType (S)
		HLAfederation (N)	HLAadjust (N)	HLAsetSwitches (S)

Table 6—MOM attribute table

Object	Attribute	Datatype	Update type	Update condition	T/A	P/S	Available dimensions	Transportation	Order
HLAobjectRoot. HLAmanager.HLAfederate	HLAfederateHandle	HLAhandle	Static	NA	N	P	Federate	HLAreliable	Receive
	HLAfederateType	HLAunicodeString	Static	NA	N	P	Federate	HLAreliable	Receive
	HLAfederateHost	HLAunicodeString	Static	NA	N	P	Federate	HLAreliable	Receive
	HLARTIversion	HLAunicodeString	Static	NA	N	P	Federate	HLAreliable	Receive
	HLAFDDID	HLAunicodeString	Static	NA	N	P	Federate	HLAreliable	Receive
	HLAtimeConstrained	HLAboolean	Conditional	Service invocation	N	P	Federate	HLAreliable	Receive
	HLAtimeRegulating	HLAboolean	Conditional	Service invocation	N	P	Federate	HLAreliable	Receive
	HLAasynchronousDelivery	HLAboolean	Conditional	Service invocation	N	P	Federate	HLAreliable	Receive
	HLAfederateState	HLAfederateState	Conditional	Service invocation	N	P	Federate	HLAreliable	Receive
	HLAtimeManagerState	HLAtimeState	Conditional	Service invocation	N	P	Federate	HLAreliable	Receive
	HLAlogicalTime	HLAlogicalTime	Periodic	HLAsetTiming. HLAreportPeriod	N	P	Federate	HLAreliable	Receive
	HLAlookahead	HLAtimeInterval	Periodic	HLAsetTiming. HLAreportPeriod	N	P	Federate	HLAreliable	Receive
	HLAGALT	HLAlogicalTime	Periodic	HLAsetTiming. HLAreportPeriod	N	P	Federate	HLAreliable	Receive
	HLALITS	HLAlogicalTime	Periodic	HLAsetTiming. HLAreportPeriod	N	P	Federate	HLAreliable	Receive

Table 6—MOM attribute table *(continued)*

Object	Attribute	Datatype	Update type	Update condition	T/A	P/S	Available dimensions	Transportation	Order
	HLAROlength	HLAcount	Periodic	HLAsetTiming. HLAreportPeriod	N	P	Federate	HLAreliable	Receive
	HLATSOlength	HLAcount	Periodic	HLAsetTiming. HLAreportPeriod	N	P	Federate	HLAreliable	Receive
	HLAreflectionsReceived	HLAcount	Periodic	HLAsetTiming. HLAreportPeriod	N	P	Federate	HLAreliable	Receive
	HLAupdatesSent	HLAcount	Periodic	HLAsetTiming. HLAreportPeriod	N	P	Federate	HLAreliable	Receive
	HLAinteractionsReceived	HLAcount	Periodic	HLAsetTiming. HLAreportPeriod	N	P	Federate	HLAreliable	Receive
	HLAinteractionsSent	HLAcount	Periodic	HLAsetTiming. HLAreportPeriod	N	P	Federate	HLAreliable	Receive
	HLAobjectInstancesThatCanBeDeleted	HLAcount	Periodic	HLAsetTiming. HLAreportPeriod	N	P	Federate	HLAreliable	Receive
	HLAobjectInstancesUpdated	HLAcount	Periodic	HLAsetTiming. HLAreportPeriod	N	P	Federate	HLAreliable	Receive
	HLAobjectInstancesReflected	HLAcount	Periodic	HLAsetTiming. HLAreportPeriod	N	P	Federate	HLAreliable	Receive
	HLAobjectInstancesDeleted	HLAcount	Periodic	HLAsetTiming. HLAreportPeriod	N	P	Federate	HLAreliable	Receive
	HLAobjectInstancesRemoved	HLAcount	Periodic	HLAsetTiming. HLAreportPeriod	N	P	Federate	HLAreliable	Receive
	HLAobjectInstancesRegistered	HLAcount	Periodic	HLAsetTiming. HLAreportPeriod	N	P	Federate	HLAreliable	Receive
	HLAobjectInstancesDiscovered	HLAcount	Periodic	HLAsetTiming. HLAreportPeriod	N	P	Federate	HLAreliable	Receive
	HLAtimeGrantedTime	HLAmsec	Periodic	HLAsetTiming. HLAreportPeriod	N	P	Federate	HLAreliable	Receive
	HLAtimeAdvancingTime	HLAmsec	Periodic	HLAsetTiming. HLAreportPeriod	N	P	Federate	HLAreliable	Receive
HLAmanager.HLAfederation	HLAfederationName	HLAunicodeString	Static	NA	N	P	NA	HLAreliable	Receive
	HLAfederatesInFederation	HLAhandleList	Conditional	Federate joins or resigns	N	P	NA	HLAreliable	Receive
	HLARTIversion	HLAunicodeString	Static	NA	N	P	NA	HLAreliable	Receive

Table 6—MOM attribute table *(continued)*

Object	Attribute	Datatype	Update type	Update condition	T/A	P/S	Available dimensions	Transportation	Order
	HLAFDDID	HLAunicodeString	Static	NA	N	P	NA	HLAreliable	Receive
	HLAlastSaveName	HLAunicodeString	Conditional	service invocation	N	P	NA	HLAreliable	Receive
	HLAlastSaveTime	HLAlogicalTime	Conditional	service invocation	N	P	NA	HLAreliable	Receive
	HLAnextSaveName	HLAunicodeString	Conditional	service invocation	N	P	NA	HLAreliable	Receive
	HLAnextSaveTime	HLAlogicalTime	Conditional	service invocation	N	P	NA	HLAreliable	Receive
	HLAautoProvide	HLAswitch	Conditional	MOM interaction	N	P	NA	HLAreliable	Receive
	HLAconveyRegionDesignatorSets	HLAswitch	Conditional	MOM interaction	N	P	NA	HLAreliable	Receive

Table 7—MOM parameter table

Interaction		Parameter	Datatype	Available dimensions	Transportation	Order
HLAinteractionRoot	HLAmanager	NA	NA	NA	HLAreliable	Receive
HLAmanager	HLAfederate	HLAfederate	HLAhandle	NA	HLAreliable	Receive
HLAmanager.HLAfederate	HLAadjust	NA	NA	NA	HLAreliable	Receive
HLAmanager.HLAfederate.HLAadjust	HLAsetTiming	HLAreportPeriod	HLAseconds	NA	HLAreliable	Receive
	HLAmodifyAttributeState	HLAobjectInstance	HLAhandle	NA	HLAreliable	Receive
		HLAattribute	HLAhandle			
		HLAattributeState	HLAownership			
	HLAsetServiceReporting	HLAreportingState	HLAboolean	NA	HLAreliable	Receive
	HLAsetExceptionReporting	HLAreportingState	HLAboolean	NA	HLAreliable	Receive
HLAmanager.HLAfederate	HLArequest	NA	NA	NA	HLAreliable	Receive
HLAmanager.HLAfederate.HLArequest	HLArequestPublications	NA	NA	NA	HLAreliable	Receive
	HLArequestSubscriptions	NA	NA	NA	HLAreliable	Receive
	HLArequestObjectInstancesThatCanBeDeleted	NA	NA	NA	HLAreliable	Receive
	HLArequestObjectInstancesUpdated	NA	NA	NA	HLAreliable	Receive
	HLArequestObjectInstancesReflected	NA	NA	NA	HLAreliable	Receive
	HLArequestUpdatesSent	NA	NA	NA	HLAreliable	Receive
	HLArequestInteractionsSent	NA	NA	NA	HLAreliable	Receive
	HLArequestReflectionsReceived	NA	NA	NA	HLAreliable	Receive
	HLArequestInteractionsReceived	NA	NA	NA	HLAreliable	Receive
	HLArequestObjectInstanceInformation	HLAobjectInstance	HLAhandle	NA	HLAreliable	Receive
	HLArequestSynchronizationPoints	NA	NA	NA	HLAreliable	Receive
	HLArequestSynchronizationPointStatus	NA	NA	NA	HLAreliable	Receive
HLAmanager.HLAfederate	HLAreport	NA	NA	NA	HLAreliable	Receive

Table 7—MOM parameter table *(continued)*

Interaction		Parameter	Datatype	Available dimensions	Transportation	Order
HLAmanager.HLAfederate.HLAreport	HLAreportObjectClassPublication	HLAnumberOfClasses	HLAcount	Federate	HLAreliable	Receive
		HLAobjectClass	HLAhandle			
		HLAattributeList	HLAhandleList			
	HLAreportObjectClassSubscription	HLAnumberOfClasses	HLAcount	Federate	HLAreliable	Receive
		HLAobjectClass	HLAhandle			
		HLAactive	HLAboolean			
		HLAattributeList	HLAhandleList			
	HLAreportInteractionPublication	HLAinteractionClassList	HLAhandleList	Federate	HLAreliable	Receive
	HLAreportInteractionSubscription	HLAinteractionClassList	HLAinteractionSubList	Federate	HLAreliable	Receive
	HLAreportObjectInstancesThatCanBeDeleted	HLAobjectInstanceCounts	HLAobjectClassBasedCounts	Federate	HLAreliable	Receive
	HLAreportObjectInstancesUpdated	HLAobjectInstanceCounts	HLAobjectClassBasedCounts	Federate	HLAreliable	Receive
	HLAreportObjectInstancesReflected	HLAobjectInstanceCounts	HLAobjectClassBasedCounts	Federate	HLAreliable	Receive
	HLAreportUpdatesSent	HLAtransportation	HLAtransportationName	Federate	HLAreliable	Receive
		HLAupdateCounts	HLAobjectClassBasedCounts			
	HLAreportReflectionsReceived	HLAtransportation	HLAtransportationName	Federate	HLAreliable	Receive
		HLAreflectCounts	HLAobjectClassBasedCounts			
	HLAreportInteractionsSent	HLAtransportation	HLAtransportationName	Federate	HLAreliable	Receive
		HLAinteractionCounts	HLAinteractionCounts	Federate	HLAreliable	Receive
	HLAreportInteractionsReceived	HLAtransportation	HLAtransportationName	Federate	HLAreliable	Receive
		HLAinteractionCounts	HLAinteractionCounts			
	HLAreportObjectInstanceInformation	HLAobjectInstance	HLAhandle	Federate	HLAreliable	Receive
		HLAownedInstanceAttributeList	HLAhandleList			
		HLAregisteredClass	HLAhandle			
		HLAknownClass	HLAhandle			
	HLAreportException	HLAservice	HLAunicodeString	Federate	HLAreliable	Receive
		HLAexception	HLAunicodeString			

Table 7—MOM parameter table *(continued)*

Interaction		Parameter	Datatype	Available dimensions	Transportation	Order
HLAmanager.HLAfederate.HLAreport	HLAreportServiceInvocation	HLAservice	HLAunicodeString	Federate ServiceGroup	HLAreliable	Receive
		HLAsuccessIndicator	HLAboolean			
		HLAsuppliedArguments	HLAargumentList			
		HLAreturnedArguments	HLAargumentList			
		HLAexception	HLAunicodeString			
		HLAserialNumber	HLAcount			
	HLAreportMOMexception	HLAservice	HLAunicodeString	Federate	HLAreliable	Receive
		HLAexception	HLAunicodeString			
		HLAparameterError	HLAboolean			
	HLAreportSynchronizationPoints	HLAsyncPoints	HLAsyncPointList	Federate	HLAreliable	Receive
	HLAreportSynchronizationPointStatus	HLAsyncPointName	HLAunicodeString	Federate	HLAreliable	Receive
		HLAsyncPointFederates	HLAsyncPointFederateList			
HLAmanager.HLAfederate	HLAservice	NA	NA	NA	HLAreliable	Receive
HLAmanager.HLAfederate.HLAservice	HLAresignFederationExecution	HLAresignAction	HLAresignAction	NA	HLAreliable	Receive
	HLAsynchronizationPointAchieved	HLAlabel	HLAunicodeString	NA	HLAreliable	Receive
	HLAfederateSaveBegun	NA	NA	NA	HLAreliable	Receive

Table 7—MOM parameter table *(continued)*

Interaction		Parameter	Datatype	Available dimensions	Transportation	Order
HLAmanager.HLAfederate.HLAservice	HLAfederateSaveComplete	HLAsuccessIndicator	HLAboolean	NA	HLAreliable	Receive
	HLAfederateRestoreComplete	HLAsuccessIndicator	HLAboolean	NA	HLAreliable	Receive
	HLApublishObjectClassAttributes	HLAobjectClass	HLAhandle	NA	HLAreliable	Receive
		HLAattributeList	HLAhandleList			
	HLAunpublishObjectClassAttributes	HLAobjectClass	HLAhandle	NA	HLAreliable	Receive
		HLAattributeList	HLAhandleList			
	HLApublishInteractionClass	HLAinteractionClass	HLAhandle	NA	HLAreliable	Receive
	HLAunpublishInteractionClass	HLAinteractionClass	HLAhandle	NA	HLAreliable	Receive
	HLAsubscribeObjectClassAttributes	HLAobjectClass	HLAhandle	NA	HLAreliable	Receive
		HLAattributeList	HLAhandleList			
		HLAactive	HLAboolean			
	HLAunsubscribeObjectClassAttributes	HLAobjectClass	HLAhandle	NA	HLAreliable	Receive
		HLAattributeList	HLAhandleList			
	HLAsubscribeInteractionClass	HLAinteractionClass	HLAhandle	NA	HLAreliable	Receive
		HLAactive	HLAboolean			
	HLAunsubscribeInteractionClass	HLAinteractionClass	HLAhandle	NA	HLAreliable	Receive
	HLAdeleteObjectInstance	HLAobjectInstance	HLAhandle	NA	HLAreliable	Receive
		HLAtag	HLAopaqueData			
		HLAtimeStamp	HLAlogicalTime			
	HLAlocalDeleteObjectInstance	HLAobjectInstance	HLAhandle	NA	HLAreliable	Receive
	HLAchangeAttributeTransportationType	HLAobjectInstance	HLAhandle	NA	HLAreliable	Receive
		HLAattributeList	HLAhandleList			
		HLAtransportation	HLAtransportationName			
	HLAchangeInteractionTransportationType	HLAinteractionClass	HLAhandle	NA	HLAreliable	Receive
		HLAtransportation	HLAtransportationName			

Table 7—MOM parameter table *(continued)*

Interaction		Parameter	Datatype	Available dimensions	Transportation	Order
HLAmanager.HLAfederate.HLAservice	HLAunconditionalAttributeOwnershipDivestiture	HLAobjectInstance	HLAhandle	NA	HLAreliable	Receive
		HLAattributeList	HLAhandleList			
	HLAenableTimeRegulation	HLAlookahead	HLAtimeInterval	NA	HLAreliable	Receive
	HLAdisableTimeRegulation	NA	NA	NA	HLAreliable	Receive
	HLAenableTimeConstrained	NA	NA	NA	HLAreliable	Receive
	HLAdisableTimeConstrained	NA	NA	NA	HLAreliable	Receive
	HLAtimeAdvanceRequest	HLAtimeStamp	HLAlogicalTime	NA	HLAreliable	Receive
	HLAtimeAdvanceRequestAvailable	HLAtimeStamp	HLAlogicalTime	NA	HLAreliable	Receive
	HLAnextMessageRequest	HLAtimeStamp	HLAlogicalTime	NA	HLAreliable	Receive
	HLAnextMessageRequestAvailable	HLAtimeStamp	HLAlogicalTime	NA	HLAreliable	Receive
	HLAflushQueueRequest	HLAtimeStamp	HLAlogicalTime	NA	HLAreliable	Receive
	HLAenableAsynchronousDelivery	NA	NA	NA	HLAreliable	Receive
	HLAdisableAsynchronousDelivery	NA	NA	NA	HLAreliable	Receive
	HLAmodifyLookahead	HLAlookahead	HLAtimeInterval	NA	HLAreliable	Receive
	HLAchangeAttributeOrderType	HLAobjectInstance	HLAhandle	NA	HLAreliable	Receive
		HLAattributeList	HLAhandleList			
		HLAsendOrder	HLAorderType			
	HLAchangeInteractionOrderType	HLAinteractionClass	HLAhandle	NA	HLAreliable	Receive
		HLAsendOrder	HLAorderType			
HLAmanager	HLAfederation	NA	NA	NA	HLAreliable	Receive
HLAmanager.HLAfederation	HLAadjust	NA	NA	NA	HLAreliable	Receive
HLAmanager.HLAfederation.HLAadjust	HLAsetSwitches	HLAautoProvide	HLAswitch	NA	HLAreliable	Receive
		HLAconveyRegionDesignatorSets	HLAswitch	NA	HLAreliable	Receive

Table 8—MOM dimension table

Name	Datatype	Dimension upper bound	Normalization function	Value when unspecified
Federate	HLAfederateHandle	—	*Normalize Federate Handle* service	Excluded
ServiceGroup	HLAserviceGroupName	7	*Normalize Service Group* service	Excluded
NOTE—The value of the Dimension upper bound entry for the Federate dimension is RTI implementation dependent.				

Table 9—MOM basic data representation table

Name	Size in bits	Interpretation	Endian	Encoding
—				
HLAinteger32BE	32	Integer in the range [-2^{31}, $2^{31}-1$].	Big	32-bit two's complement signed integer. The most significant bit contains the sign.
—				
HLAoctetPairBE	16	16-bit value.	Big	Assumed to be portable among devices.
—				
HLAoctet	8	8-bit value.	Big	Assumed to be portable among hardware devices.
—				
NOTE—This does not include the complete set of predefined basic data representations from the OMT Specification (IEEE Std 1516.2-2000). Only the basic data representations needed for the MOM are present here.				

Table 10—MOM simple datatype table

Name	Representation	Units	Resolution	Accuracy	Semantics
—					
HLAunicodeChar	HLAoctetPairBE	NA	NA	NA	Unicode UTF-16 character (see the Unicode Standard, version 3.0).
HLAbyte	HLAoctet	NA	NA	NA	Uninterpreted 8-bit byte.
HLAcount	HLAinteger32BE	NA	NA	NA	NA
HLAseconds	HLAinteger32BE	s	NA	NA	NA
HLAmsec	HLAinteger32BE	ms	NA	NA	NA
HLAfederateHandle	HLAinteger32BE	NA	NA	NA	The type of the argument to the *Normalize Federate Handle* service. This is a pointer to a RTI defined programming language object, NOT an integer 32.
NOTE—This does not include the complete set of predefined simple datatypes from the OMT Specification (IEEE Std 1516.2-2000). Only the simple datatypes needed for the MOM are present here.					

Table 11—MOM enumerated datatype table

Name	Representation	Enumerator	Values	Semantics
HLAboolean	HLAinteger32BE	HLAfalse	0	Standard boolean type
		HLAtrue	1	
HLAfederateState	HLAinteger32BE	ActiveFederate	1	State of the federate
		FederateSaveInProgress	3	
		FederateRestoreInProgress	5	
HLAtimeState	HLAinteger32BE	TimeGranted	0	State of time advancement
		TimeAdvancing	1	
HLAownership	HLAinteger32BE	Unowned	0	NA
		Owned	1	
HLAresignAction	HLAinteger32BE	DivestOwnership	1	Action to be performed by RTI in conjunction with resignation
		DeleteObjectInstances	2	
		CancelPendingAcquisitions	3	
		DeleteObjectInstancesThenDivestOwnership	4	
		CancelPendingAcquisitionsThenDeleteObjectInstancesThenDivestOwnership	5	
		NoAction	6	
HLAorderType	HLAinteger32BE	Receive	0	Order type to be used for sending attributes or interactions
		TimeStamp	1	
HLAswitch	HLAinteger32BE	Enabled	1	NA
		Disabled	0	
HLAsyncPointStatus	HLAinteger32BE	NoActivity	0	Joined federate synchronization point status
		AttemptingToRegisterSyncPoint	1	
		MovingToSyncPoint	2	
		WaitingForRestOfFederation	3	
HLAserviceGroupName	HLAinteger32BE	FederationManagement	0	Service group identifier
		DeclarationManagement	1	
		ObjectManagement	2	
		OwnershipManagement	3	
		TimeManagement	4	
		DataDistributionManagement	5	
		SupportServices	6	

Table 12—MOM array datatype table

Name	Element Type	Cardinality	Encoding	Semantics
—				
HLAunicodeString	HLAunicodeChar	Dynamic	HLAvariableArray	Unicode string representation.
HLAopaqueData	HLAbyte	Dynamic	HLAvariableArray	Uninterpreted sequence of bytes.
HLAhandle	HLAbyte	Dynamic	HLAvariableArray	Encoded value of a handle. The encoding is based on the type of handle.
HLAtransportationName	HLAunicodeChar	Dynamic	HLAvariableArray	String whose legal value shall be name from any row in the OMT Transportation Table (IEEE 1516.2-2000).
HLAlogicalTime	HLAbyte	Dynamic	HLAvariableArray	An encoded logical time. An empty array shall indicate that the value is not defined.
HLAtimeInterval	HLAbyte	Dynamic	HLAvariableArray	An encoded logical time interval. An empty array shall indicate that the value is not defined.
HLAhandleList	HLAhandle	Dynamic	HLAvariableArray	List of encoded handles.
HLAinteractionSubList	HLAinteractionSubscription	Dynamic	HLAvariableArray	List of interaction subscription indicators.
HLAargumentList	HLAunicodeString	Dynamic	HLAvariableArray	List of arguments.
HLAobjectClassBasedCounts	HLAobjectClassBasedCount	Dynamic	HLAvariableArray	List of counts of various items based on object class.
HLAinteractionCounts	HLAinteractionCount	Dynamic	HLAvariableArray	List of interaction counts.
HLAsyncPointList	HLAunicodeString	Dynamic	HLAvariableArray	List of names of synchronization points.
HLAsyncPointFederateList	HLAsyncPointFederate	Dynamic	HLAvariableArray	List of joined federates and the synchronization status of each.
NOTE—This does not include the complete set of predefined array datatypes from the OMT Specification (IEEE Std 1516.2-2000). Only the array datatypes needed for the MOM are present here.				

Table 13—MOM fixed record datatype table

Record name	Field			Encoding	Semantics
	Name	Type	Semantics		
HLAinteractionSubscription	HLAinteractionClass	HLAhandle	Encoded interaction class handle	HLAfixedRecord	Interaction subscription information.
	HLAactive	HLAboolean	Whether subscription is active (HLAtrue = active)		
HLAobjectClassBasedCount	HLAobjectClass	HLAhandle	Encoded object class handle	HLAfixedRecord	Object class and count of associated items.
	HLAcount	HLAcount	Number of items		
HLAinteractionCount	HLAinteractionClass	HLAhandle	Encoded interaction class handle	HLAfixedRecord	Count of interactions of a class.
	HLAinteractionCount	HLAcount	Number of interactions		
HLAsyncPointFederate	HLAfederate	HLAhandle	Encoded joined federate handle	HLAfixedRecord	A particular joined federate and its synchronization point status.
	HLAfederateSyncStatus	HLAsyncPointStatus	Synchronization status of the particular joined federate		

Table 14—MOM object class definitions table

Object	Definition
HLAmanager	This object class is the root class of all MOM object classes.
HLAmanager.HLAfederation	This object class shall contain RTI state variables relating to a federation execution. The RTI shall publish it and shall register one object instance for the federation execution. It shall not automatically update the values of the instance attributes; a joined federate shall use a *Request Attribute Value Update* service to obtain values for the instance attributes.
HLmanager.HLAfederate	This object class shall contain RTI state variables relating to a joined federate. The RTI shall publish it and shall register one object instance for each joined federate in a federation. Dynamic attributes that shall be contained in an object instance shall be updated periodically, where the period should be determined by an interaction of the class *HLAmanager.HLAfederate.HLAadjust.HLAsetTiming*. If this value is never set or is set to zero, no periodic update shall be performed by the RTI.

Table 15—MOM interaction class definitions table

Interaction		Definition
HLAmanager		Root class of MOM interactions.
HLAmanager	HLAfederate	Root class of MOM interactions that deal with a specific joined federate.
HLAmanager.HLAfederate	HLAadjust	Permit a joined federate to adjust the RTI state variables associated with another joined federate.
	HLArequest	Permit a joined federate to request RTI data about another joined federate.
	HLAreport	Report RTI data about a joined federate. The RTI shall send these interactions in response to interactions of class *HLAmanager.HLAfederate.HLArequest* that correspond to services that are normally invoked by federates.
	HLAservice	The interaction class shall be acted upon by the RTI. These interactions shall invoke HLA services on behalf of another joined federate. They shall cause the RTI to react as if the service has been invoked by that other joined federate. If exceptions arise as a result of the use of these interactions, they shall be reported via the *HLAmanager.HLAfederate.HLAreport.HLAreportMOMexception* interaction to all joined federates that subscribe to this interaction. There are two ways an error can occur: The sending federate does not provide all the required arguments as parameters, or the preconditions of the spoofed service are not met. Each type of error is reported via *HLAmanager.HLAfederate.HLAreport.HLAreportMOMexception*. NOTE—These interactions shall have the potential to disrupt normal federation execution and should be used with great care.
HLAmanager.HLAfederate.HLAadjust	HLAsetTiming	Adjust the period between updates of the *HLAmanager.HLAfederate* object instance for the specified joined federate. If this interaction is never sent, the RTI shall not perform periodic updates.

Table 15—MOM interaction class definitions table *(continued)*

Interaction		Definition
HLAmanager.HLAfederate.HLAadjust	HLAmodifyAttributeState	Modify the ownership state of an attribute of an object instance for the specified joined federate. If the interaction is used to give ownership of the instance attribute to the specified joined federate and another joined federate currently owns the instance attribute, the owning joined federate shall be divested of ownership of the instance attribute before ownership is granted to the specified joined federate. No notification of change of ownership of the instance attribute shall be provided to either joined federate. In order for ownership of the instance attribute to be granted to the specified joined federate, the following conditions shall be true: – The specified joined federate knows about the object instance. – The specified joined federate is publishing the corresponding class attribute at the known class of the specified object instance at that joined federate. – The specified instance attribute is not owned by the RTI (i.e., it is not a predefined attribute of a MOM object class). If one or more of the above conditions are not met, the interaction shall have no effect and an error shall be reported via an interaction of class *HLAmanager.HLAfederate.HLAreport.HLAreportMOMexception*.
	HLAsetServiceReporting	Specify whether to report service invocations to or from the specified joined federate via *HLAmanager.HLAfederate.HLAreport.HLAreportServiceInvocation* interactions (enable or disable Service-Reporting). If the specified joined federate is subscribed to the *HLAmanager.HLAfederate.HLAreport.HLAreportServiceInvocation* interaction, all attempts to enable Service-Reporting for that joined federate by sending an *HLAmanager.HLAfederate.HLAadjust.HLAsetServiceReporting* interaction with an *HLAreportingState* parameter value of HLAtrue shall fail and be reported via the normal MOM interaction failure means.
	HLAsetExceptionReporting	Specify whether the RTI shall report service invocation exceptions via *HLAmanager.HLAfederate.HLAreport.HLAreportException* interactions.
HLAmanager.HLAfederate.HLArequest	HLArequestPublications	Request that the RTI send report interactions that contain the publication data of a joined federate. It shall result in one interaction of class *HLAmanager.HLAfederate.HLAreport.HLAreportInteractionPublication* and one interaction of class *HLAmanager.HLAfederate.HLAreport.HLAreportObjectClassPublication* for each object class published.
	HLArequestSubscriptions	Request that the RTI send report interactions that contain the subscription data of a joined federate. It shall result in one interaction of class *HLAmanager.HLAfederate.HLAreport.HLAreportInteractionSubscription* and one interaction of class *HLAmanager.HLAfederate.HLAreport.HLAreportObjectClassSubscription* for each object class published.

Table 15—MOM interaction class definitions table *(continued)*

Interaction		Definition
HLAmanager.HLAfederate.HLArequest	HLArequestObjectInstancesThatCanBeDeleted	Request that the RTI send a report interaction that contains the object instances that can be deleted at a joined federate. It shall result in one interaction of class *HLAmanager.HLAfederate.HLAreport.HLAreportObjectInstancesThatCanBeDeleted*.
	HLArequestObjectInstancesUpdated	Request that the RTI send a report interaction that contains the object instance updating responsibility of a joined federate. It shall result in one interaction of class *HLAmanager.HLAfederate.HLAreport.HLAreportObjectInstancesUpdated*.
	HLArequestObjectInstancesReflected	Request that the RTI send a report interaction that contains the object instances for which a joined federate has had a *Reflect Attribute Values †* service invocation. It shall result in one interaction of class *HLAmanager.HLAfederate.HLAreport.HLAreportObjectInstancesReflected*.
	HLArequestUpdatesSent	Request that the RTI send a report interaction that contains the number of updates that a joined federate has generated. It shall result in one interaction of class *HLAmanager.HLAfederate.HLAreport.HLAreportUpdatesSent* for each transportation type that is used to send updates.
	HLArequestInteractionsSent	Request that the RTI send a report interaction that contains the number of interactions that a joined federate has generated. This count shall include interactions sent with region. It shall result in one interaction of class *HLAmanager.HLAfederate.HLAreport.HLAreportInteractionsSent* for each transportation type that is used to send interactions.
	HLArequestReflectionsReceived	Request that the RTI send a report interaction that contains the number of reflections that a joined federate has received. It shall result in one interaction of class *HLAmanager.HLAfederate.HLAreport.HLAreportReflectionsReceived* for each transportation type used in receiving reflections.
	HLArequestInteractionsReceived	Request that the RTI send a report interaction that contains the number of interactions that a joined federate has received. It shall result in one interaction of class *HLAmanager.HLAfederate.HLAreport.HLAreportInteractionsReceived* for each transportation type used in receiving interactions.
	HLArequestObjectInstanceInformation	Request that the RTI send a report interaction that contains the information that a joined federate maintains on a single object instance. It shall result in one interaction of class *HLAmanager.HLAfederate.HLAreport.HLAreportObjectInstanceInformation*.
	HLArequestSynchronizationPoints	Request that the RTI send a report interaction that contains a list of all in progress federation synchronization points. It shall result in one interaction of class *HLAmanager.HLAfederate.HLAreport.HLAreportSynchronizationPoints*.
	HLArequestSynchronizationPointStatus	Request that the RTI send a report interaction that contains a list that includes each federate (and its synchronization status) that is associated with a particular synchronization point. It shall result in one interaction of class *HLAmanager.HLAfederate.HLAreport.HLAreportSynchronizationPointStatus*.

Table 15—MOM interaction class definitions table *(continued)*

Interaction		Definition
HLAmanager.HLAfederate.HLAreport	HLAreportObjectClassPublication	The interaction shall be sent by the RTI in response to an interaction of class *HLAmanager.HLAfederate.HLArequest.HLArequestPublications*. It shall report the attributes of one object class published by the joined federate. One of these interactions shall be sent for each object class containing attributes that are published by the joined federate.
	HLAreportInteractionPublication	The interaction shall be sent by the RTI in response to an interaction of class *HLAmanager.HLAfederate.HLAfequest.HLArequestPublications*. It shall report the interaction classes published by the joined federate.
	HLAreportObjectClassSubscription	The interaction shall be sent by the RTI in response to an interaction of class *HLAmanager.HLAfederate.HLArequest.HLArequestSubscriptions*. It shall report the attributes of one object class subscribed to by the joined federate. One of these interactions shall be sent for each object class that is subscribed to by the joined federate. This information shall reflect related DDM usage.
	HLAreportInteractionSubscription	The interaction shall be sent by the RTI in response to an interaction of class *HLAmanager.HLAfederate.HLArequest.HLArequestSubscriptions*. It shall report the interaction classes subscribed to by the joined federate. This information shall reflect related DDM usage.
	HLAreportObjectInstancesThatCanBeDeleted	The interaction shall be sent by the RTI in response to an interaction of class *HLAmanager.HLAfederate.HLArequest.HLArequestObjectInstancesThatCanBeDeleted*. It shall report the number of object instances (by registered class of the object instances) whose **HLAprivilegeToDeleteObject** attributes are owned by the joined federate.
	HLAreportObjectInstancesUpdated	The interaction shall be sent by the RTI in response to an interaction of class *HLAmanager.HLAfederate.HLArequest.HLArequestObjectInstancesUpdated*. It shall report the number of object instances (by registered class of the object instances) for which the joined federate has invoked the *Update Attribute Values* service.
	HLAreportObjectInstancesReflected	The interaction shall be sent by the RTI in response to an interaction of class *HLAmanager.HLAfederate.HLArequest.HLArequestObjectInstancesReflected*. It shall report the number of object instances (by registered class of the object instances) for which the joined federate has had a *Reflect Attribute Values †* service invocation.
	HLAreportUpdatesSent	The interaction shall be sent by the RTI in response to an interaction of class *HLAmanager.HLAfederate.HLArequest.HLArequestUpdatesSent*. It shall report the number of updates sent (by registered class of the object instances of the updates) by the joined federate since the beginning of the federation execution. One interaction of this class shall be sent by the RTI for each transportation type used.
	HLAreportReflectionsReceived	The interaction shall be sent by the RTI in response to an interaction of class *HLAmanager.HLAfederate.HLArequest.HLArequestReflectionsReceived*. It shall report the number of reflections received (by registered class of the object instances of the reflects) by the joined federate since the beginning of the federation execution. One interaction of this class shall be sent by the RTI for each transportation type used.

Table 15—MOM interaction class definitions table *(continued)*

Interaction		Definition
HLAmanager.HLAfederate.HLAreport	HLAreportInteractionsSent	The interaction shall be sent by the RTI in response to an interaction of class *HLAmanager.HLAfederate.HLArequest.HLArequestInteractionsSent*. It shall report the number of interactions sent (by sent class of the interactions) by the joined federate since the beginning of the federation execution. This count shall include interactions sent with region. One interaction of this class shall be sent by the RTI for each transportation type used.
	HLAreportInteractionsReceived	The interaction shall be sent by the RTI in response to an interaction of class *HLAmanager.HLAfederate.HLArequest.HLArequestInteractions Received*. It shall report the number of interactions received (by sent class of the interactions) by the joined federate since the beginning of the federation execution. One interaction of this class shall be sent by the RTI for each transportation type used.
	HLAreportObjectInstanceInformation	The interaction shall be sent by the RTI in response to an interaction of class *HLAmanager.HLAfederate.HLArequest.HLArequestObjectInstanceInformation*. It shall report on a single object instance and portray the instance attributes of that object instance that are owned by the joined federate, the registered class of the object instance, and the known class of the object instance at that joined federate.
	HLAreportException	The interaction shall be sent by the RTI when an exception occurs as the result of a service invocation at the indicated joined federate. This interaction shall be sent only if the last *HLAmanager.HLAfederate.HLAadjust.HLAsetExceptionReporting* interaction changing the *HLAreportingState* parameter sets the parameter to HLAtrue, for the indicated joined federate.
	HLAreportServiceInvocation	This interaction shall be sent by the RTI whenever an HLA service is invoked, either by the indicated joined federate or by the RTI at the indicated joined federate, and Service-Reporting is Enabled for the indicated joined federate. This interaction shall always contain the arguments supplied by the service invoker. If the service invocation was successful, the interaction also shall contain the value returned to the invoker (if the service returns a value); otherwise, the interaction also shall contain an indication of the exception that was raised to the invoker.
	HLAreportMOMexception	The interaction shall be sent by the RTI when one the following occurs: a MOM interaction without all the necessary parameters is sent or an interaction that imitates a federate's invocation of an HLA service is sent and not all of the service's pre-conditions are met.
	HLAreportSynchronizationPoints	The interaction shall be sent by the RTI in response to an interaction of class *HLAmanager.HLAfederate.HLArequest.HLArequestSynchronizationPoints*. It shall report the list of active synchronization points in the federation execution.
	HLAreportSynchronizationPointStatus	The interaction shall be sent by the RTI in response to an interaction of class *HLAmanager.HLAfederate.HLArequest.HLArequestSynchronizationPointStatus*. It shall report the status of a particular synchronization point. This shall be a list that includes each federate (and its synchronization status) that is associated with the particular synchronization point.

Table 15—MOM interaction class definitions table *(continued)*

Interaction		Definition
HLAmanager.HLAfederate.HLAservice	HLAresignFederationExecution	Cause the joined federate to resign from the federation execution. A joined federate shall be able to send this interaction anytime.
	HLAsynchronizationPointAchieved	Mimic the joined federate's report of achieving a synchronization point.
	HLAfederateSaveBegun	Mimic the joined federate's report of starting a save.
	HLAfederateSaveComplete	Mimic the joined federate's report of completion of a save. A joined federate shall be able to send this interaction during a federation save.
	HLAfederateRestoreComplete	Mimic the joined federate's report of completion of a restore. A joined federate shall be able to send this interaction during a federation restore.
	HLApublishObjectClassAttributes	Set the joined federate's publication status of attributes of an object class.
	HLAunpublishObjectClassAttributes	Cause the joined federate no longer to publish attributes of an object class.
	HLApublishInteractionClass	Set the joined federate's publication status of an interaction class.
	HLAunpublishInteractionClass	Cause the joined federate no longer to publish an interaction class.
	HLAsubscribeObjectClassAttributes	Set the joined federate's subscription status of attributes of an object class.
	HLAunsubscribeObjectClassAttributes	Cause the joined federate no longer to subscribe to attributes of an object class.
	HLAsubscribeInteractionClass	Set the joined federate's subscription status to an interaction class.
	HLAunsubscribeInteractionClass	Cause the joined federate no longer to subscribe to an interaction class.
	HLAdeleteObjectInstance	Cause an object instance to be deleted from the federation.
	HLAlocalDeleteObjectInstance	Inform the RTI that it shall treat the specified object instance as if the joined federate did not know about the object instance.
	HLAchangeAttributeTransportationType	Change the transportation type used by the joined federate when sending attributes belonging to the object instance.
	HLAchangeInteractionTransportationType	Change the transportation type used by the joined federate when sending a class of interaction.
	HLAunconditionalAttributeOwnershipDivestiture	Cause the ownership of attributes of an object instance to be unconditionally divested by the joined federate.
	HLAenableTimeRegulation	Cause the joined federate to begin regulating the logical time of other joined federates.
	HLAdisableTimeRegulation	Cause the joined federate to cease regulating the logical time of other joined federates.
	HLAenableTimeConstrained	Cause the logical time of the joined federate to begin being constrained by the logical times of other joined federates.
	HLAdisableTimeConstrained	Cause the logical time of the joined federate to cease being constrained by the logical times of other joined federates.

Table 15—MOM interaction class definitions table *(continued)*

Interaction		Definition
HLAmanager.HLAfederate.HLAservice	HLAtimeAdvanceRequest	Request an advance of the joined federate's logical time on behalf of the joined federate, and release zero or more messages for delivery to the joined federate.
	HLAtimeAdvanceRequestAvailable	Request an advance of the joined federate's logical time, on behalf of the joined federate, and release zero or more messages for delivery to the joined federate.
	HLAnextMessageRequest	Request the logical time of the joined federate to be advanced to the time stamp of the next TSO message that shall be delivered to the joined federate, provided that the message shall have a time stamp no greater than the logical time specified in the request, and release zero or more messages for delivery to the joined federate.
	HLAnextMessageRequestAvailable	Request the logical time of the joined federate to be advanced to the time stamp of the next TSO message that shall be delivered to the joined federate, provided that the message shall have a time stamp no greater than the logical time specified in the request, and release zero or more messages for delivery to the joined federate.
	HLAflushQueueRequest	Request the logical time of the joined federate to be advanced as far as possible, provided that the time stamp is less than or equal to the logical time specified in the request. All TSO and RO messages shall be delivered to the joined federate.
	HLAenableAsynchronousDelivery	Cause the RTI to deliver RO messages to the joined federate at any time, even if the joined federate is time-constrained.
	HLAdisableAsynchronousDelivery	When the joined federate is time-constrained, cause the RTI to deliver RO messages to the joined federate only when its time manager state is "Time Advancing."
	HLAmodifyLookahead	Change the lookahead value used by the joined federate.
	HLAchangeAttributeOrderType	Change the order type used by the joined federate when sending attributes belonging to the object instance.
	HLAchangeInteractionOrderType	Change the order type used by the joined federate when sending a class of interaction.
HLAmanager	HLAfederation	Root class of MOM interactions that deal with a specific federation execution.
HLAmanager.HLAfederation	HLAadjust	Permit a joined federate to adjust the RTI state variables associated with a federation execution.
HLAmanager.HLAfederation.HLAadjust	HLAsetSwitches	Set the values of several HLA switches.

Table 16—MOM attribute definitions table

Class	Attribute	Definition
HLAmanager.HLAfederate	HLAfederateHandle	Handle of the joined federate returned by a *JoinFederationExecution* service invocation.
	HLAfederateType	Type of the joined federate specified by the joined federate when it joined the federation.
	HLAfederateHost	Host name of the computer on which the joined federate is executing.
	HLARTIversion	Version of the RTI software being used.
	HLAFDDID	Identifier associated with the FDD used in the joined federate.
	HLAtimeConstrained	Whether the time advancement of the joined federate is constrained by other joined federates.
	HLAtimeRegulating	Whether the joined federate influences the time advancement of other joined federates.
	HLAasynchronousDelivery	Whether the RTI shall deliver RO messages to the joined federate while the joined federate's time manager state is not "Time Advancing" (only matters if the joined federate is time-constrained).
	HLAfederateState	State of the joined federate.
	HLAtimeManagerState	State of the joined federate's time manager.
	HLAlogicalTime	Joined federate's logical time. Initial value of this information is initial value of time of the TRADT.
	HLAlookahead	Minimum duration into the future that a TSO message will be scheduled. The value shall not be defined if the joined federate is not time-regulating.
	HLAGALT	Joined federate's GALT. The value shall not be defined if GALT is not defined for the joined federate.
	HLALITS	Joined federate's LITS. The value shall not be defined if LITS is not defined for the joined federate.
	HLAROlength	Number of RO messages queued for delivery to the joined federate.
	HLATSOlength	Number of TSO messages queued for delivery to the joined federate.

Table 16—MOM attribute definitions table *(continued)*

Class	Attribute	Definition
HLAmanager.HLAfederate	HLAreflectionsReceived	Total number of reflections received by the joined federate.
	HLAupdatesSent	Total number of updates sent by the joined federate.
	HLAinteractionsReceived	Total number of interactions received by the joined federate.
	HLAinteractionsSent	Total number of interactions sent by the joined federate. This information shall reflect related DDM usage.
	HLAobjectInstancesThatCanBedeleted	Total number of object instances whose **HLAprivilegeToDeleteObject** attribute is owned by the joined federate.
	HLAobjectInstancesUpdated	Total number of object instances for which the joined federate has invoked the *Update Attribute Values* service.
	HLAobjectInstancesReflected	Total number of object instances for which the joined federate has had a *Reflect Attribute Values †* service invocation.
	HLAobjectInstancesDeleted	Total number of times the *Delete Object Instance* service was invoked by the joined federate since the federate joined the federation.
	HLAobjectInstancesRemoved	Total number of times the *Remove Object Instance †* service was invoked for the joined federate since the federate joined the federation.
	HLAobjectInstancesRegistered	Total number of times the *Register Object Instance* service or *Register Object Instance with Regions* was invoked by the joined federate since the federate joined the federation.
	HLAobjectInstancesDiscovered	Total number of times the *Discover Object Instance †* service was invoked for the joined federate since the federate joined the federation.
	HLAtimeGrantedTime	Wall-clock time duration that the federate has spent in the Time Granted state since the last update of this attribute.
	HLAtimeAdvancingTime	Wall-clock time duration that the federate has spent in the Time Advancing state since the last update of this attribute.
HLAmanager.HLAfederation	HLAfederationName	Name of the federation to which the joined federate belongs.
	HLAfederatesinFederation	Identifiers of joined federates that are joined to the federation.
	HLARTIversion	Version of the RTI software.

Table 16—MOM attribute definitions table *(continued)*

Class	Attribute	Definition
HLAmanager.HLAfederation	HLAFDDID	Identifier associated with the FDD used in the relevant *Create Federation Execution* service invocation.
	HLAlastSaveName	Name associated with the last federation state save (null if no saves have occurred).
	HLAlastSaveTime	Logical time at which the last federation state timed save occurred. The value shall not be defined if no timed saves have occurred.
	HLAnextSaveName	Name associated with the next federation state save (null if no saves are scheduled).
	HLAnextSaveTime	Logical time at which the next federation state timed save is scheduled. The value shall not be defined if no timed saves are scheduled.
	HLAautoProvide	Value of federation-wide Auto-Provide Switch. Updated when value of switch changes.
	HLAconveyRegionDesignatorSets	Value of federation-wide Convey Region Designator Sets Switch. Updated when value of switch changes.

Table 17—MOM parameter definitions table

Class		Parameter	Definition
HLAmanager	HLAfederate	HLAfederate	Handle of the joined federate that was provided when joining.
HLAmanager.HLAfederate.HLAadjust	HLAsetTiming	HLAreportPeriod	Number of seconds between updates of instance attribute values of the *HLAfederate* object instance. (A zero value causes periodic updates to cease.)
	HLAmodifyAttributeState	HLAobjectInstance	Handle of the object instance whose attribute state is being changed.
		HLAattribute	Handle of the instance attribute whose state is being changed.
		HLAattributeState	New state for the attribute of the object instance.
	HLAsetServiceReporting	HLAreportingState	Whether the RTI should report service invocations (default = HLAfalse).
	HLAsetExceptionReporting	HLAreportingState	Whether the RTI should report exceptions (default = HLAfalse).
HLAmanager.HLAfederate.HLArequest	HLArequestObjectInstanceInformation	HLAobjectInstance	Handle of the object instance for which information is being requested.
HLAmanager.HLAfederate.HLAreport	HLAreportobjectClassPublication	HLAnumberOfClasses	The number of object classes for which the joined federate publishes attributes.
		HLAobjectClass	The object class whose publication is being reported.
		HLAattributeList	List of handles of HLAobjectClass attributes that the joined federate is publishing.
	HLAreportInteractionPublication	HLAinteractionClassList	List of interaction classes that the joined federate is publishing.
	HLAreportobjectClassSubscription	HLAnumberOfClasses	The number of object classes for which the joined federate subscribes to attributes. This information shall reflect related DDM usage.
		HLAobjectClass	The object class whose subscription is being reported.
		HLAactive	Whether the subscription is active.
		HLAattributeList	List of handles of class attributes to which the joined federate is subscribing.

Table 17—MOM parameter definitions table *(continued)*

Class		Parameter	Definition
HLAmanager.HLAfederate.HLAreport	HLAreportInteractionSubscription	HLAinteractionClassList	List of interaction class/subscription type pairs. Each pair consists of the handle of an interaction class that the joined federate is subscribed to and whether the joined federate is actively subscribing. This information shall reflect related DDM usage.
	HLAreportObjectInstancesThatCanBeDeleted	HLAobjectInstanceCounts	A list of object instance counts. Each object instance count consists of an object class handle and the number of object instances of that class.
	HLAreportObjectInstancesUpdated	HLAobjectInstanceCounts	List of object instance counts. Each object instance count consists of an object class handle and the number of object instances of that class.
	HLAreportobjectInstancesReflected	HLAobjectInstanceCounts	List of object instance counts. Each object instance count consists of an object class handle and the number of object instances of that class.
	HLAreportUpdatesSent	HLAtransportation	Transportation type used in sending updates.
		HLAupdateCounts	List of update counts. Each update count consists of an object class handle and the number of updates sent of that class.
	HLAreportReflectionsReceived	HLAtransportation	Transportation type used in receiving reflections.
		HLAreflectCounts	List of reflection counts. Each reflection count consists of an object class handle and the number of reflections received of that class.
	HLAreportInteractionsSent	HLAtransportation	Transportation type used in sending interactions.
		HLAinteractionCounts	List of interaction counts. Each interaction count consists of an interaction class handle and the number of interactions of that class.This information shall reflect related DDM usage.
	HLAreportInteractionsReceived	HLAtransportation	Transportation type used in receiving interactions.
		HLAinteractionCounts	List of interaction counts. Each interaction count consists of an interaction class handle and the number of interactions of that class.

Table 17—MOM parameter definitions table *(continued)*

Class		Parameter	Definition
HLAmanager.HLAfederate.HLAreport	HLAreportobjectInstanceInformation	HLAobjectInstance	Handle of the object instance for which the interaction was sent.
		HLAownedInstanceAttributeList	List of the handles of all instance attributes, of the object instance, owned by the joined federate.
		HLAregisteredClass	Handle of the registered class of the object instance.
		HLAknownClass	Handle of the known class of the object instance at the joined federate.
	HLAreportException	HLAservice	Name of the service that raised the exception.
		HLAexception	Textual depiction of the exception.
	HLAreportServiceInvocation	HLAservice	Textual name of the service.
		HLAsuccessIndicator	Whether the service invocation was successful. Exception values are returned along with a HLAfalse value.
		HLAsuppliedArguments	Textual depiction of the arguments supplied in the service invocation.
		HLAreturnedArgument	Textual depiction of the argument returned by the service invocation (null if the service does not normally return a value or if *HLAsuccessIndicator* is HLAfalse).
		HLAexception	Textual description of the exception raised by this service invocation (null if *HLAsuccessIndicator* is HLAtrue).
		HLAserialNumber	This is a per-joined federate serial number that shall start at zero and shall increment by 1 for each *HLAmanager.HLAfederate.HLAreport.HLAreportServiceInvocation* interaction that represents service invocations to or from the respective joined federate.

Table 17—MOM parameter definitions table *(continued)*

Class		Parameter	Definition
HLAmanager.HLAfederate.HLAreport	HLAreportMOMexception	HLAservice	Name of the service interaction that had a problem or raised an exception.
		HLAexception	Textual depiction of the problem/exception.
		HLAparameterError	HLAtrue if there was an incorrect number of interaction parameters, HLAfalse otherwise.
	HLAreportSynchronizationPoints	HLAsyncPoints	List of the in progress federation execution synchronization points.
	HLAreportSynchronizationPointStatus	HLAsyncPointName	Name of a particular synchronization point.
		HLAsyncPointFederates	List of each federate (and its synchronization status) associated with the particular synchronization point.
HLAmanager.HLAfederate.HLAservice	HLAresignFederationExecution	HLAresignAction	Action that the RTI is to take in conjunction with the resignation.
	HLAsynchronizationPointAchieved	HLAlabel	Label associated with the synchronization point.
	HLAfederateSaveComplete	HLAsuccessIndicator	Whether the save was successful.
	HLAfederateRestoreComplete	HLAsuccessIndicator	Whether the restore was successful.
	HLApublishObjectClassAttributes	HLAobjectClass	Object class for which the joined federate's publication shall change.
		HLAattributeList	List of handles of attributes of *HLAobjectClass,* that the joined federate shall now publish.
	HLAunpublishObjectClassAttributes	HLAobjectClass	Object class for which the joined federate's unpublication shall change.
		HLAattributeList	List of handles of attributes of *HLAobjectClass,* that the joined federate shall now unpublish.
	HLApublishInteractionClass	HLAinteractionClass	Interaction class that the joined federate shall publish.
	HLAunpublishInteractionClass	HLAinteractionClass	Interaction class that the joined federate shall no longer publish.

Table 17—MOM parameter definitions table *(continued)*

Class		Parameter	Definition
HLAmanager.HLAfederate.HLAservice	HLAsubscribeObjectClassAttributes	HLAobjectClass	Object class for which the joined federate's subscription shall change.
		HLAattributeList	List of handles of attributes of *HLAobjectClass* to which the joined federate shall now subscribe.
		HLAactive	Whether the subscription is active.
	HLAunsubscribeObjectClassAttributes	HLAobjectClass	Object class for which the joined federate's subscription shall change.
		HLAattributeList	List of handles of attributes of *HLAobjectClass* to which the joined federate shall now unsubscribe.
	HLAsubscribeInteractionClass	HLAinteractionClass	Interaction class to which the joined federate shall subscribe.
		HLAactive	Whether the subscription is active.
	HLAunsubscribeInteractionClass	HLAinteractionClass	Interaction class to which the joined federate will no longer be subscribed.
	HLAdeleteObjectInstance	HLAobjectInstance	Handle of the object instance that is to be deleted.
		HLAtag	Tag associated with the deletion.
		HLAtimeStamp	Time stamp of the deletion (optional).
	HLAlocalDeleteObjectInstance	HLAobjectInstance	Handle of the object instance that is to be deleted.
	HLAchangeAttributeTransportationType	HLAobjectInstance	Handle of the object instance whose attribute transportation type is to be changed.
		HLAattributeList	List of the handles of instance attributes whose transportation type is to be changed.
		HLAtransportation	Transportation type to be used for updating instance attributes in the list.
	HLAchangeInteractionTransportationType	HLAinteractionClass	Interaction class whose transportation type is changed by this service invocation.
		HLAtransportation	Transportation type to be used for sending the interaction class.

Table 17—MOM parameter definitions table *(continued)*

Class		Parameter	Definition
HLAmanager.HLAfederate.HLAservice	HLAunconditionalAttributeOwnershipDivestiture	HLAobjectInstance	Handle of the object instance whose attributes' ownership is to be divested.
		HLAattributeList	List of handles of instance attributes belonging to *HLAobjectInstance* whose ownership is to be divested by the joined federate.
	HLAenableTimeRegulation	HLAlookahead	Lookahead to be used by the joined federate while regulating other joined federates.
	HLAtimeAdvanceRequest	HLAtimeStamp	Time stamp requested.
	HLAtimeAdvanceRequestAvailable	HLAtimeStamp	Time stamp requested.
	HLAnextMessageRequest	HLAtimeStamp	Time stamp requested.
	HLAnextMessageRequestAvailable	HLAtimeStamp	Time stamp requested.
	HLAflushQueueRequest	HLAtimeStamp	Time stamp requested.
	HLAmodifyLookahead	HLAlookahead	New value for lookahead.
	HLAchangeAttributeOrderType	HLAobjectInstance	Handle of the object instance whose attribute order type is to be changed.
		HLAattributeList	List of the handles of instance attributes whose order type is to be changed.
		HLAsendOrder	Order type to be used for sending the instance attribute list.
	HLAchangeInteractionOrderType	HLAinteractionClass	Interaction class whose order type is changed by this service invocation.
		HLAsendOrder	Order type to be used for sending the interaction class.
HLAmanager.HLAfederation.HLAadjust	HLAsetSwitches	HLAautoProvide	Set the federation-wide Auto-Provide Switch to the provided value (this parameter is required only when a change of this switch value is needed).
		HLAconveyRegionDesignatorSets	Set the federation-wide Convey Region Designator Sets Switch to the provided value (this parameter is required only when a change of this switch value is needed).

12. Programming language mappings

12.1 Overview

The services described in this document to this point have been described in a programming language neutral way. This is because support is provided for multiple programming languages as described in the annexes to this document. This clause provides the reader with a mapping from the abstract view from the body of this document (Clause 4 through Clause 11) to the programming language specific views in the APIs (Annex A through Annex C).

The mappings to be described in Clause 12 are

— Of designators from Clause 4 through Clause10 to handles and names in the APIs

— From services in Clause 4 through Clause10 to methods in the APIs

— From abstract datatypes in Clause 4 through Clause10 to classes or datatypes in the APIs

12.2 Designators

As explained in 1.4.2, designators are a generalization of both names and handles. In most cases, what is referred to as a designator in Clause 4 through Clause 10 has both a name and a handle. Which specific form of a designator is used in the APIs is dependent on the language and service in which it is used. The APIs should be referenced to see which form is needed, although in most cases, the handle is used. Clause 10 presents services that shall be used to map names to handles and vice versa.

Names in most cases (e.g., object class names, attribute names, interaction class names, etc.) come from the FDD. As such they are known ahead of time to all federates. Object instance names, however, are different. These names may be chosen by the registering federate, or they may be created dynamically by the RTI. As such, these names often are not predictable.

Handles in all cases are generated by the RTI. In most cases, where the handles correspond to names from the FDD, these handles shall be generated in a repeatable manner. This means that if two federation executions are created on the same version of the same RTI with the same FDD file, then the handles assigned to the same name shall be the same in each federation execution. This property facilitates saves and restores such that handles, rather than just the names, can be saved by federates. This property does not, however, imply that an RTI must assign handles in a manner known a priori to federates.

Handles shall be unique in a federation execution. Each federate shall know the same item by the same handle. This means that the federate registering an object instance shall know it by the same handle as another federate that discovers the object instance.

Often during an federation execution, it is useful for federates to send information identifying an item to another federate. For example, one federate may wish to send an interaction to another federate informing it that one particular object instance is of particular importance. To send information identifying an item to another federate, there are two options: send the name of the item, or send the handle of the item.

To send a handle as an attribute value or as a parameter value, the handle must be encoded prior to sending, and it must be reconstituted by the receiver. The precise means of encoding and reconstituting a handle will be described for each API in the following sections (see 12.5.2.4, 12.3.2.6, and 12.4.2.4). However, a handle encoded using any API (e.g., C++) shall be reconstitutable in all APIs. Also, a handle of a given type (e.g., an object instance handle) shall only be reconstitutable as that type.

Handles that are sent as part of MOM attribute value updates or as parameter values of MOM interactions are sent encoded and may be reconstituted in the manner described for each API.

12.3 Ada 95

The Ada 95 API is intended to achieve several goals as follows:

— To provide as natural a mapping to the abstract interface specification as possible.

— To provide safe access to the RTI data and services.

— To provide flexibility to RTI implementers while adhering to a common API (as used by the federate programmer) and without adding significant overhead.

The following sections elaborate on some of these details, from the perspective of the federate programmer. This is not intended as a guide for RTI implementation.

The Ada 95 API is composed of a single package. As the name of this package is supplier dependent, it is highly recommended to the federate programmer to create an RTI package that renames the supplied package. A new RTI Ada 95 package specification should be created containing the renaming instruction:

```
-- File: rti.ads
package RTI renames <Some_Supplier>_RTI;
```

Then, the federate programmer code looks like:

```
with RTI;
package My_Federate is ...
```

The same should be done for the package, including concrete implementations for Logical Time and Logical Time Interval types.

12.3.1 Services

In general, each service in Clause 4 through Clause 10 corresponds to one subprogram in each API. In some cases, because of optional arguments to a service or programming language constraints, a single service may be mapped to multiple subprograms in an API.

For the Ada 95 API, the federate initiated services invoked on the RTI are subprograms taking an **`RTI_Ambassador`** instance as first argument, and RTI initiated service callbacks on the federate are primitives of the **`Federate_Ambassador`** tagged type. The RTI and Federate Ambassador services in the Ada 95 API that do not follow a “one-to-one” mapping with services in Clause 4 through Clause 10 of the specification are described in the following sections.

12.3.1.1 *Join federation execution*

In addition to the supplied arguments listed with the description of this service, the Ada 95 API requires the federate to supply an instance of **`Federate_Ambassador`**, **`Logical_Time`**, and **`Logical_ Time_Interval`**.

12.3.1.2 *Register federation synchronization point*

The optional argument representing the set of joined federate designators is implemented in the Ada 95 API as two **`Register_Federation_Synchronization_Point`** procedures: one that takes a **`Federate_Handle_Set`** as an argument, and one that does not.

12.3.1.3 *Confirm synchronization point registration †*

The registration success indicator is implemented in the Ada 95 API as two distinct primitives: **`Synchronization_Point_Registration_Succeeded`** and **`Synchronization_Point_Registration_Failed`**. A Synchronization Point Failure Reason (discussed in 12.3.2.10) is passed as an argument to **`Synchronization_Point_Registration_Failed`**.

12.3.1.4 *Request federation save*

The optional time stamp argument is implemented as two **`Request_Federation_Save`** procedures, one of which takes a **`Logical_Time`** as an argument.

12.3.1.5 *Initiate federate save †*

The optional argument representing the time stamp is implemented in the Ada 95 API as two **`Initiate_Federate_Save`** primitives: one that takes a **`Logical_Time`** as an argument, and one that does not.

12.3.1.6 *Federate save complete*

The federate save-success indicator is implemented in the Ada 95 API as two distinct primitives: **`Federate_Save_Complete`** and **`Federate_Save_Not_Complete`**.

12.3.1.7 *Federation saved †*

This service corresponds to two procedures in the Ada 95 API: **`Federation_Saved`** and **`Federation_Not_Saved`**. A Save Failure Reason (discussed in 12.3.2.11) is passed as an argument to **`Federation_Not_Saved`**.

12.3.1.8 *Confirm federation restoration request †*

This service is mapped to the following two procedures in the Ada 95 API: **`Request_Federation_Restore_Succeeded`** and **`Request_Federation_Restore_Failed`**.

12.3.1.9 *Federate restore complete*

This service is mapped to the following two procedures in the Ada 95 API: **`Federate_Restore_Complete`** and **`Federate_Restore_Not_Complete`**.

12.3.1.10 *Federation restored †*

This service corresponds to two primitives in the Ada 95 API: **`Federation_Restored`** and **`Federation_Not_Restored`**. A Restore Failure Reason (discussed in 12.3.2.12) is passed as an argument to **`Federation_Not_Restored`**.

12.3.1.11 *Unpublish Object class attributes*

The optional argument representing the set of attribute designators is implemented in the Ada 95 API as two procedures: **`Unpublish_Object_Class`** and **`Unpublish_Object_Class_Attributes`**. The latter of the two takes an **`Attribute_Handle_Set`** as an argument.

12.3.1.12 *Unsubscribe object class attributes*

The optional argument representing the set of attribute designators is implemented in the Ada 95 API as two procedures: **`Unsubscribe_Object_Class`** and **`Unsubscribe_Object_Class_Attributes`**. The latter of the two takes an **`Attribute_Handle_Set`** as an argument.

12.3.1.13 *Object instance name reserved* †

This service corresponds to two primitives in the Ada 95 API: **`Object_Instance_Name_Reservation_Succeeded`** and **`Object_Instance_Name_Reservation_Failed`**.

12.3.1.14 *Register object instance*

The optional argument representing the object instance name is implemented in the Ada 95 API as two **`Register_Object_Instance`** procedures. One that takes a **`Wide_String`** as an argument for the object instance name, and one that does not.

12.3.1.15 *Update attribute values*

The optional argument representing the time stamp is implemented in the Ada 95 API as two **`Uptdate_Attribute_Values`** procedures. One that takes a **`Logical_Time`** as an argument for the time stamp, and one that does not. The version that takes a **`Logical_Time`** also returns a **`Message_Retraction_Handle`** for the message retraction designator.

12.3.1.16 *Reflect attribute values* †

This service corresponds to six overloaded versions of the **`Reflect_Attribute_Values`** primitives in the Ada 95 API. The first version takes none of the optional arguments. The second version adds only the optional set of sent region designators, represented as an instance of a **`Region_Handle_Set`**. Each of the remaining four versions takes a **`Logical_Time`** as an argument for the time stamp, and takes an **`Order_Type`** as an argument for the received message order type. To this, the fourth version adds the optional set of sent region designators, represented as an instance of a **`Region_Handle_Set`**. The fifth and sixth versions take a **`Message_Retraction_Handle`** as an argument for the message retraction designator. And finally, the sixth version adds the optional set of sent region designators, represented as an instance of a **`Region_Handle_Set`**.

12.3.1.17 *Send interaction*

The optional argument representing the time stamp is implemented in the Ada 95 API as two **`Send_Interaction`** procedures. One that takes a **`Logical_Time`** as an argument for the time stamp, and one that does not. The version that takes a **`Logical_Time`** also returns a **`Message_Retraction_Handle`** for the message retraction designator.

12.3.1.18 *Receive interaction* †

This service corresponds to six overloaded versions of the **`Receive_Interaction`** primitives in the Ada 95 API. The first version takes none of the optional arguments. The second version adds only the optional set of sent region designators, represented as an instance of a **`Region_Handle_Set`**. Each of the remaining four versions takes a **`Logical_Time`** as an argument for the time stamp, and takes an **`Order_Type`** as an argument for the received message order type. To this, the fourth version adds the optional set of sent region designators, represented as an instance of a **`Region_Handle_Set`**. The fifth and sixth versions take a **`Message_Retraction_Handle`** as an argument for the message retraction designator. And finally, the sixth version adds the optional set of sent region designators, represented as an instance of a **`Region_Handle_Set`**.

12.3.1.19 *Delete object instance*

The optional argument representing the time stamp is implemented in the Ada 95 API as two **`Delete_Object_Instance`** procedures. One that takes a **`Logical_Time`** as an argument for the time stamp, and one that does not. The version that takes a **`Logical_Time`** also returns a **`Message_Retraction_Handle`** for the message retraction designator.

12.3.1.20 *Remove object instance †*

This service corresponds to three overloaded versions of the **`Remove_Object_Instance`** primitives in the Ada 95 API. The first version takes none of the optional arguments. Both of the remaining two versions take a **`Logical_Time`** as an argument for the time stamp, and take an **`Order_Type`** as an argument for the received message order type. Finally, the third version takes a **`Message_Retraction_Handle`** as an argument for the message retraction designator.

12.3.1.21 *Request attribute value update*

This service corresponds to two overloaded procedures named **`Request_Attribute_Value_Update`** in the Ada 95 API. One takes an **`Object_Instance_Handle`** to represent the object instance designator. The other takes an **`Object_Class_Handle`** to represent the object class designator.

12.3.1.22 *Inform attribute ownership †*

In the Ada 95 API, this service corresponds to three distinct primitives: **`Attribute_Is_Not_Owned`**, **`Attribute_Is_Owned_By_RTI`**, and **`Inform_Attribute_Ownership`**. All three primitives take an **`Object_Instance_Handle`** to represent the object instance designator and an **`Attribute_Handle`** to represent the attribute designator as arguments. In addition, **`Inform_Attribute_Ownership`** takes a **`Federate_Handle`** as an argument to represent the federate designator.

12.3.1.23 *Register object instance with regions*

This service corresponds to two overloaded versions of the **`Register_Object_Instance_With_Regions`** procedure in the Ada 95 API. The second version takes a **`Wide_String`** as an argument to represent the optional object instance name.

12.3.1.24 *Send interaction with regions*

The optional argument representing the time stamp is implemented in the Ada 95 API as two **`Send_Interaction_With_Regions`** procedures. One that takes a **`Logical_Time`** as an argument for the time stamp, and one that does not. The version that takes a **`Logical_Time`** also returns a **`Message_Retraction_Handle`** for the message retraction designator.

12.3.1.25 *Initialize RTI*

The *Initialize RTI* service corresponds to the **`Initialize_RTI`** function in the Ada 95 API. The optional RTI initialization set of strings argument is supplied as an array of **`Unbounded_Wide_String`**. RTI unused arguments are returned by the **`Initialize_RTI`** function as an array of **`Unbounded_Wide_String`**.

12.3.2 Abstract datatypes

Subclause 1.4.2 presents general conventions used to describe the abstract datatypes used in Clause 4 through Clause 10. In this clause the abstract datatypes are further elaborated by mapping an individual datatype to its implementation in the Ada 95 API. Note that the datatypes from Clause 11 are explained there and are not repeated here. Additionally, designators are a subject in their own right and are covered in 12.2.

The ISO/IEC committee ratified the Ada 95 Standard as ISO/IEC 8652:1995 *Programming Languages—Ada* [B7]. The HLA Ada 95 API does not make use of the optional parts of the Ada 95 Standard.

A description of the nature and properties of these standard Ada 95 constructs is far beyond the scope of this document. Instead, readers are referred to the Ada 95 ISO standard as well as any of the vast assortment of Ada 95 references. Four such references are as follows:

> [B1] This document is intended to help computer professionals produce better Ada programs by identifying a set of stylistic guidelines that will directly impact the quality of their Ada 95 programs.
>
> [B2] This gives an introduction to the new features of Ada, and it explains the rationale behind them. Ada 95 new programmers should read this first.
>
> [B3] This document contains all of the text in the "*Ada 95 Reference Manual,*" plus various annotations. It is intended primarily for compiler writers, validation test writers, and others who wish to study the fine details. The annotations include detailed rationale for individual rules and explanations of some of the more arcane interactions among the rules.
>
> [B7] This document is also called "*Ada 95 Reference Manual.*"

12.3.2.1 Names, labels, and strings

The names, labels, and strings described in Clause 4 through Clause 10 of this specification are mapped to the Ada 95 predefined **`Wide_String`** type.

12.3.2.2 Boolean values

The Boolean values described in Clause 4 through Clause 10 of this specification are mapped to the Ada 95 predefined **`Boolean`** type.

12.3.2.3 Exceptions

The exceptions described in Clause 4 through Clause 10 are mapped to Ada 95 exceptions declared in the API.

12.3.2.4 Encoded values

The subtype **`Encoded_Value`** is provided for encoded values. This type is a subtype of the standard Ada 95 type **`Ada.Streams.Stream_Element_Array`**. Note that the use of serialization capabilities provided by the language combined with the use of the **`Ada.Streams`** package could ease the encoding and decoding of user complex datatypes.

12.3.2.5 User-supplied tag

The User-Supplied Tag is mapped to the subtype **`User_Supplied_Tag`** of the predefined Ada 95 type **`Ada.Streams.Stream_Element_Array`**.

12.3.2.6 Handles

Most designators in the Ada 95 API are handles. Each handle is a distinct private type. The API defines the following handle types: **`Federate_Handle`**, **`Object_Class_Handle`**, **`Interaction_Class_Handle`**, **`Object_Instance_Handle`**, **`Attribute_Handle`**, **`Parameter_Handle`**, **`Dimension_Handle`**, **`Message_Retraction_Handle`**, and **`Region_Handle`**.

Two instances of a type can be compared for equality or order ("**`<`**" function) and one can be assigned to the other. These handles are always generated by an RTI implementation. Additionally an **`Encode`** function is provided that returns the corresponding handle's encoded data of type **`Encoded_Value`**. Furthermore, each handle type has a **`Decode`** function that takes the corresponding encoded data as an argument.

Using this interface, it is possible for the RTI user to save and restore any handle provided by the RTI, as well as to pass handles as attribute values or parameter values among federates. To send a handle as an attribute value or as a parameter value, the federate programmer should invoke the **`Encode`** function of the handle. Once delivered, the encoded handle can be reconstituted into an instance of its corresponding handle type using that handle's **`Decode`** function.

Finally, the **`Image`** function is provided to convert an instance of a handle to a **`Wide_String`**. The **`Wide_String`** produced, however, is not guaranteed to be the same as the name to which the handle corresponds (assuming there is a name that corresponds to that kind of handle). If the user needs the name form of a handle, the appropriate support service from clause 10 should be used.

12.3.2.7 Resignation directive

The *Resign Federation Execution* service allows the specification of one of six directives. The Ada 95 API defines the enumeration type **`Resign_Action`** with six values:

- **`Unconditionally_Divest_Attributes`**
- **`Delete_Objects`**
- **`Cancel_Pending_Ownership_Acquisitions`**
- **`Delete_Objects_Then_Divest`**
- **`Cancel_Then_Delete_Then_Divest`**
- **`No_Action`**

12.3.2.8 Sets of designators

The sets of designators are implemented using indefinite **`private`** types. For all of these types a **`Positive`** indexed unconstrained array of the corresponding designator is declared. Set types ensure that contained designators are unique, which is something that **`array`** does not ensure. The set types are as follows:

- **`Federate_Handle_Set`** type implements the set of federate designators.
- **`Attribute_Handle_Set`** type implements the set of attribute designators.
- **`Dimension_Handle_Set`** type implements the set of dimension designators.
- **`Region_Handle_Set`** type implements the set of region designators.

Ada 95 predefined operations on set types are comparison and assignment. API provided operations on sets are as follows:

— The **`To_Set`** function returns a set built from a given array of designators.

— The **`To_Array`** function returns an array of contained designators sorted by increasing order from a given set.

— The "**`-`**" function returns a set, which is the difference between two given sets.

— The **`Intersection`** function returns a set, which is the intersection of two given sets.

12.3.2.9 Success indicators

Several of the requests a federate may make of a RTI must be acknowledged by the RTI to the federate. This acknowledgment can indicate that the request succeeded or failed. In the Ada 95 API, each request acknowledgment corresponds to a positive and a negative primitive of the **`Federate_Ambassador`** tagged type. For example

— **Synchronization_Point_Registration_Succeeded**

— **Synchronization_Point_ Registration_Failed**

— **Federation_Saved** and **Federation_Not_Saved**

— **Request_Federation_Restore_Succeeded**

— **Request_Federation_Restore_ Failed**

— **Federation_Resored** and **Federation_Not_Restored**

12.3.2.10 Synchronization point failure reason

When the **`Synchronization_Point_Registration_Failed`** procedure is invoked by the RTI on a federate, the RTI provides a value of type **`Synchronization_Failure_Reason`** as an argument. This type is implemented using an enumeration with the following two values, each of which can be compared with the value returned by the RTI:

— **Synchronization_Point_Label_Not_Unique**

— **Synchronization_Set_Member_Not_Joined**

12.3.2.11 Save failure reason

When the **`Federation_Not_Saved`** procedure is invoked by the RTI on a federate, the RTI provides a value of type **`Save_Failure_Reason`** as an argument. This type is implemented using an enumeration with the following five values, each of which can be compared with the value returned by the RTI:

— **RTI_Unable_To_Save**

— **Federate_Reported_Failure**

— **Federate_Resigned**

— **RTI_Detected_Failure**

— **Save_Time_Cannot_Be_Honored**

12.3.2.12 Restore failure reason

When the **Federation_Not_Restored** procedure is invoked by the RTI on a federate, the RTI provides a value of type **Restore_Failure_Reason** as an argument. This type is implemented using an enumeration with the following four values, each of which can be compared with the value returned by the RTI:

- **RTI_Unable_To_Restore**
- **Federate_Reported_Failure**
- **Federate_Resigned**
- **RTI_Detected_Failure**

12.3.2.13 List of joined federates save status

The list of joined federates and the save status of each is implemented as an array of **Federate_Save_Status** records. This record has two fields. The first field named **Federate** is the handle of a joined federate. The second field named **Status** of enumeration type **Save_Status** is the save status of this joined federate. **Save_Status** defines the following values:

- **No_Save_In_Progress**
- **Federate_Instructed_To_Save**
- **Federate_Saving**
- **Federate_Waiting_For_Federation_To_Save**

12.3.2.14 List of joined federates restore status

The list of joined federates and the restore status of each is implemented as an array of **Federate_Restore_Status** records. This record has two fields. The first field named **Federate** is the handle of a joined federate. The second field named **Status** of enumeration type **Restore_Status** is the restore status of this joined federate. **Restore_Status** defines the following values:

- **No_Restore_In_Progress**
- **Federate_Restore_Request_Pending**
- **Federate_Waiting_For_Restore_To_Begin**
- **Federate_Prepared_To_Restore**
- **Federate_Restoring**
- **Federate_Waiting_For_Federation_To_Restore**

12.3.2.15 Passive subscription indicator

The *Subscribe Object Class Attributes*, *Subscribe Interaction Class*, *Subscribe Object Class Attributes With Regions*, and *Subscribe Interaction Class With Regions* services each may optionally provide a **Boolean** value to indicate whether or not the subscription is active (**True** means the subscription is active). In the Ada 95 API, this is a default parameter to the procedures corresponding to each of the services. The default value is **True**.

12.3.2.16 Constrained set of attribute designator and value pairs

The *Update Attribute Values* and *Reflect Attribute Values †* services require a constrained set of attribute designator and value pairs as arguments. A constrained set of attribute designator and value pairs is implemented in the Ada 95 API as the **`limited private`** type **`Attribute_Handle_Value_Map`**. The following services are provided for this type:

- The **`Add`** procedure adds a pair, attribute handle, and value.
- The **`Cardinality`** function returns the number of pairs in the map.
- The **`Clear`** procedure empties the map.
- A **`Get_Value`** function returns the value at a given positive rank.
- Another **`Get_Value`** function returns the value for a given attribute handle.
- The **`Get_Attribute`** function returns the attribute handle at a given positive rank.
- The **`Get_Attributes`** function returns the set of attribute handles in the map.

As the **`Attribute_Handle_Value_Map`** type is **`limited`**, an instance cannot be compared nor assigned to another.

12.3.2.17 Constrained set of interaction parameter designator and value pairs

The *Send Interaction, Receive Interaction †*, and *Send Interaction With Regions* services require a constrained set of parameter designator and value pairs as arguments. A constrained set of parameter designator and value pairs is implemented in the Ada 95 API as the **`limited private`** type **`Parameter_Handle_Value_Map`**. The following services are provided for this type:

- The **`Add`** procedure adds a pair, parameter handle, and value.
- The **`Cardinality`** function returns the number of pairs in the map.
- The **`Clear`** procedure empties the map.
- A **`Get_Value`** function returns the value at a given positive rank.
- Another **`Get_Value`** function returns the value for a given parameter handle.
- The **`Get_Parameter`** function returns the parameter handle at a given positive rank.

As the **`Parameter_Handle_Value_Map`** type is **`limited`**, an instance cannot be compared nor assigned to another.

12.3.2.18 Message order type

Message order types are represented by the enumeration type **`Order_Type`** with two values: **`Receive_Ordered`** and **`Time_Stamp_Ordered`**.

An **`Encode`** function is provided to encode an **`Order_Type`** into a compact, opaque representation suitable for network transmission. This **`Encode`** function returns an **`Encoded_Value`**. Furthermore, a **`Decode`** function is provided that takes an **`Encoded_Value`** as an argument and returns an **`Order_Type`**.

The *Get Order Type* service converts an **`Order_Type`** into a **`Wide_String`** suitable for federation execution wide exchanges. Reciprocally, the *Get Order Name* service converts a **`Wide_String`** into an **`Order_Type`**. The predefined Ada 95 functions **`'Image`** and **`'Value`** are not suitable for federation execution-wide exchanges.

12.3.2.19 Transportation type

Transportation types are represented by the enumeration type **`Transportation_Type`** with two values: **`Reliable`** and **`Best_Effort.`**

An **`Encode`** function is provided to encode a **`Transportation_Type`** into a compact, opaque representation suitable for network transmission. This **`Encode`** function returns an **`Encoded_Value`**. Furthermore, a **`Decode`** function is provided that takes an **`Encoded_Value`** as an argument and returns a **`Transportation_Type`**.

The *Get Transportation Type* service converts a **`Transportation_Type`** into a **`Wide_String`** suitable for federation execution-wide exchanges. Reciprocally, the *Get Transportation Name* service converts a **`Wide_String`** into a **`Transportation_Type`**. The predefined Ada 95 functions **`'Image`** and **`'Value`** are not suitable for federation execution wide exchanges.

It is possible that some implementations of the RTI will allow additional transportation types. In this case, those RTI implementations shall extend the **`Transportation_Type`** type to include these additional transportation types. In no case shall the existing **`Transportation_Type`** type values be reduced or eliminated.

12.3.2.20 Ownership designator

The *Inform Attribute Ownership †* service supplies an ownership designator as one of its arguments. In the Ada 95 API, this corresponds to three distinct primitives: **`Attribute_Is_Not_Owned`**, **`Attribute_Is_Owned_By_RTI`**, and **`Inform_Attribute_Ownership`**. All three primitives take an **`Object_Instance_Handle`** and an **`Attribute_Handle`** as arguments. In addition, **`Inform_Attribute_Ownership`** takes a **`Federate_Handle`** as an argument. Together, these three primitives combine to represent the ownership designator of the *Inform Attribute Ownership †* service.

12.3.2.21 Definition indicator

The *Query GALT* and *Query LITS* services return a time value that may be valid or not. These procedures also return a **`Boolean`** value that tells the validity of the returned time value.

12.3.2.22 Optional message retraction designator

The *Update Attribute Values*, *Send Interaction*, *Send Interaction With Regions*, and *Delete Object Instance* services return a message retraction designator that may be valid or not. As said previously, this designator is implemented by the type **`Message_Retraction_Handle`**. These procedures also return a **`Boolean`** value that tells the validity of the returned **`Message_Retraction_Handle`**.

12.3.2.23 Collection of attribute designator set and region designator set pairs

Several services require the federate to provide the RTI with a collection of attribute designator set and region designator set pairs. In the Ada 95 API, this corresponds to the **`limited private`** type **`Attributes_Regions_Pairs`**. An object of this type is constrained at the declaration by setting the **`Capacity`** discriminant with a natural integer value. The following services are provided for this type:

— The **`Set_Attributes_And_Regions`** procedure sets the attributes and the regions at a given rank.

— The **`Set_Attributes`** procedure sets the attributes at a given rank.

— The **`Set_Regions`** procedure sets the regions at a given rank.

— The **`Get_Attributes`** function returns the attributes at a given rank.

— The **`Get_Regions`** function returns the regions at a given rank.

As the **Attributes_Regions_Pairs** type is **limited**, an instance cannot be compared nor assigned to another.

12.3.2.24 Logical time, time stamps, and lookahead

In order to support customized implementations of logical time, time stamps, and lookahead, the Ada 95 API defines two abstract tagged types: **Logical_Time** and **Logical_Time_Interval**. These types have been designed to be implemented by the federate developer for use by an RTI implementation.

The **Logical_Time** type is used to represent logical time and time stamp values. It has functions to compare the relative order of one instance with another. Functions to compare an instance with the initial value (e.g., zero) and with the final value (e.g., infinity) are provided as well as are procedures to set an instance to the initial, the final value and to the same value of another. This generality is provided to allow federate programmers to define arbitrary lower and upper bounds on their implementation of the **Logical_Time** type.

Procedures to increase and decrease the current value of an instance are provided. However it should be noted that the second argument to these procedures is of type **Logical_Time_Interval**, and not of type **Logical_Time**.

Also, one instance of a **Logical_Time** may be subtracted from another, but not added. The result is of type **Logical_Time_Interval**.

Finally, the **Image** function is used to convert an instance of a **Logical_Time** to a **Wide_String**. The **Encode** function is provided to encode a **Logical_Time** into a compact, opaque representation suitable for network transmission. This **Encode** function returns an **Encoded_Value**. Furthermore, a **Decode** function is provided that takes an **Encoded_Value** as an argument and returns a **Logical_Time**.

The **Logical_Time_Interval** type is analogous to the **Logical_Time** type, but it is used to represent lookahead values. The chief differences in its interface are that there is no concept of a “final” or “largest possible” **Logical_Time_Interval** instance; consequently there are only primitives to set a **Logical_Time_Interval** instance to zero and another to set it to epsilon. Also, two **Boolean** functions, **Is_Zero** and **Is_Epsilon**, are provided for comparison to zero and epsilon, respectively. Epsilon is the smallest nonzero **Logical_Time_Interval** that is possible with the particular implementation. This value is used by an RTI implementation to support time management services for federates with zero lookahead. The value of epsilon is intended to be constant.

One instance of a **Logical_Time_Interval** may be subtracted from another, but not added.

Finally, the **Image** function is used to convert an instance of a **Logical_Time_Interval** to a **Wide_String**. The **Encode** function is provided to encode a **Logical_Time_Interval** into a compact, opaque representation suitable for network transmission. This **Encode** function returns an **Encoded_Value**. Furthermore, a **Decode** function is provided that takes an **Encoded_Value** as an argument and returns a **Logical_Time_Interval**.

Implementers of the Ada 95 API shall provide, as part of their implementation, implementations of the logical time interfaces, as follows:

— An implementation of the abstract type **Logical_Time** called **Integer_64_Time**. This implementation shall use a 64-bit two’s complement signed integer type as its underlying representation of time. It shall use 0 as the value of initial time and $2^{63} – 1$ as its final time. Those primitives that are defined to return or accept a **Logical_Time** shall actually return or expect an **Integer_64_Time**. Those primitives defined as returning or accepting a **Logical_**

Time_Interval'Class shall return or expect an **Integer_64_Time_Interval** (see next paragraph).

— An implementation of **Logical_Time_Interval** called **Integer_64_Time_Interval**. This implementation shall use a 64-bit two's complement signed integer type as its underlying representation of a time interval. Its value of epsilon shall be 1. Those primitives defined as returning or accepting a **Logical_Time_Interval** shall return or expect an **Integer_64_Time_ Interval**.

12.3.2.25 Dimension upper bound and range lower and upper bounds

Several DDM related services allow bounds to be get and set. The *Get Dimension Upper Bound* service gets the upper bound of a particular dimension. The remaining services are used to get and set the bounds of ranges of a region. All bounds are represented in the Ada 95 API using the type **HLA_DDM_Coordinate**.

Range lower and upper bounds are gathered into a **Range_Bounds** record that includes two **HLA_DDM_Coordinate** fields named **Lower** and **Upper**.

12.3.2.26 Wall-clock time

The *Evoke Callback* and *Evoke Multiple Callbacks* services take arguments specifying durations of wall-clock time expressed in seconds. Wall-clock time is represented in the Ada 95 API using a **Duration**.

12.3.3 Memory management semantics

The Ada 95 API is designed to avoid memory corruption. The memory cannot be corrupted in a normal usage of the API. However, unchecked byte-to-byte conversion on an RTI data (use of **Unchecked_Conversion** or equivalent) is a way to corrupt the memory, and so is highly not recommended.

The Ada 95 API is designed to avoid memory leaks. However, the programmer should understand that some calls are needed to allow memory release. For example, a federate-created region that becomes useless should be deleted using the **Delete_Region** procedure.

12.4 Java

The Java API is composed of interfaces and classes. Where an interface is specified, the RTI implementers shall furnish an implementation of the interface. For instance, **RTIambassador** is specified as an interface. Therefore, an RTI implementer shall furnish an implementation of the interface, for instance:

```
package com.myrticompany.amazingrti; public final class MyRTIimpl
implements hla.rti.RTIambassador { … }
```

Each such interface as specified remains part of the API. Thus, federate code can use the specified interface type. The **FederateAmbassador**, however, is an exception to this rule and must be implemented by a federate developer.

Classes defined in the Java API, such as **RTIexception** and **ResignAction**, shall be included in each implementation of the RTI.

The entire Java API is included in the package **hla.rti**. Because the specified interfaces and classes constitute the API, the entire package **hla.rti** shall be furnished without additions or deletions or change of package name as part of each implementation. Implementers shall place their implementation classes in another package.

12.4.1 Services

In general, each service in Clause 4 through Clause 10 corresponds to one method in each API. In some cases, because of optional arguments to a service or programming language constraints, a single service may be mapped to multiple methods in an API.

For the Java API, the federate-initiated services invoked on the RTI are methods of the **`RTIambassador`** interface, and RTI-initiated service callbacks on the federate are methods of the **`FederateAmbassador`** interface. The RTI and Federate Ambassador services in the Java API that do not follow a "one-to-one" mapping with the services in Clause 4 through Clause 10 of the specification are described in the following sections.

12.4.1.1 *Join federation execution*

In addition to the supplied arguments listed with the description of this service, the Java API requires the federate to supply an instance of **`interface FederateAmbassador`** and **`class MobileFederateServices`**. The latter must be filled in with instances of **`LogicalTimeFactory`** and **`LogicalTimeIntervalFactory`**.

12.4.1.2 *Register federation synchronization point*

The optional argument representing the set of joined federate designators is implemented in the Java API as two **`registerFederationSynchronizationPoint()`** methods. One that takes a **`FederateHandleSet`** as an argument, and one that does not.

12.4.1.3 *Confirm synchronization point registration* †

The registration success indicator is implemented in the Java API as two distinct methods: **`synchronizationPointRegistrationSucceeded()`** and **`synchronizationPointRegistrationFailed()`**. A Synchronization Point Failure Reason (discussed in 12.4.2.9), in the form of a **`SynchronizationPointFailureReason`**, is passed as an argument to **`synchronizationPointRegistrationFailed()`**.

12.4.1.4 *Request federation save*

The optional time stamp argument is implemented as two methods, one of which takes a **`LogicalTime`** as an argument.

12.4.1.5 *Initiate federate save* †

The optional time stamp argument is implemented as two methods, one of which takes a **`LogicalTime`** as an argument.

12.4.1.6 *Federate save complete*

This service corresponds to two methods in the Java API: **`federateSaveComplete()`** and **`federateSaveNotComplete()`**.

12.4.1.7 *Federation saved* †

This service corresponds to two methods in the Java API: **`federationSaved()`** and **`federationNotSaved()`**. A Save Failure Reason (discussed in 12.4.2.10), in the form of a **`SaveFailureReason`**, is passed as an argument to **`federationNotSaved()`**.

12.4.1.8 *Confirm federation restoration request †*

This service is mapped to the following two methods in the Java API: **`requestFederationRestoreSucceeded()`** and **`requestFederationRestoreFailed()`**.

12.4.1.9 *Federate restore complete*

This service is mapped to the following two methods in the Java API: **`federateRestoreComplete()`** and **`federateRestoreNotComplete()`**.

12.4.1.10 *Federation restored †*

This service corresponds to two methods in the Java API: **`federationRestored()`** and **`federationNotRestored()`**. A Restore Failure Reason (discussed in 12.4.2.11), in the form of a **`RestoreFailureReason`**, is passed as an argument to **`federationNotRestored()`**.

12.4.1.11 *Unpublish object class attributes*

The optional argument representing the set of attribute designators is implemented in the Java API as two methods: **`unpublishObjectClass()`** and **`unpublishObjectClassAttributes()`**. The latter of the two takes an **`AttributeHandleSet`** as an argument.

12.4.1.12 *Subscribe object class attributes*

The optional passive subscription indicator is implemented in the Java API as two methods: **`subscribeObjectClassAttributes()`** and **`subscribeObjectClassAttributesPassively()`**.

12.4.1.13 *Unsubscribe object class attributes*

The optional argument representing the set of attribute designators is implemented in the Java API as two methods: **`unsubscribeObjectClass()`** and **`unsubscribeObjectClassAttributes()`**. The latter of the two takes an **`AttributeHandleSet`** as an argument.

12.4.1.14 *Subscribe interaction class*

The optional passive subscription indicator is implemented in the Java API as two methods: **`subscribeInteractionClass()`** and **`subscribeInteractionClassPassively()`**.

12.4.1.15 *Object instance name reserved †*

This service corresponds to two methods in the Java API: **`objectInstanceNameReservationSucceeded()`** and **`objectInstanceNameReservationFailed()`**.

12.4.1.16 *Register object instance*

The optional argument representing the object instance name is implemented in the Java API as two **`registerObjectInstance()`** methods. One that takes a **`String`** as an argument for the object instance name, and one that does not.

12.4.1.17 *Update attribute values*

The optional argument representing the time stamp is implemented in the Java API as two **`uptdateAttributeValues()`** methods. One that takes a **`LogicalTime`** as an argument for the time

stamp, and one that does not. The version that takes a **`LogicalTime`** also returns a **`MessageRetractionReturn`** for the optional message retraction designator.

12.4.1.18 *Reflect Attribute Values †*

This service corresponds to six overloaded versions of the **`reflectAttributeValues()`** method in the Java API. The first version takes none of the optional arguments. The second version adds only the optional set of sent region designators, represented as an instance of a **`RegionHandleSet`**. Each of the remaining four versions takes a **`LogicalTime`** as an argument for the time stamp and takes an **`OrderType`** as an argument for the receive message order type. To this, the fourth version adds the optional set of sent region designators, represented as an instance of a **`RegionHandleSet`**. The fifth and sixth versions take a **`MessageRetractionHandle`** as an argument for the message retraction designator. And finally, the sixth version adds the optional set of sent region designators, represented as an instance of a **`RegionHandleSet`**.

12.4.1.19 *Send interaction*

The optional argument representing the time stamp is implemented in the Java API as two **`sendInteraction()`** methods. One that takes a **`LogicalTime`** as an argument for the time stamp, and one that does not. The version that takes a **`LogicalTime`** also returns a **`MessageRetractionReturn`** for the optional message retraction designator.

12.4.1.20 *Receive interaction †*

This service corresponds to six overloaded versions of the **`receiveInteraction()`** method in the Java API. The first version takes none of the optional arguments. The second version adds only the optional set of sent region designators, represented as an instance of a **`RegionHandleSet`**. Each of the remaining four versions takes a **`LogicalTime`** as an argument for the time stamp and takes an **`OrderType`** as an argument for the receive message order type. To this, the fourth version adds the optional set of sent region designators, represented as an instance of a **`RegionHandleSet`**. The fifth and sixth versions take a **`MessageRetractionHandle`** as an argument for the message retraction designator. And finally, the sixth version adds the optional set of sent region designators, represented as an instance of a **`RegionHandleSet`**.

12.4.1.21 *Delete object instance*

The optional argument representing the time stamp is implemented in the Java API as two **`deleteObjectInstance()`** methods. One that takes a **`LogicalTime`** as an argument for the time stamp, and one that does not. The version that takes a **`LogicalTime`** also returns a **`MessageRetractionReturn`** for the optional message retraction designator.

12.4.1.22 *Remove object instance †*

This service corresponds to three overloaded versions of the **`removeObjectInstance()`** method in the Java API. The first version takes none of the optional arguments. Both of the remaining two versions take a **`LogicalTime`** as an argument for the time stamp, and take an **`OrderType`** as an argument for the receive message order type. Finally, the third version takes an **`MessageRetractionHandle`** as an argument for the message retraction designator.

12.4.1.23 *Request attribute value update*

This service corresponds to two overloaded methods named **`requestAttributeValueUpdate()`** in the Java API. One takes an **`ObjectInstanceHandle`** to represent the object instance designator. The other takes an **`ObjectClassHandle`** to represent the object class designator.

12.4.1.24 *Inform attribute ownership †*

In the Java API, this service corresponds to three distinct methods: **`attributeIsNotOwned()`**, **`attributeIsOwnedByRTI()`**, and **`informAttributeOwnership()`**. All three methods take an **`ObjectInstanceHandle`** to represent the object instance designator and an **`AttributeHandle`** to represent the attribute designator as arguments. In addition, **`informAttributeOwnership()`** takes a **`FederateHandle`** as an argument to represent the federate designator.

12.4.1.25 *Register object instance with regions*

This service corresponds to two overloaded versions of the **`registerObjectInstanceWithRegions()`** method in the Java API. The second version takes a **`String`** as an argument to represent the optional object instance name.

12.4.1.26 *Subscribe object class attributes with regions*

The optional passive subscription indicator is implemented in the Java API as two methods: **`subscribeObjectClassAttributesWithRegions()`** and **`subscribeObjectClassAttributesPassivelyWithRegions()`**.

12.4.1.27 *Subscribe interaction class with regions*

The optional passive subscription indicator is implemented in the Java API as two methods: **`subscribeInteractionClassWithRegions()`** and **`subscribeInteractionClassPassivelyWithRegions()`**.

12.4.1.28 *Send interaction with regions*

The optional argument representing the time stamp is implemented in the Java API as two **`sendInteractionWithRegions()`** methods. One that takes a **`LogicalTime`** as an argument for the time stamp, and one that does not. The version that takes a **`LogicalTime`** also returns a **`MessageRetractionReturn`** for the optional message retraction designator.

12.4.1.29 *Initialize RTI*

In the Java API, **`initializeRTI()`** causes the RTI to perform whatever actions are necessary (after construction of the **`RTIambassador`** implementation) for the RTI to be ready for other services to be invoked. These actions may vary from implementation to implementation. Data necessary for these actions are passed as Java **`Properties`**. **`initializeRTI()`** returns **`Properties`** that were passed in but were unrecognized or not acted upon during **`initializeRTI()`**.

12.4.2 Abstract datatypes

In 1.4.2 general conventions used to describe the abstract datatypes used in Clause 4 through Clause 10 are presented. In this clause the abstract datatypes are further elaborated by mapping an individual datatype to its implementation in the Java API. Note that the datatypes from Clause 11 are explained there and are not repeated here. Additionally, designators are a subject in their own right and are covered in 12.2.

Abstract datatypes are mapped to idiomatic Java types (if possible, to a type in the Java Platform APIs). Java 2 is assumed. This is especially evident in the use of the Java Development Kit Collections framework. All interfaces extend, and classes implement, **`java.io.Serializable`**.

12.4.2.1 Names, labels, and strings

The names, labels, and strings described in Clause 4 through Clause 10 of this specification are mapped to **`java.lang.String`**.

12.4.2.2 Boolean values

The Java primitive type **`boolean`** is used to represent all Boolean values.

12.4.2.3 Exceptions

All exceptions in the Java API inherit from a class named **`RTIexception`**, which extends **`java.lang.Exception`**. This allows all exceptions in the Java API to be caught as a single base class.

12.4.2.4 Handles

Most designators in the Java API are handles. Each handle is a distinct Java interface that must be implemented by the RTI implementer. Each handle interface supports **`equals()`** and **`hashCode()`** so that handles can be used as keys in **`HashTables`**. Handles of different types are incommensurable.

Handle instances typically are returned from RTI services, e.g., **`getAttributeHandle()`**. These services return a reference to the handle interface and, thus, hide the implementation class. The federate programmer typically uses only handle references that have been returned by the RTI.

Each handle interface supports **`encode()`**, which will place a compact, opaque representation suitable for network transmission into the provided **`byte`** array at the specified offset. Each handle interface has a corresponding handle factory interface, which must also be implemented by the RTI implementer. The factory interface defines **`decode()`**, which returns a new handle instance from a representation in the provided **`byte`** array at the indicated offset. This mechanism is intended to allow handles to be saved and restored later, as well as to allow handles to be passed among federates as attribute values or parameter values.

Handle classes also provide a **`toString()`** method that returns a printable form of a handle. The string produced, however, is not guaranteed to be the same as the name to which the handle corresponds (assuming there is a name that corresponds to that kind of handle). If the user needs the name form of a handle, the appropriate support service from Clause 10 should be used.

12.4.2.5 Resignation directive

The *Resign Federation Execution* service allows the specification of one of six directives. These are represented by a type-safe enumeration, **`ResignAction`**. This class defines six **`static public final`** instances that define the six values that instances of this class may take.

12.4.2.6 Attribute and parameter values and user-supplied tag

Several services in the HLA interface allow the federate programmer to supply additional information to be transported by the RTI in an opaque manner, such as *Update Attribute Values*. Attribute and parameter values are represented as **`byte`** arrays, **`byte[]`**. **`byte[]`** is also used to supply or deliver user-supplied tags.

12.4.2.7 Sets of designators

Sets of designators, e.g., federate designators in *Register Federation Synchronization Point* or attribute designator sets in *Publish Object Class Attributes*, are represented by interfaces that extend

java.util.Set for the respective handle type. This interface is part of the Java 2 Collections framework. The RTI implementer's implementation of each such set shall adhere to the following:

— Members shall be of the appropriate type, e.g., **AttributeHandle** for **AttributeHandleSet**. Attempts to add an instance of an inappropriate type shall elicit **IllegalArgumentException**.

— Implementations of the sets must implement all defined operations (none are optional).

— Methods that might take an incommensurable set as an actual parameter shall throw **IllegalArgumentException**.

Each such handle set has a corresponding factory interface. Thus, the implementation class shall be hidden from the federate programmer. A set of handles is provided for federate handles, attribute handles, dimension handles, and region handles.

12.4.2.8 Success indicators

Several of the requests a federate may make of an RTI must be acknowledged by the RTI to the federate. This acknowledgment can indicate that the request succeeded or failed. In the Java API, each request acknowledgment corresponds to a positive and a negative callback method on the **FederateAmbassador**. For example, **federationSaved()** and **federationNotSaved()**, **requestFederation RestoreSucceeded()** and **requestFederationRestoreFailed()**, and **federation Restored()** and **federationNotRestored()**.

12.4.2.9 Synchronization point failure reason

When the **synchronizationPointRegistrationFailed()** method is invoked by the RTI on a federate, the RTI provides an instance of an object of type **SynchronizationPointFailureReason** as an argument. This class is a type-safe enumeration, similar to **ResignAction** described in 12.4.2.5. To facilitate the federate programmer's determination of the particular instance received from the RTI, the following **static public final** members are provided, each of which can be compared to the object returned by the RTI:

— **SYNCHRONIZATION_POINT_LABEL_NOT_UNIQUE**

— **SYNCHRONIZATION_SET_MEMBER_NOT_JOINED**

12.4.2.10 Save failure reason

When the **federationNotSaved()** method is invoked by the RTI on a federate, the RTI provides an instance of an object of type **SaveFailureReason** as an argument. This class is a type-safe enumeration, similar to **ResignAction** described in 12.4.2.5. To facilitate the federate programmer's determination of the particular instance received from the RTI, the following **static public final** members are provided, each of which can be compared to the object returned by the RTI:

— **RTI_UNABLE_TO_SAVE**

— **FEDERATE_REPORTED_FAILURE**

— **FEDERATE_RESIGNED**

— **RTI_DETECTED_FAILURE**

— **SAVE_TIME_CANNOT_BE_HONORED**

12.4.2.11 Restore failure reason

When the **`federationNotRestored()`** method is invoked by the RTI on a federate, the RTI provides an instance of an object of type **`RestoreFailureReason`** as an argument. This class is a type-safe enumeration, similar to **`ResignAction`** described in 12.4.2.5. To facilitate the federate programmer's determination of the particular instance received from the RTI, the following **`static public final`** members are provided, each of which can be compared to the object returned by the RTI:

— **RTI_UNABLE_TO_RESTORE**

— **FEDERATE_REPORTED_FAILURE**

— **FEDERATE_RESIGNED**

— **RTI_DETECTED_FAILURE**

12.4.2.12 List of joined federates save status

The list of joined federates and the save status of each is implemented as an array of **`FederateHandleSaveStatusPair`** objects. Each **`FederateHandleSaveStatusPair`** object contains a **`FederateHandle`** and a **`SaveStatus`** representing the save status for that joined federate. **`SaveStatus`** is a type-safe enumeration, similar to **`ResignAction`** described in 12.4.2.5, and it provides the following **`static public final`** members:

— **NO_SAVE_IN_PROGRESS**

— **FEDERATE_INSTRUCTED_TO_SAVE**

— **FEDERATE_SAVING**

— **FEDERATE_WAITING_FOR_FEDERATION_TO_SAVE**

12.4.2.13 List of joined federates restore status

The list of joined federates and the restore status of each is implemented as an array of **`FederateHandleRestoreStatusPair`** objects. Each **`FederateHandleRestoreStatusPair`** object contains a **`FederateHandle`** and a **`RestoreStatus`** representing the restore status for that joined federate. **`RestoreStatus`** is a type-safe enumeration, similar to **`ResignAction`** described in 12.4.2.5, and it provides the following **`static public final`** members:

— **NO_RESTORE_IN_PROGRESS**

— **FEDERATE_RESTORE_REQUEST_PENDING**

— **FEDERATE_WAITING_FOR_RESTORE_TO_BEGIN**

— **FEDERATE_PREPARED_TO_RESTORE**

— **FEDERATE_RESTORING**

— **FEDERATE_WAITING_FOR_FEDERATION_TO_RESTORE**

12.4.2.14 Passive subscription indicator

The *Subscribe Object Class Attributes, Subscribe Interaction Class*, *Subscribe Object Class Attributes With Regions*, and *Subscribe Interaction Class With Regions* services each may optionally provide a Boolean value to indicate whether or not the subscription is active. In the Java API, this corresponds to a pair of methods on **`RTIambassador`**, e.g., **`subscribeObjectClassAttributes()`** and **`subscribeObjectClassAttributesPassively()`**.

12.4.2.15 Constrained set of attribute designator and value pairs

The *Update Attribute Values* and *Reflect Attribute Values †* services require as arguments a set of attribute designator and value pairs. Such sets are instances of **`AttributeHandleValueMap`**. This interface extends **`java.util.Map`**, part of the Java 2 Collections framework. The RTI implementer's implementation of **`AttributeHandleValueMap`** shall adhere to the following:

— The keys are instances of **`AttributeHandle`**; **`IllegalArgumentException`** shall be thrown for violations.

— The values are instances of **`byte[]`**; **`IllegalArgumentException`** shall be thrown for violations.

— All **`java.util.Map`** operations shall be implemented; none are optional.

— The implementation shall not accept null mappings, i.e., a mapping to a **`null`** value.

The interface **`AttributeHandleValueMapFactory`** shall be implemented, and the implementation class for **`AttributeHandleValueMap`** shall be hidden from the federate programmer. The federate programmer shall obtain a reference to a factory instance by the method **`getAttributeHandleVal ueMapFactory()`** on **`RTIambassador`**.

12.4.2.16 Constrained set of interaction parameter designator and value pairs

The *Send Interaction, Receive Interaction †*, and *Send Interaction With Regions* services require as arguments a constrained set of parameter designator and value pairs. Such sets are instances of **`ParameterHandleValueMap`**. This interface extends **`java.util.Map`**, part of the Java 2 Collections framework. The RTI implementer's implementation of **`ParameterHandleValueMap`** shall adhere to the following:

— The keys are instances of **`ParameterHandle`**; **`IllegalArgumentException`** shall be thrown for violations.

— The values are instances of **`byte[]`**; **`IllegalArgumentException`** shall be thrown for violations.

— All **`java.util.Map`** operations shall be implemented; none are optional.

— The implementation shall not accept null mappings, i.e., a mapping to a **`null`** value.

The interface **`ParameterHandleValueMapFactory`** shall be implemented, and the implementation class for **`ParameterHandleValueMap`** shall be hidden from the federate programmer. The federate programmer shall obtain a reference to a factory instance by the method **`getParameterHandleVal ueMapFactory()`** on **`RTIambassador`**.

12.4.2.17 Message order type

Message order types are represented by a class that implements an enumerated type. This class, named **`OrderType`**, is very similar to the **`ResignAction`** class described in 12.5.2.5. The following global constants are provided: **`RECEIVE`** and **`TIMESTAMP`**. These constants can be passed to the RTI or compared to an instance of an **`OrderType`** object returned by the RTI.

The **`OrderType`** class shares some aspects of the interface to handles (see 12.4.2.4); specifically, the means to encode an order type, reconstitute an encoded order type, and produce a printable form of an order type are provided. Encoding order types and reconstitution of encoded order types work in the same way as for handles, with the exception that the **`decode()`** method is provided as a **`static`** member of the **`Order-Type`** class rather than as a member of a corresponding factory. Producing a printable form works the same

as for handles. The **`toString()`** method is used, and the string produced is not guaranteed to be the same as the name to which the order type corresponds (i.e., the name of the order type as found in the FDD). If the user needs the name form of an order type, the *Get Order Name* service should be used.

12.4.2.18 Transportation type

As with message order types described above, transportation types are represented by a class that implements an enumerated type. This class, named **`TransportationType`**, is very similar to the **`ResignAction`** class described in 12.5.2.5. The following global constants are provided: **`RELIABLE`** and **`BEST_EFFORT`**. These constants can be passed to the RTI or compared to an instance of an **`OrderType`** object returned by the RTI.

The **`TransportationType`** class shares some aspects of the interface to handles (see 12.4.2.4); specifically, the means to encode a transportation type, reconstitute an encoded transportation type, and produce a printable form of a transportation type are provided. Encoding transportation types and reconstitution of encoded transportation types work in the same way as for handles, with the exception that the **`decode()`** method is provided as a **`static`** member of the **`TransportationType`** class rather than as a member of a corresponding factory. Producing a printable form works the same as for handles. The **`toString()`** method is used, and the string produced is not guaranteed to be the same as the name to which the transportation type corresponds (i.e., the name of the transportation type as found in the FDD). If the user needs the name form of a transportation type, the *Get Transportation Name* service should be used.

It is possible that some implementations of the RTI will allow additional transportation types. In this case, those RTI implementations shall extend the **`TransportationType`** class to include these additional transportation types as well as to add the corresponding global constants. In no case shall the existing **`TransportationType`** class or the global constants of its type be reduced or eliminated.

12.4.2.19 Ownership designator

The *Inform Attribute Ownership †* service supplies an ownership designator as one of its arguments. In the Java API, this corresponds to three distinct methods: **`attributeIsNotOwned()`**, **`attributeIsOwnedByRTI()`**, and **`informAttributeOwnership()`**. All three methods take an **`ObjectInstanceHandle`** and an **`AttributeHandle`** as arguments. In addition, **`informAttributeOwnership()`** takes a **`FederateHandle`** as an argument. Together, these three methods combine to represent the ownership designator of the *Inform Attribute Ownership †* service.

12.4.2.20 Definition indicator

The *Query GALT* and *Query LITS* services are required to return an indicator to define whether or not their return value is valid. Both services return a value of type **`TimeQueryReturn`**. This structure contains a **`boolean`** member that indicates whether the **`LogicalTime`** is valid. If the member **`timeIsValid`** is **`false`**, the member **`time`** should not be used.

12.4.2.21 Optional message retraction designator

The *Update Attribute Values*, *Send Interaction*, *Send Interaction With Regions*, and *Delete Object Instance* services may return a message retraction designator under certain circumstances. In the Java API, this is implemented much like the Definition Indicator described above. The methods corresponding to these services always return a **`MessageRetractionReturn`**. The member **`handle`** of this structure should be used only if the member **`retractionHandleIsValid`** is **`true`**.

12.4.2.22 Collection of attribute designator set and region designator set pairs

Several services require the federate to provide the RTI with a collection of attribute designator set and region designator set pairs. In the Java API, this corresponds to the interface **`AttributeSetRegionSetPairList`**. This interface extends **`java.util.List`**, part of the Java 2 Collections framework. The RTI implementer's implementation of **`ParameterHandleValuePairSet`** shall adhere to the following:

— The elements are instances of the class **`AttributeRegionAssociation`**; **`IllegalArgumentException`** shall be thrown for violations.

— All **`java.util.List`** operations shall be implemented; none are optional.

— The implementation shall not accept null elements; i.e., an element whose value is **`null`**.

— No operation shall accept an incommensurate type; **`IllegalArgumentException`** shall be thrown for violations.

12.4.2.23 Logical time, time stamps, and lookahead

In order to support customized implementations of logical time, time stamps, and lookahead, the Java API defines a group of closely related interfaces: **`LogicalTime`**, **`LogicalTimeInterval`**, **`LogicalTimeFactory`**, and **`LogicalTimeIntervalFactory`**. These classes have been designed to be implemented by the federate developer for use by an RTI implementation.

The **`LogicalTime`** interface is used to represent logical time and time stamp values. It has methods to compare the relative order of one instance with another, and **`LogicalTime`** and **`LogicalTimeInterval`** extend the **`Comparable`** interface. Methods to compare an instance with the initial value (e.g., zero) and with the final value (e.g., infinity) are provided as well as are methods to set an instance to the initial and final values. This generality is provided to allow federate programmers to define arbitrary lower and upper bounds on their implementation of the **`LogicalTime`** class. A method to set one instance to the same value as another is provided as well.

Methods to increase and decrease the current value of an instance are provided. However, it should be noted that the argument to these methods is of type **`LogicalTimeInterval`**, and not of type **`LogicalTime`**.

`LogicalTime` and **`LogicalTimeInterval`** specify the **`toString()`** method, which yields, as usual, a representation suitable for printing.

Both interfaces also specify an **`encode()`** method. This method shall place a compact, opaque representation suitable for network transmission into the provided **`byte`** array at the specified offset.

The **`LogicalTimeFactory`** interface has a method to create a **`LogicalTime`** instance whose value is the user-defined initial value. Additionally, a **`decode()`** method is provided that creates a **`LogicalTime`** from the network representation in the provided **`byte`** array at the indicated offset.

The **`LogicalTimeInterval`** interface is analogous to the **`LogicalTime`** interface, but it is used to represent lookahead values. The chief differences in its interface are that there is no concept of a "final" or "largest possible" **`LogicalTimeInterval`** instance; consequently, there are only methods to set a **`LogicalTimeInterval`** instance to zero or epsilon and to test an instance to determine if it is zero or epsilon. Epsilon is the smallest nonzero **`LogicalTimeInterval`** that is possible with the particular implementation. This value is used by an RTI implementation to support time management services for federates with zero lookahead. The value of epsilon is intended to be constant.

One instance of a **LogicalTimeInterval** may be subtracted from another, but not added. The **LogicalTimeIntervalFactory** is completely analogous to its **LogicalTime** counterpart.

Implementers of the Java API shall provide, as part of their implementation, implementations of the logical time interfaces, as follows:

— An implementation of the interface **LogicalTime** called **Integer64Time**. This implementation shall use a Java **long** as its underlying representation of time. It shall use 0 as the value of "initialTime" and **java.lang.Long.MAX_VALUE** as its final time. Those methods that are defined to return or accept a **LogicalTime** reference shall actually return or expect a reference to **Integer64Time**. Those methods defined as returning or accepting a **LogicalTimeInterval** reference shall return or expect a reference to **Integer64TimeInterval** (see next paragraph).

— An implementation of **LogicalTimeFactory** called **Integer64TimeFactory**. All of the methods of this implementation shall return references to **Integer64Time**, as defined above.

— An implementation of **LogicalTimeInterval** called **Integer64TimeInterval**. This implementation shall use a Java **long** as its underlying representation of a time interval. Its value of epsilon shall be 1. Those methods defined as returning or accepting a **LogicalTimeInterval** reference shall return or expect a reference to **Integer64TimeInterval**.

— An implementation of **LogicalTimeIntervalFactory** called **Integer64TimeIntegerFactory**. All of the methods of this implementation shall return references to **Integer64TimeInterval** as defined above.

These implementations shall be placed in a package other than **hla.rti** of the implementer's choosing.

12.4.2.24 Dimension upper bound and range lower and upper bounds

Several DDM-related services allow bounds to be get and set. The *Get Dimension Upper Bound* service gets the upper bound of a particular dimension. The *Get Range Bounds* and *Set Range Bounds* services are used to get and set the bounds of a range of a region. All bounds are represented in the Java API using a **long** integer. The *Get Dimension Upper Bound* service, as it is getting a single bound value, returns a **long**. The *Get Range Bounds* and *Set Range Bounds* services, as they are getting and setting a pair of values, actually return and take a **RangeBounds** object, which is composed of a pair of **long**s.

12.4.2.25 Wall-clock time

The *Evoke Callback* and *Evoke Multiple Callbacks* services take arguments specifying durations of wall-clock time expressed in seconds. Wall-clock time is represented in the Java API using a **double**.

12.4.3 Memory ownership semantics

The following are memory management conventions for parameters to methods on **RTIambassador** and **FederateAmbassador**.

All Java parameters, including object references, are passed by value. Therefore, there is no need to specify further conventions for primitive types.

Unless otherwise noted, reference parameters adhere to the following convention:

— The referenced object is created (or acquired) by the caller. The callee must copy during the call anything it wishes to save beyond the completion of the call.

— Unless otherwise noted, a reference returned from a method represents a new object created by the callee. The caller is free to modify the object whose reference is returned.

12.5 C++

The C++ API is intended to achieve several goals as follows:

— To provide as natural a mapping to the abstract interface specification as possible.

— To prevent as many common federate programming errors as practical. This includes compile-time type safety as well as dynamic memory management errors.

— To provide flexibility to RTI implementers while adhering to a common API (as used by the federate programmer) and without adding significant overhead.

— To embrace the recently finalized C++ standard; thus, creating an API that is future-proof.

The following sections elaborate on some of these details, from the perspective of the federate programmer. This is not intended as a guide for RTI implementation.

12.5.1 Services

In general, each service in Clause 4 through Clause 10 corresponds to one method in each API. In some cases, because of optional arguments to a service or programming language constraints, a single service may be mapped to multiple methods in an API.

For the C++ API, the federate-initiated services invoked upon the RTI are members of the **`RTI_RTIambassador`** class, and RTI, initiated service callbacks on the federate are members of the **`RTI_FederateAmbassador`** class. The RTI and Federate Ambassador services in the C++ API that do not follow a "one-to-one" mapping with the services in Clause 4 through Clause 10 of the specification are described in the following sections.

12.5.1.1 *Join federation execution*

In addition to the supplied arguments listed with the description of this service, the C++ API requires the federate to supply an instance of **`class RTI_FederateAmbassador`**, **`class RTI_LogicalTimeFactory`**, and **`class RTI_LogicalTimeIntervalFactory`**.

12.5.1.2 *Register federation synchronization point*

The optional argument representing the set of joined federate designators is implemented in the C++ API as two **`registerFederationSynchronizationPoint()`** methods. One that takes an **`RTI_FederateHandleSet`** as an argument, and one that does not.

12.5.1.3 *Confirm synchronization point registration* †

The registration success indicator is implemented in the C++ API as two distinct methods: **`synchronizationPointRegistrationSucceeded()`** and **`synchronizationPointRegistrationFailed()`**. A Synchronization Failure Reason (discussed in 12.5.2.9) is passed as an argument to **`synchronizationPointRegistrationFailed()`**.

12.5.1.4 *Request federation save*

The optional time stamp argument is implemented as two methods, one of which takes an **`RTI_LogicalTime`** as an argument.

12.5.1.5 *Initiate federate save †*

The optional time stamp argument is implemented as two methods, one of which takes an **`RTI_LogicalTime`** as an argument.

12.5.1.6 *Federate save complete*

This service corresponds to two methods in the C++ API: **`federateSaveComplete()`** and **`federateSaveNotComplete()`**.

12.5.1.7 *Federation saved †*

This service corresponds to two methods in the C++ API: **`federationSaved()`** and **`federationNotSaved()`**. A Save Failure Reason (discussed in 12.5.2.10) is passed as an argument to **`federationNotSaved()`**.

12.5.1.8 *Confirm federation restoration request †*

This service is mapped to the following two methods in the C++ API: **`requestFederationRestoreSucceeded()`** and **`requestFederationRestoreFailed()`**.

12.5.1.9 *Federate restore complete*

This service is mapped to the following two methods in the C++ API: **`federateRestoreComplete()`** and **`federateRestoreNotComplete().`**

12.5.1.10 *Federation restored †*

This service corresponds to two methods in the C++ API: **`federationRestored()`** and **`federationNotRestored()`**. A Restore Failure Reason (discussed in 12.5.2.11) is passed as an argument to **`federationNotRestored()`**.

12.5.1.11 *Unpublish object class attributes*

The optional argument representing the set of attribute designators is implemented in the C++ API as two methods: **`unpublishObjectClass()`** and **`unpublishObjectClassAttributes()`**. The latter of the two takes an **`RTI_AttributeHandleSet`** as an argument.

12.5.1.12 *Unsubscribe object class attributes*

The optional argument representing the set of attribute designators is implemented in the C++ API as two methods: **`unsubscribeObjectClass()`** and **`unsubscribeObjectClassAttributes()`**. The latter of the two takes an **`RTI_AttributeHandleSet`** as an argument.

12.5.1.13 *Object instance name reserved †*

This service corresponds to two methods in the C++ API: **`objectInstanceNameReservationSucceeded()`** and **`objectInstanceNameReservationFailed()`**.

12.5.1.14 *Register object instance*

The optional argument representing the object instance name is implemented in the C++ API as two **`registerObjectInstance()`** methods. One that takes an **`RTI_wstring`** as an argument for the object instance name, and one that does not.

12.5.1.15 *Update attribute values*

The optional argument representing the time stamp is implemented in the C++ API as two **`uptdateAttributeValues()`** methods. One that takes an **`RTI_LogicalTime`** as an argument for the time stamp, and one that does not. The version that takes an **`RTI_LogicalTime`** also returns an **`RTI_MessageRetractionHandle`** for the message retraction designator.

12.5.1.16 *Reflect attribute values* †

This service corresponds to six overloaded versions of the **`reflectAttributeValues()`** method in the C++ API. The first version takes none of the optional arguments. The second version adds only the optional set of sent region designators, represented as an instance of an **`RTI_RegionHandleSet`**. Each of the remaining four versions takes an **`RTI_LogicalTime`** as an argument for the time stamp and takes an **`RTI_OrderType`** as an argument for the receive message order type. To this, the fourth version adds the optional set of sent region designators, represented as an instance of an **`RTI_RegionHandleSet`**. The fifth and sixth versions take an **`RTI_MessageRetractionHandle`** as an argument for the message retraction designator. And finally, the sixth version adds the optional set of sent region designators, represented as an instance of an **`RTI_RegionHandleSet`**.

12.5.1.17 *Send interaction*

The optional argument representing the time stamp is implemented in the C++ API as two **`sendInteraction()`** methods. One that takes an **`RTI_LogicalTime`** as an argument for the time stamp, and one that does not. The version that takes an **`RTI_LogicalTime`** also returns an **`RTI_MessageRetraction Handle`** for the message retraction designator.

12.5.1.18 *Receive interaction* †

This service corresponds to six overloaded versions of the **`receiveInteraction()`** method in the C++ API. The first version takes none of the optional arguments. The second version adds only the optional set of sent region designators, represented as an instance of an **`RTI_RegionHandleSet`**. Each of the remaining four versions takes an **`RTI_LogicalTime`** as an argument for the time stamp and takes an **`RTI_OrderType`** as an argument for the receive message order type. To this, the fourth version adds the optional set of sent region designators, represented as an instance of an **`RTI_RegionHandleSet`**. The fifth and sixth versions take an **`RTI_MessageRetractionHandle`** as an argument for the message retraction designator. And finally, the sixth version adds the optional set of sent region designators, represented as an instance of an **`RTI_RegionHandleSet`**.

12.5.1.19 *Delete object instance*

The optional argument representing the time stamp is implemented in the C++ API as two **`deleteObjectInstance()`** methods. One that takes an **`RTI_LogicalTime`** as an argument for the time stamp, and one that does not. The version that takes an **`RTI_LogicalTime`** also returns an **`RTI_MessageRetractionHandle`** for the message retraction designator.

12.5.1.20 *Remove object instance* †

This service corresponds to three overloaded versions of the **`removeObjectInstance()`** method in the C++ API. The first version takes none of the optional arguments. Both of the remaining two versions take an **`RTI_LogicalTime`** as an argument for the time stamp and take an **`RTI_OrderType`** as an argument for the receive message order type. Finally, the third version takes an **`RTI_MessageRetractionHandle`** as an argument for the message retraction designator.

12.5.1.21 *Request attribute value update*

This service corresponds to two overloaded methods named **`requestAttributeValueUpdate()`** in the C++ API. One takes an **`RTI_ObjectInstanceHandle`** to represent the object instance designator. The other takes an **`RTI_ObjectClassHandle`** to represent the object class designator.

12.5.1.22 *Inform attribute ownership †*

In the C++ API, this service corresponds to three distinct methods: **`attributeIsNotOwned()`**, **`attributeIsOwnedByRTI()`**, and **`informAttributeOwnership()`**. All three methods take an **`RTI_ObjectInstanceHandle`** to represent the object instance designator and an **`RTI_AttributeHandle`** to represent the attribute designator as arguments. In addition, **`informAttributeOwnership()`** takes an **`RTI_FederateHandle`** as an argument to represent the federate designator.

12.5.1.23 *Register object instance with regions*

This service corresponds to two overloaded versions of the **`registerObjectInstanceWithRegions()`** method in the C++ API. The second version takes an **`RTI_wstring`** as an argument to represent the optional object instance name.

12.5.1.24 *Send interaction with regions*

The optional argument representing the time stamp is implemented in the C++ API as two **`sendInteractionWithRegions()`** methods. One that takes an **`RTI_LogicalTime`** as an argument for the time stamp, and one that does not. The version that takes an **`RTI_LogicalTime`** also returns an **`RTI_MessageRetractionHandle`** for the message retraction designator.

12.5.1.25 *Initialize RTI*

This service is implemented in the C++ API as the **`createRTIambassador`** method on the **`RTI_RTIambassadorFactory`** class. The optional set of initialization strings argument is implemented as **`RTI_vector<RTI_wstring>`**, which behaves in other respects as does **`std::vector<RTI_wstring>`**. See the discussion in 12.5.2 for further explanation.

12.5.2 Abstract datatypes

Subclause 1.4.2 presents general conventions used to describe the abstract datatypes used in Clause 4 through Clause 10. In this clause the abstract datatypes are further elaborated by mapping an individual datatype to its implementation in the C++ API. Note that the datatypes from Clause 11 are explained there and are not repeated here. Additionally, designators are a subject in their own right and are covered in 12.2.

The ISO/IEC committee ratified the C++ Standard as *ISO/IEC 14882 Standard for the C++ Programming Language* [B8]. The HLA C++ API utilizes Standard C++ constructs to the fullest extent possible. For example, **`std::wstring`** is used to represent character strings that might previously have been represented with the C-style construct, pointer to **`char`** (i.e., **`char *`**). Additionally, several standard C++ containers are used to represent the sets, constrained sets of pairs, and collections described in Clause 1. These containers are **`std::set<T>`**, **`std::map<T>`**, and **`std::vector<T>`**. Also, the C++ standard **`std::auto_ptr<T>`** is used to implement strict memory ownership transfer when passing instances of dynamically allocated datatypes across API.

A description of the nature and properties of these standard C++ constructs is well beyond the scope of this clause. Instead, readers are referred to the C++ ISO standard as well as any of the vast assortment of C++ references. Two such excellent references are [B10] and [B4].

Although the C++ standard has been complete for some time, many compilers and C++ libraries do not yet fully comply with this standard. Some compilers do not even support certain key features, such as namespaces. Balancing the reality of lagging compiler implementations with the benefits of adopting the use of standard C++ has proven to be somewhat troublesome for the implementation of the C++ API. While it is possible to work around compiler limitations, such workarounds can themselves become a problem. This is because users of the RTI may themselves choose to work around these limitations. However, not all users will necessarily chose the same approach; and all approaches are not necessarily compatible.

In order to ensure that the workaround for deficient C++ compilers chosen by the RTI will be compatible with all users, implementers, and platforms of the RTI, the C++ API makes use of an implementation of a subset of the standard C++ library that has been specially modified for this purpose. This modified subset of the standard C++ library is provided with the C++ API. It includes what is referred to as the standard template library (STL) as well as an implementation of **`std::wstring`**. This implementation closely tracks the C++ standard for those components, with an important, but trivial, difference:

> The use of the namespace **`std`** has been replaced with the logically identical prefix **`RTI_`**.

For example, instead of **`std::wstring`**, or **`std::set<T>`**, the RTI uses **`RTI_wstring`** and **`RTI_set<T>`**, respectively. The names of the corresponding C++ header files have been likewise prepended with the prefix **`RTI_`**. These files shall be supplied with RTI implementations. Note that related types are similarly renamed, e.g., **`std::set<int>::iterator`** becomes **`RTI_set <int>::RTI_iterator`**. Functionally, however, there is no difference between the **`std::`** types and their **`RTI_`** counterparts.

> It must be stressed that this workaround, necessitated by the unfortunate widespread use of nonconforming C++ compilers, is a solution necessitated by a temporary problem.

At such a time as when nonconforming C++ compilers are extinct, a future revision of this C++ API will deprecate all occurrences of the **`RTI_`** prefix in favor of the native implementation of the standard C++ library belonging to every C++ compiler.

12.5.2.1 Names, labels, and strings

The names, labels, and strings described in Clause 4 through Clause 10 of this specification are mapped to the **`RTI_wstring`** type. With the exception of its name, the **`RTI_wstring`** type has all of the properties of the **`std::wstring`** type of standard C++. See the discussion in 12.5.2 for further explanation.

12.5.2.2 Boolean values

Standard C++ provides a fundamental type, **`bool`**, intended for the representation of Boolean values. Regrettably, this type is not yet supported by all C++ compilers. As with the modified subset of the standard C++ library described in section 12.5.2, pending the extinction of such nonconforming compilers, the C++ API makes use of a class named **`RTI_bool`**, and the companion global constants: **`RTI_true`** and **`RTI_false`**.

12.5.2.3 Exceptions

In keeping with the philosophy outlined in section 12.5.2, exceptions in the C++ API have an interface identical with **`std::exception`**. In fact, all exceptions in the C++ API inherit from a class named **`RTI_exception`**. This allows all exceptions in the C++ API to be caught as a single base class.

12.5.2.4 Handles

Most designators in the C++ API are handles. Each handle is a distinct type, but they all share a common interface, implemented by the **RTI_Handle<T>** class template and specialized into one of the following concrete classes: **RTI_FederateHandle**, **RTI_ObjectClassHandle**, **RTI_InteractionClassHandle**, **RTI_ObjectInstanceHandle**, **RTI_AttributeHandle**, **RTI_ParameterHandle**, **RTI_DimensionHandle**, **RTI_MessageRetractionHandle**, and **RTI_RegionHandle**.

Two instances of each class with identical types can be compared for equality or order (i.e., less than), and one can be assigned to the other. A copy constructor is provided as well. These handles are always generated by an RTI implementation; however a default constructor is provided for convenience. Additionally, an **encode()** method is provided that returns an encoded form of a handle, **RTI_EncodedData**.

RTI_EncodedData has a **data()** method that returns an untyped pointer (i.e., a **void ***) as well as a **size()** method that returns the size in bytes (represented as a **size_t**) of the data returned by the **data()** method. A copy constructor and a copy-assignment operator are also provided.

Using this interface, it is possible for the RTI user to save and restore any handle provided by the RTI, as well as to pass handles as attribute values or parameter values among federates. To send a handle as an attribute value or as a parameter value, the **encode()** method of the handle should be invoked by the federate programmer. The returned **RTI_EncodedData** class has a **data()** method and a **size()** method that can be used to fill in the attribute value or parameter value. Once delivered, the encoded handle can be reconstituted into an instance of its corresponding handle class using that handle class's constructor.

Note well that the pointer value returned by the **data()** method is not required to have any particular byte alignment. Consequently, naively attempting to simply type cast this pointer and assign its contents to some type may generate an exception in the processor's memory management unit, likely resulting in the crash of the program. If for some reason it is desirable to perform this type of operation, a function such as **memcpy()** should be used.

Handle classes also provide a **toString()** method that returns a printable form of a handle. The string produced, however, is not guaranteed to be the same as the name to which the handle corresponds (assuming there is a name that corresponds to that kind of handle). If the user needs the name form of a handle, the appropriate support service from Clause 10 should be used.

12.5.2.5 Resignation directive

The *Resign Federation Execution* service allows the specification of one of six directives. To provide this specification, a type-safe enumeration, **enum**[9], has been implemented in the C++ API: **class RTI_ResignAction**. The six different directives of *Resign Federation Execution* can be accessed by invoking their corresponding **static** class method, e.g., **RTI_ResignAction::deleteObjects()**. Additionally, global **const** objects exist for each of the six directives, corresponding to the six **static** class methods, e.g., **RTI_DELETE_OBJECTS**.

This provides a familiar interface to users comfortable with the more primitive C++ **enum**, while providing compile-time type-safety as well. This means that an **int** cannot be used where the C++ API calls for the use of an **RTI_ResignAction**, or vice versa. Two instances of an **RTI_ResignAction** can be compared for equality or inequality, but not ordered.

[9] See Item 46 of [B9].

12.5.2.6 User-supplied tag

Several services in the HLA allow the user to supply additional information to be transported by the RTI in an opaque manner, such as *Update Attribute Values*. The C++ API provides **`class RTI_UserTag`** to achieve this goal. This class has a constructor that requires the user to provide an untyped pointer (i.e., a **`void *`**) as well as the size in bytes (represented as a **`size_t`**) of the data to which it points. Methods to return these values are provided as well, as is a copy constructor and a copy-assignment operator.

Note that the pointer value returned by the **`data()`** method is not required to have any particular byte alignment. Consequently, naively attempting to simply type cast this pointer and assign its contents to some type may generate an exception in the processor's memory management unit, likely resulting in the crash of the program. If for some reason, it is desirable to perform this type of operation, a function such as **`memcpy()`** should be used.

12.5.2.7 Sets of designators

Sets of designators are implemented using the standard C++ set container with the appropriate handle type. For convenience, each different set of designators has been aliased using a **`typedef`**. To ensure maximum portability between RTI implementations, federate programmers should use only this **`typedef`**. With the exception of the name, each handle set behaves in other respects as does **`std::set<T>`**. See the discussion in 12.5.2 for further explanation. The list of set types, with their corresponding **`typedef`**s are as follows:

— A set of federate designators uses **`RTI_set<RTI_FederateHandle>`** and the **`typedef RTI_FederateHandleSet`** should be used.

— A set of attribute designators uses **`RTI_set<RTI_AttributeHandle>`** and the **`typedef RTI_AttributeHandleSet`** should be used.

— A set of dimension designators uses **`RTI_set<RTI_DimensionHandle>`** and the **`typedef RTI_DimensionHandleSet`** should be used.

— A set of region designators uses **`RTI_set<RTI_RegionHandle>`** and the **`typedef RTI_RegionHandleSet`** should be used.

12.5.2.8 Success indicators

Several of the requests a federate may make of an RTI must be acknowledged by the RTI to the federate. This acknowledgment can indicate that the request succeeded or failed. In the C++ API, each request acknowledgment corresponds to a positive and a negative callback method on the **`RTI_FederateAmbassador`**. For example, **`federationSaved()`** and **`federationNotSaved()`**, **`requestFederationRestoreSucceeded()`** and **`requestFederationRestoreFailed()`**, and **`federationRestored()`** and **`federationNotRestored()`**.

12.5.2.9 Synchronization failure reason

When the **`synchronizationPointRegistrationFailed()`** method is invoked by the RTI on a federate, the RTI provides an instance of an object of type **`RTI_SynchronizationFailureReason`** as an argument. This class is a type-safe **`enum`**, very similar to **`RTI_ResignAction`** described in 12.5.2.5. To facilitate the federate programmer's determination of the particular instance received from the RTI, the following global constants are provided, each of which can be compared to the object returned by the RTI:

— **RTI_SYNCHRONIZATION_POINT_LABEL_NOT_UNIQUE**

— **RTI_SYNCHRONIZATION_SET_MEMBER_NOT_JOINED**

12.5.2.10 Save failure reason

When the **federationNotSaved()** method is invoked by the RTI on a federate, the RTI provides an instance of an object of type **RTI_SaveFailureReason** as an argument. This class is a type-safe **enum**, very similar to **RTI_ResignAction** described in 12.5.2.5. To facilitate the federate programmer's determination of the particular instance received from the RTI, the following global constants are provided, each of which can be compared to the object returned by the RTI:

— **RTI_RTI_UNABLE_TO_SAVE**

— **RTI_FEDERATE_REPORTED_FAILURE_DURING_SAVE**

— **RTI_FEDERATE_RESIGNED_DURING_SAVE**

— **RTI_RTI_DETECTED_FAILURE_DURING_SAVE**

— **RTI_SAVE_TIME_CANNOT_BE_HONORED**

12.5.2.11 Restore failure reason

When the **federationNotRestored()** method is invoked by the RTI on a federate, the RTI provides an instance of an object of type **RTI_RestoreFailureReason** as an argument. This class is a type-safe **enum**, very similar to **RTI_ResignAction** described in 12.5.2.5. To facilitate the federate programmer's determination of the particular instance received from the RTI, the following global constants are provided, each of which can be compared to the object returned by the RTI:

— **RTI_RTI_UNABLE_TO_RESTORE**

— **RTI_FEDERATE_REPORTED_FAILURE_DURING_RESTORE**

— **RTI_FEDERATE_RESIGNED_DURING_RESTORE**

— **RTI_RTI_DETECTED_FAILURE_DURING_RESTORE**

12.5.2.12 List of joined federates save status

The list of joined federates and the save status of each is implemented as a vector of federate designator and save status pairs. In the C++ API, this corresponds to an **RTI_vector<RTI_pair< RTI_FederateHandle, RTI_SaveStatus>>**. For convenience, this type has been aliased using a **typedef** to the name **RTI_FederateHandleSaveStatusPairVector**. To ensure maximum portability between RTI implementations, federate programmers should use only this **typedef**. With the exception of the name, an **RTI_FederateHandleSaveStatusPairVector** behaves in other respects as does **std::vector<std::pair <RTI_FederateHandle, RTI_SaveStatus>>**. See the discussion in 12.5.2 for further explanation.

RTI_SaveStatus is a type-safe **enum**, very similar to **RTI_ResignAction** described in 12.5.2.5. To facilitate the federate programmer's determination of the particular instance received from the RTI, the following global constants are provided, each of which can be compared to the object returned by the RTI:

— **RTI_NO_SAVE_IN_PROGRESS**

— **RTI_FEDERATE_INSTRUCTED_TO_SAVE**

— **RTI_FEDERATE_SAVING**

— **RTI_FEDERATE_WAITING_FOR_FEDERATION_TO_SAVE**

12.5.2.13 List of joined federates restore status

The list of joined federates and the restore status of each is implemented as a vector of federate designator and restore status pairs. In the C++ API, this corresponds to an **`RTI_vector<RTI_pair< RTI_FederateHandle, RTI_RestoreStatus>>`**. For convenience, this type has been aliased using a **`typedef`** to the name **`RTI_Federate HandleRestoreStatusPairVector`**. To ensure maximum portability between RTI implementations, federate programmers should use only this **`typedef`**. With the exception of the name, an **`RTI_FederateHandleRestoreStatusPairVector`** behaves in other respects as does **`std::vector<std::pair <RTI_FederateHandle, RTI_Restore Status>>`**. See the discussion in 12.5.2 for further explanation.

`RTI_RestoreStatus` is a type-safe **`enum`**, very similar to **`RTI_ResignAction`** described in 12.5.2.5. To facilitate the federate programmer's determination of the particular instance received from the RTI, the following global constants are provided, each of which can be compared to the object returned by the RTI:

- **RTI_NO_RESTORE_IN_PROGRESS**
- **RTI_FEDERATE_RESTORE_REQUEST_PENDING**
- **RTI_FEDERATE_WAITING_FOR_RESTORE_TO_BEGIN**
- **RTI_FEDERATE_PREPARED_TO_RESTORE**
- **RTI_FEDERATE_RESTORING**
- **RTI_FEDERATE_WAITING_FOR_FEDERATION_TO_RESTORE**

12.5.2.14 Passive subscription indicator

The *Subscribe Object Class Attributes*, *Subscribe Interaction Class*, *Subscribe Object Class Attributes With Regions*, and *Subscribe Interaction Class With Regions* services each may optionally provide an **`RTI_bool`** value to indicate whether or not the subscription is active (**`RTI_true`** means the subscription is active). In the C++ API, this is a default parameter to the methods corresponding to each of the services. The default value is **`RTI_true`**.

12.5.2.15 Constrained set of attribute designator and value pairs

The *Update Attribute Values* and *Reflect Attribute Values †* services require a constrained set of attribute designator and value pairs as arguments. A constrained set of attribute designator and value pairs is implemented in the C++ API as an **`RTI_map<RTI_AttributeHandle, RTI_AttributeValue>`**. For convenience, this type has been aliased using a **`typedef`** to the name **`RTI_AttributeHandleValue Map`**. To ensure maximum portability between RTI implementations, federate programmers should use only this **`typedef`**. With the exception of the name, an **`RTI_AttributeHandleValueMap`** behaves in other respects as does **`std::map<RTI_AttributeHandle, RTI_AttributeValue>`**. See the discussion in 12.5.2 for further explanation.

An **`RTI_AttributeValue`** is a class with an interface identical to that of the **`RTI_UserTag`** described above. This class has a constructor that requires the user to provide an untyped pointer (i.e., a **`void *`**) as well as the size in bytes (represented as a **`size_t`**) of the data to which it points. Methods to return these values are provided as well, i.e., **`data()`** and **`size()`**, as is a copy constructor and a copy-assignment operator.

Note that the pointer value returned by the **`data()`** method is not required to have any particular byte alignment. Consequently, naively attempting to simply type cast this pointer and assign its contents to some type may generate an exception in the processor's memory management unit, likely resulting in the crash of

the program. If for some reason it is desirable to perform this type of operation, a function such as **memcpy()** should be used.

12.5.2.16 Constrained set of interaction parameter designator and value pairs

The *Send Interaction*, *Receive Interaction †*, and *Send Interaction With Regions* services require a constrained set of parameter designator and value pairs as arguments. A constrained set of parameter designator and value pairs is implemented in the C++ API as an **RTI_map<RTI_ParameterHandle, RTI_ParameterValue>**. For convenience, this type has been aliased using a **typedef** to the name **RTI_ParameterHandleValueMap**. To ensure maximum portability between RTI implementations, federate programmers should use only this **typedef**. With the exception of the name, an **RTI_ParameterHandleValueMap** behaves in other respects as does **std::map< RTI_ParameterHandle, RTI_ParameterValue>**. See the discussion in the section 12.5.2 for further explanation.

As with **RTI_AttributeValue**, **RTI_ParameterValue** is a class with an interface identical to that of the **RTI_UserTag** described above. This class has a constructor that requires the user to provide an untyped pointer (i.e., a **void ***) as well as the size in bytes (represented as a **size_t**) of the data to which it points. Methods to return these values are provided as well, i.e., **data()** and **size()**, as is a copy constructor and a copy-assignment operator.

NOTE that the pointer value returned by the **data()** method is not required to have any particular byte alignment. Consequently naively attempting to simply type cast this pointer and assign its contents to some type may generate an exception in the processor's memory management unit, likely resulting in the crash of the program. If for some reason it is desirable to perform this type of operation, a function such as **memcpy()** should be used.

12.5.2.17 Message order type

Message order types are represented by a class that implements a type-safe **enum**. This class, named **RTI_OrderType**, is very similar to the **RTI_ResignAction** class described in 12.5.2.5. As a convenience, the following global constants are provided: **RTI_RECEIVE_ORDERED** and **RTI_TIME_STAMP_ORDERED**. These constants can be passed to the RTI or compared to an instance of an **RTI_OrderType** object returned by the RTI.

The **RTI_OrderType** class shares some aspects of the interface to handles (see 12.5.2.4); specifically, the means to encode an order type, reconstitute an encoded order type, and produce a printable form of an order type are provided. Encoding order types and reconstitution of encoded order types work in the same way as for handles. Likewise, producing a printable form works the same as for handles. The **toString()** method is used, and the string produced is not guaranteed to be the same as the name to which the order type corresponds (i.e., the name of the order type as found in the FDD). If the user needs the name form of an order type, the *Get Order Name* service should be used.

12.5.2.18 Transportation type

As with message order types described above, transportation types are represented by a class that implements a type-safe **enum**. This class, named **RTI_TransportationType**, is very similar to the **RTI_ResignAction** class described in 12.5.2.5. As a convenience, the following global constants are provided: **RTI_RELIABLE** and **RTI_BEST_EFFORT**. These constants can be passed to the RTI or compared to an instance of an **RTI_OrderType** object returned by the RTI.

The **RTI_TransportationType** class shares some aspects of the interface to handles (see 12.5.2.4); specifically, the means to encode a transportation type, reconstitute an encoded transportation type, and produce a printable form of a transportation type are provided. Encoding transportation types and

reconstitution of encoded transportation types work in the same way as for handles. Likewise, producing a printable form works the same as for handles. The **`toString()`** method is used, and the string produced is not guaranteed to be the same as the name to which the transportation type corresponds (i.e., the name of the transportation type as found in the FDD). If the user needs the name form of an order type, the *Get Transportation Name* service should be used.

It is possible that some implementations of the RTI will allow additional transportation types. In this case, those RTI implementations shall extend the **`RTI_TransportationType`** class to include these additional transportation types as well as add the corresponding global constants. In no case shall the existing **`RTI_TransportationType`** class or the global constants of its type be reduced or eliminated.

12.5.2.19 Ownership designator

The *Inform Attribute Ownership †* service supplies an ownership designator as one of its arguments. In the C++ API, this corresponds to three distinct methods: **`attributeIsNotOwned()`**, **`attributeIsOwnedByRTI()`**, and **`informAttributeOwnership()`**. All three methods take an **`RTI_ObjectInstanceHandle`** and an **`RTI_AttributeHandle`** as arguments. In addition, **`informAttributeOwnership()`** takes an **`RTI_FederateHandle`** as an argument. Together, these three methods combine to represent the ownership designator of the *Inform Attribute Ownership †* service.

12.5.2.20 Definition indicator

The *Query GALT* and *Query LITS* services are required to return an indicator to define whether or not their return value is valid. Since both services return values of type **`RTI_auto_ptr<RTI_LogicalTime>`**, the definition indicator is trivial: if the returned value is **`NULL`**, the value is undefined (invalid); otherwise, the returned value is defined (valid).

12.5.2.21 Optional message retraction designator

The *Update Attribute Values*, *Send Interaction*, *Send Interaction with Regions*, and *Delete Object Instance* services may return a message retraction designator under certain circumstances. In the C++ API, this is implemented much like the Definition Indicator described above. The methods corresponding to these services always return an **`RTI_auto_ptr<RTI_MessageRetractionHandle>`**. If the returned value is **`NULL`**, then the operation did not generate a message retraction designator; otherwise, the **`RTI_auto_ptr<RTI_MessageRetractionHandle>`** holds the returned message retraction designator.

12.5.2.22 Collection of attribute designator set and region designator set pairs

Several services require the federate to provide the RTI with a collection of attribute designator set and region designator set pairs. In the C++ API, this corresponds to an **`RTI_vector<RTI_ pair<RTI_AttributeHandleSet, RTI_RegionHandleSet>>`**. For convenience, this type has been aliased using a **`typedef`** to the name **`RTI_AttributeHandleSetRegionHandleSetPair Vector`**. To ensure maximum portability between RTI implementations, federate programmers should use only this **`typedef`**. With the exception of the name, an **`RTI_AttributeHandleSetRegionHandle SetPairVector`** behaves in other respects as does **`std::vector<std::pair <RTI_Attribute HandleSet, RTI_RegionHandleSet>>`**. See the discussion in 12.5.2 for further explanation.

12.5.2.23 Logical time, time stamps, and lookahead

In order to support customized implementations of logical time, time stamps, and lookahead, the C++ API defines a group of closely related abstract base classes: **`RTI_LogicalTime`**, **`RTI_Logical TimeInterval`**, **`RTI_LogicalTimeFactory`**, **`RTI_LogicalTimeInterval Factory`**, **`RTI_`**

EncodedLogicalTime, and **RTI_EncodedLogicalTimeInterval**. These classes have been designed to be implemented by the federate developer for use by an RTI implementation.

The **RTI_LogicalTime** class is used to represent logical time and time stamp values. It has methods to compare the relative order of one instance with another. Methods to compare an instance with the initial value (e.g., zero) and with the final value (e.g., infinity) are provided as well as are methods to set an instance to the initial and final values. This generality is provided to allow federate programmers to define arbitrary lower and upper bounds on their implementation of the **RTI_LogicalTime** class. A method to set one instance to the same value as another is provided as well.

Methods to increase and decrease the current value of an instance are provided. However, it should be noted that the argument to these methods is of type **RTI_LogicalTimeInterval**, and not of type **RTI_LogicalTime**.

Finally, methods to convert an instance of an **RTI_LogicalTime** to an **RTI_wstring** are provided, as is a method to encode an **RTI_LogicalTime** into a compact, opaque representation suitable for network transmission. This **encode()** method returns an object of type **RTI_EncodedLogicalTime**. The **RTI_EncodedLogicalTime** class has an interface identical to that of the **RTI_UserTag** class described in 12.5.2.6. However, the **RTI_EncodedLogicalTime** class is an abstract class whose concrete implementation is provided by the federate programmer.

The **RTI_LogicalTimeFactory** has a method to create an **RTI_LogicalTime** instance whose value is the user-defined initial value. Additionally, a **decode()** method is provided that converts an **RTI_EncodedLogicalTime** into its corresponding **RTI_LogicalTime**.

The **RTI_LogicalTimeInterval** class is analogous to the **RTI_LogicalTime** class, but it is used to represent lookahead values. The chief differences in its interface are that there is no concept of a "final" or "largest possible" **RTI_LogicalTimeInterval** instance; consequently, there are only methods to set an **RTI_LogicalTimeInterval** instance to zero and to test an instance to determine if it is zero. Also, one instance of an **RTI_LogicalTimeInterval** may be subtracted from another, but not added. The **RTI_LogicalTimeIntervalFactory** and the **RTI_EncodedLogicalTimeInterval** classes are completely analogous to their **RTI_LogicalTime** counterparts.

However, the **RTI_LogicalTimeIntervalFactory** does have one more method than does its **RTI_LogicalTimeFactory** analogue. The additional method, **epsilon()**, returns a constant reference to the smallest nonzero **RTI_LogicalTimeInterval** that is possible with the particular implementation. This value is used by an RTI implementation to support time management services for federates with zero lookahead. The value of epsilon is intended to be constant; therefore, if **anRTI_LogicalTimeIntervalFactory** is an instance of an **RTI_LogicalTimeInterval Factory**, the expression **anRTI_Logical TimeIntervalFactory.epsilon().isEqualTo(anRTI_LogicalTimeIntervalFactory.epsilon())** must always return **RTI_true**.

Implementers of the C++ API shall provide, as part of their implementation, implementations of the logical time classes, as follows:

— An implementation of the **RTI_LogicalTime** class called **RTI_Integer64Time**. This implementation shall use a 64-bit two's complement signed integer type as its underlying representation of time. It shall use 0 as the value of its initial time and $2^{63} - 1$ as its final time. Those methods that are defined to return or accept an **RTI_LogicalTime** reference shall actually return or expect a reference to **RTI_Integer64Time**. Those methods defined as returning or accepting a **RTI_LogicalTimeInterval** reference shall return or expect a reference to **RTI_Integer64TimeInterval** (see next paragraph). Those methods defined as returning an **RTI_EncodedLogicalTime** reference shall return a reference to **RTI_EncodedInteger64Time** (see next paragraph).

— An implementation of **RTI_LogicalTimeFactory** called **RTI_Integer64TimeFactory**. All of the methods of this implementation shall return references to **RTI_Integer64Time** as defined above. Those methods defined as accepting an **RTI_EncodedLogicalTime** reference shall expect a reference to **RTI_EncodedInteger64Time** (see next paragraph).

— An implementation of **RTI_EncodedLogicalTime** called **RTI_EncodedInteger64Time**.

— An implementation of **RTI_LogicalTimeInterval** called **RTI_Integer64TimeInterval**. This implementation shall use a 64-bit two's complement signed integer type as its underlying representation of a time interval. Its value of epsilon shall be 1. Those methods defined as returning or accepting an **RTI_LogicalTimeInterval** reference shall return or expect a reference to **RTI_Integer64TimeInterval**. Those methods defined as returning an **RTI_EncodedLogicalTimeInterval** reference shall return a reference to **RTI_EncodedInteger64TimeInterval** (see next paragraph).

— An implementation of **RTI_LogicalTimeIntervalFactory** called **RTI_Integer64TimeIntervalFactory**. All of the methods of this implementation shall return references to **RTI_Integer64TimeInterval** as defined above. Those methods defined as accepting an **RTI_EncodedLogicalTimeInterval** reference shall expect a reference to **RTI_Encoded Integer64TimeInterval** (see next paragraph).

— An implementation of **RTI_EncodedLogicalTimeInterval** called **RTI_Encoded Integer64TimeInterval**.

12.5.2.24 Dimension upper bound and range lower and upper bounds

Several DDM-related services allow bounds to be get and set. The *Get Dimension Upper Bound* service gets the upper bound of a particular dimension. The remaining services are used to get and set the bounds of ranges of a region. All bounds are represented in the C++ API using an **unsigned** integer.

Range lower and upper bounds are gathered into an **RTI_RangeBounds** object.

12.5.2.25 Wall-clock time

The *Evoke Callback* and *Evoke Multiple Callbacks* services take arguments specifying durations of wall-clock time expressed in seconds. Wall-clock time is represented in the C++ API using a **double**.

12.5.3 Memory ownership semantics

Many programming errors in C++ can be traced back to the improper use of the **delete** operator. Frequently, this is a result of poor programming practice, rather than an inherent limitation of the C++ programming language. Considerable effort has been made to prevent memory corruption and leaks resulting from the misuse of the **delete** operator in the C++ API.

In particular, the federate programmer never explicitly deletes objects exchanged with the RTI. Simple objects are passed by value. More complex objects are either passed by constant reference (i.e., **T const &**) or are passed as an **RTI_auto_ptr<T>** object. In all three cases, the C++ compiler prevents the federate programmer from calling the delete operator on the object.

As explained in 12.5.2, the **RTI_auto_ptr<T>** implements the strong memory ownership and transfer semantics of the **std::auto_ptr<T>** class[10].

[10] See [B10] for a detailed description.

Annex A

(normative)

Ada 95 application programmer's interface

A.1 API

The following file shall comprise the Ada 95 HLA API. The text **<some_supplier>** is a place holder for an RTI implementor to supply their specific name to the file.

```
-- File: <some_supplier>_rti.ads
-- ===========================================================================
-- IEEE 1516.1 High Level Architecture Federate Interface Specification
-- Ada 95 API
-- ===========================================================================


with Ada.Streams;
with Ada.Strings.Wide_Unbounded;

package <Some_Supplier>_RTI is

--
-- RTI version
--

HLA_Versiosn : constant Wide_String := "1516.1.5";

--
-- Data interchange subtypes
--

subtype Encoded_Value is Ada.Streams.Stream_Element_Array;
subtype User_Supplied_Tag is Ada.Streams.Stream_Element_Array;

--
-- Exceptions
--
 Asynchronous_Delivery_Already_Disabled,
 Asynchronous_Delivery_Already_Enabled,
 Attribute_Acquisition_Was_Not_Canceled,
 Attribute_Acquisition_Was_Not_Requested,
 Attribute_Already_Being_Acquired,
 Attribute_Already_Being_Divested,
 Attribute_Already_Owned,
 Attribute_Divestiture_Was_Not_Requested,
 Attribute_Not_Defined,
 Attribute_Not_Owned,
 Attribute_Not_Published,
 Attribute_Not_Recognized,
 Attribute_Not_Subscribed,
 Attribute_Relevance_Advisory_Switch_Is_Off,
 Attribute_Relevance_Advisory_Switch_Is_On,
 Attribute_Scope_Advisory_Switch_Is_Off,
 Attribute_Scope_Advisory_Switch_Is_On,
 Bad_Initialization_Parameter,
 Could_Not_Discover,
 Could_Not_Initiate_Restore,
 Could_Not_Open_FDD,
 Delete_Privilege_Not_Held,
```



```
  Error_Reading_FDD,
  Federate_Unable_To_Use_Time,
  Federate_Already_Execution_Member,
  Federate_Has_Not_Begun_Save,
  Federate_Internal_Error,
  Federate_Not_Execution_Member,
  Federate_Owns_Attributes,
  Federate_Service_Invocations_Are_Being_Reported_Via_MOM,
  Federates_Currently_Joined,
  Federation_Execution_Already_Exists,
  Federation_Execution_Does_Not_Exist,
  Illegal_Name,
  In_Time_Advancing_State,
  Initialize_Never_Invoked,
  Initialize_Previously_Invoked,
  Interaction_Class_Not_Defined,
  Interaction_Class_Not_Published,
  Interaction_Class_Not_Recognized,
  Interaction_Class_Not_Subscribed,
  Interaction_Parameter_Not_Defined,
  Interaction_Parameter_Not_Recognized,
  Interaction_Relevance_Advisory_Switch_Is_Off,
  Interaction_Relevance_Advisory_Switch_Is_On,
  Invalid_Attribute_Handle,
  Invalid_Dimension_Handle,
  Invalid_Federate_Handle,
  Invalid_Interaction_Class_Handle,
  Invalid_Logical_Time,
  Invalid_Lookahead,
  Invalid_Object_Class_Handle,
  Invalid_Order_Name,
  Invalid_Order_Type,
  Invalid_Parameter_Handle,
  Invalid_Range_Bound,
  Invalid_Region,
  Invalid_Region_Context,
  Invalid_Resign_Action,
  Invalid_Message_Retraction_Handle,
  Invalid_Transportation_Name,
  Invalid_Transportation_Type,
  Joined_Federate_Is_Not_In_Time_Advancing_State,
  Logical_Time_Already_Passed,
  Message_Can_No_Longer_Be_Retracted,
  Name_Not_Found,
  No_Request_To_Enable_Time_Constrained_Was_Pending,
  No_Request_To_Enable_Time_Regulation_Was_Pending,
  Object_Class_Not_Defined,
  Object_Class_Not_Published,
  Object_Class_Not_Recognized,
  Object_Class_Relevance_Advisory_Switch_Is_Off,
  Object_Class_Relevance_Advisory_Switch_Is_On,
  Object_Instance_Name_In_Use,
  Object_Instance_Name_Not_Reserved,
  Object_Instance_Not_Known,
  Ownership_Acquisition_Pending,
  RTI_Internal_Error,
  Region_Does_Not_Contain_Specified_Dimension,
  Region_In_Use_For_Update_Or_Subscription,
  Region_Not_Created_By_This_Federate,
  Request_For_Time_Constrained_Pending,
  Request_For_Time_Regulation_Pending,
  Restore_In_Progress,
  Restore_Not_Requested,
  Restore_Not_In_Progress,
  Save_In_Progress,
  Save_Not_In_Progress,
  Save_Not_Initiated,
  Some_Federate_Is_Joined_To_An_Execution,
  Specified_Save_Label_Does_Not_Exist,
  Synchronization_Point_Label_Not_Unique,
  Synchronization_Point_Not_Announced,
  Synchronization_Set_Member_Not_Joined,
```



```
 Time_Constrained_Already_Enabled,
 Time_Constrained_Is_Not_Enabled,
 Time_Regulation_Already_Enabled,
 Time_Regulation_Is_Not_Enabled,
 Unable_To_Perform_Save,
 Unknown_Name : exception;
--
-- HLA_DDM_Coordinate and Range_Bounds types
--

 type HLA_DDM_Coordinate is mod 2**32;

 type Range_Bounds is record
  Lower : HLA_DDM_Coordinate;
  Upper : HLA_DDM_Coordinate;
 end record;

--
-- Types of Transportation and services
--

 -- This enumeration defines the predefined types of transportation.
 -- An implementation is allowed to define new types of transportation.
 -- None of the predefined types of transportation shall be eliminated.

 type Transportation_Type is (
  Reliable,
  Best_Effort);

 function Encode (
  Transportation : in Transportation_Type)
 return Encoded_Value;

 function Decode (
  Value : in Encoded_Value)
 return Transportation_Type;

  -- exception:
  -- Data_Error

--
-- Types of Order and services
--

 type Order_Type is (
  Receive_Ordered,
  Time_Stamp_Ordered);

 function Encode (
  Order : in Order_Type)
 return Encoded_Value;

 function Decode (
  Value : in Encoded_Value)
 return Order_Type;
  -- exception:
  -- Data_Error

--
-- Logical_Time_Interval type and services
--

 type Logical_Time_Interval is abstract tagged null record;

 -- These services express the formal needs of the RTI about the Logical Time
 -- Interval. An implementation of this type shall have the capability to
 -- represent negative values.

 procedure Set_To_Zero (
  Time_Interval : out Logical_Time_Interval) is abstract;

 procedure Set_To_Epsilon (
```

```
   Time_Interval : out Logical_Time_Interval) is abstract;

  procedure Set_To (
   Time_Interval : out Logical_Time_Interval;
   Other : in Logical_Time_Interval) is abstract;

  function "-" (
   L, R : in Logical_Time_Interval)
  return Logical_Time_Interval is abstract;

   -- exception:
   -- Constraint_Error

  function Is_Zero (
   Time_Interval : in Logical_Time_Interval)
  return Boolean is abstract;

  function Is_Epsilon (
   Time_Interval : in Logical_Time_Interval)
  return Boolean is abstract;

  function "<=" (
   L, R : in Logical_Time_Interval)
  return Boolean is abstract;

  function "<" (
   L, R : in Logical_Time_Interval)
  return Boolean is abstract;

  function ">=" (
   L, R : in Logical_Time_Interval)
  return Boolean is abstract;

  function ">" (
   L, R : in Logical_Time_Interval)
  return Boolean is abstract;

  function "=" (
   L, R : in Logical_Time_Interval)
  return Boolean is abstract;

  function Image (
   Time_Interval : in Logical_Time_Interval)
  return Wide_String is abstract;

  function Encode (
   Time_Interval : in Logical_Time_Interval)
  return Encoded_Value is abstract;

  function Decode (
   Value : in Encoded_Value)
  return Logical_Time_Interval is abstract;

   -- exception:
   -- Data_Error

--
-- Logical_Time type and services
--

  type Logical_Time is abstract tagged null record;
  -- These services express the formal needs of the RTI about the Logical Time.

  procedure Set_To_Initial (
   Time : out Logical_Time) is abstract;

  procedure Set_To_Final (
   Time : out Logical_Time) is abstract;

  procedure Set_To (
   Time : out Logical_Time;
   Other : in Logical_Time) is abstract;
```



```
 function "+" (
  L : in Logical_Time;
  R : in Logical_Time_Interval'Class)
 return Logical_Time is abstract;

  -- exception:
  -- Constraint_Error

 function "-" (
  L : in Logical_Time;
  R : in Logical_Time_Interval'Class)
 return Logical_Time is abstract;

  -- exception:
  -- Constraint_Error

 function "-" (
  L, R : in Logical_Time)
 return Logical_Time_Interval'Class is abstract;

 function Is_Initial (
  Time : in Logical_Time)
 return Boolean is abstract;

 function Is_Final (
  Time : in Logical_Time)
 return Boolean is abstract;

 function "<=" (
  L, R : in Logical_Time)
 return Boolean is abstract;

 function "<" (
  L, R : in Logical_Time)
 return Boolean is abstract;

 function ">=" (
  L, R : in Logical_Time)
 return Boolean is abstract;

 function ">" (
  L, R : in Logical_Time)
 return Boolean is abstract;

 function "=" (
  L, R : in Logical_Time)
 return Boolean is abstract;

 function Image (
  Time : in Logical_Time)
 return Wide_String is abstract;

 function Encode (
  Time : in Logical_Time)
 return Encoded_Value is abstract;

 function Decode (
  Value : in Encoded_Value)
 return Logical_Time is abstract;

  -- exception:
  -- Data_Error

--
-- Handle types and services
--

 -- Implementation requirement:
 -- An instance of handle type can be safely used, stored and duplicated
 -- by the federate along the federation execution.

 type Federate_Handle is private;
```

```
 type Object_Class_Handle is private;
 type Interaction_Class_Handle is private;
 type Object_Instance_Handle is private;
 type Attribute_Handle is private;
 type Parameter_Handle is private;
 type Dimension_Handle is private;
 type Message_Retraction_Handle is private;
 type Region_Handle is private;

 function "<" (L, R : in Federate_Handle) return Boolean;
 function "<" (L, R : in Object_Class_Handle) return Boolean;
 function "<" (L, R : in Interaction_Class_Handle) return Boolean;
 function "<" (L, R : in Object_Instance_Handle) return Boolean;
 function "<" (L, R : in Attribute_Handle) return Boolean;
 function "<" (L, R : in Parameter_Handle) return Boolean;
 function "<" (L, R : in Dimension_Handle) return Boolean;
 function "<" (L, R : in Message_Retraction_Handle) return Boolean;
 function "<" (L, R : in Region_Handle) return Boolean;

 function Image (Handle : in Federate_Handle) return Wide_String;
 function Image (Handle : in Object_Class_Handle) return Wide_String;
 function Image (Handle : in Interaction_Class_Handle) return Wide_String;
 function Image (Handle : in Object_Instance_Handle) return Wide_String;
 function Image (Handle : in Attribute_Handle) return Wide_String;
 function Image (Handle : in Parameter_Handle) return Wide_String;
 function Image (Handle : in Dimension_Handle) return Wide_String;
 function Image (Handle : in Message_Retraction_Handle) return Wide_String;
 function Image (Handle : in Region_Handle) return Wide_String;

 function Encode (Handle : in Federate_Handle) return Encoded_Value;
 function Encode (Handle : in Object_Class_Handle) return Encoded_Value;
 function Encode (Handle : in Interaction_Class_Handle) return Encoded_Value;
 function Encode (Handle : in Object_Instance_Handle) return Encoded_Value;
 function Encode (Handle : in Attribute_Handle) return Encoded_Value;
 function Encode (Handle : in Parameter_Handle) return Encoded_Value;
 function Encode (Handle : in Dimension_Handle) return Encoded_Value;
 function Encode (Handle : in Message_Retraction_Handle) return Encoded_Value;
 function Encode (Handle : in Region_Handle) return Encoded_Value;

 function Decode (Value : in Encoded_Value) return Federate_Handle;
 function Decode (Value : in Encoded_Value) return Object_Class_Handle;
 function Decode (Value : in Encoded_Value) return Interaction_Class_Handle;
 function Decode (Value : in Encoded_Value) return Object_Instance_Handle;
 function Decode (Value : in Encoded_Value) return Attribute_Handle;
 function Decode (Value : in Encoded_Value) return Parameter_Handle;
 function Decode (Value : in Encoded_Value) return Dimension_Handle;
 function Decode (Value : in Encoded_Value) return Message_Retraction_Handle;
 function Decode (Value : in Encoded_Value) return Region_Handle;

  -- exception for Decode functions:
  -- Data_Error

--
-- Sets of Handles and services
--

 -- These sets ensure that contained handles are unique.
 -- Predefined operations for sets are comparison "=" and assignment ":=".
 -- The To_Array functions return an array of handles sorted by increasing
 -- order.

 type Federate_Handle_Array is array (Positive range <>) of Federate_Handle;
 type Federate_Handle_Set (<>) is private; -- insures unicity
 Empty_Federate_Handle_Set : constant Federate_Handle_Set;

 function To_Set (Handles : in Federate_Handle_Array)
  return Federate_Handle_Set;
 function To_Array (Set : in Federate_Handle_Set)
  return Federate_Handle_Array;
 function "-" (L, R : in Federate_Handle_Set)
  return Federate_Handle_Set;
 function Intersection (L, R : in Federate_Handle_Set)
```



```
   return Federate_Handle_Set;


  type Attribute_Handle_Array is array (Positive range <>) of Attribute_Handle;
  type Attribute_Handle_Set (<>) is private; -- insures unicity
  Empty_Attribute_Handle_Set : constant Attribute_Handle_Set;

  function To_Set (Handles : in Attribute_Handle_Array)
   return Attribute_Handle_Set;
  function To_Array (Set : in Attribute_Handle_Set)
   return Attribute_Handle_Array;
  function "-" (L, R : in Attribute_Handle_Set)
   return Attribute_Handle_Set;
  function Intersection (L, R : in Attribute_Handle_Set)
   return Attribute_Handle_Set;


  type Dimension_Handle_Array is array (Positive range <>) of Dimension_Handle;
  type Dimension_Handle_Set (<>) is private; -- insures unicity
  Empty_Dimension_Handle_Set : constant Dimension_Handle_Set;

  function To_Set (Handles : in Dimension_Handle_Array)
   return Dimension_Handle_Set;
  function To_Array (Set : in Dimension_Handle_Set)
   return Dimension_Handle_Array;
  function "-" (L, R : in Dimension_Handle_Set)
   return Dimension_Handle_Set;
  function Intersection (L, R : in Dimension_Handle_Set)
   return Dimension_Handle_Set;


  type Region_Handle_Array is array (Positive range <>) of Region_Handle;
  type Region_Handle_Set (<>) is private; -- insures unicity
  Empty_Region_Handle_Set : constant Region_Handle_Set;

  function To_Set (Handles : in Region_Handle_Array)
   return Region_Handle_Set;
  function To_Array (Set : in Region_Handle_Set)
   return Region_Handle_Array;
  function "-" (L, R : in Region_Handle_Set)
   return Region_Handle_Set;
  function Intersection (L, R : in Region_Handle_Set)
   return Region_Handle_Set;

--
-- Attributes_Regions_Pairs type and services
--

  type Attributes_Regions_Pairs (Capacity : Natural) is limited private;

  -- Implementation requirement:
  -- No storage associated with an Attributes_Regions_Pairs
  -- object shall be lost upon scope exit.

  -- Sets the Attributes and Regions at a given Rank. No storage shall be lost.
  procedure Set_Attributes_And_Regions (
   Pairs : in out Attributes_Regions_Pairs;
   Rank : in Positive;
   Attributes : in Attribute_Handle_Set;
   Regions : in Region_Handle_Set);

   -- exception:
   -- Constraint_Error if Rank > Capacity.

  -- Sets the Attributes at a given Rank. No storage shall be lost.
  procedure Set_Attributes (
   Pairs : in out Attributes_Regions_Pairs;
   Rank : in Positive;
   Attributes : in Attribute_Handle_Set);

   -- exception:
   -- Constraint_Error if Rank > Capacity.
```

```
   -- Sets the Regions at a given Rank. No storage shall be lost.
   procedure Set_Regions (
    Pairs : in out Attributes_Regions_Pairs;
    Rank : in Positive;
    Regions : in Region_Handle_Set);
    -- exception:
    -- Constraint_Error if Rank > Capacity.

   -- Returns the Attributes at a given Rank.
   -- Returns an empty set if not previously set.
   function Get_Attributes (
    Pairs : in Attributes_Regions_Pairs;
    Rank : in Positive)
   return Attribute_Handle_Set;

    -- exception:
    -- Constraint_Error if Rank > Capacity.

   -- Returns the Regions at a given Rank.
   -- Returns an empty set if not previously set.
   function Get_Regions (
    Pairs : in Attributes_Regions_Pairs;
    Rank : in Positive)
   return Region_Handle_Set;

    -- exception:
    -- Constraint_Error if Rank > Capacity.

--
-- Parameter_Handle_Value_Map type and services
--

   type Parameter_Handle_Value_Map is limited private;

   -- Implementation requirement:
   -- No storage associated with a Parameter_Handle_Value_Map object
   -- shall be lost upon scope exit.

   -- Adds a Parameter_Handle, Value pair.
   procedure Add (
    Map : in out Parameter_Handle_Value_Map;
    Parameter : in Parameter_Handle;
    Value : in Encoded_Value);

   -- Empties the map. No storage shall be lost.
   procedure Clear (
    Map : in out Parameter_Handle_Value_Map);

   -- Returns the number of pairs of the map.
   function Cardinality (
    Map : in Parameter_Handle_Value_Map)
   return Natural;

   -- Returns the Value at a given Rank.
   function Get_Value (
    Map : in Parameter_Handle_Value_Map;
    Rank : in Positive)
   return Encoded_Value;

    -- exception:
    -- Constraint_Error if Rank > Cardinality.

   -- Returns the Value for a given Parameter_Handle.
   -- Returns an empty Value if not found.
   function Get_Value (
    Map : in Parameter_Handle_Value_Map;
    Handle : in Parameter_Handle)
   return Encoded_Value;

   -- Returns the Parameter_Handle at a given Rank.
   function Get_Parameter (
    Map : in Parameter_Handle_Value_Map;
```



```
   Rank : in Positive)
  return Parameter_Handle;

   -- exception:
   -- Constraint_Error if Rank > Cardinality.

--
-- Attribute_Handle_Value_Map type and services
--

  type Attribute_Handle_Value_Map is limited private;

  -- Implementation requirement:
  -- No storage associated with an Attribute_Handle_Value_Map object
  -- shall be lost upon scope exit.

  -- Adds an Attribute_Handle, Value pair.
  procedure Add (
   Map : in out Attribute_Handle_Value_Map;
   Attribute : in Attribute_Handle;
   Value : in Encoded_Value);

  -- Empties the map. No storage shall be lost.
  procedure Clear (
   Map : in out Attribute_Handle_Value_Map);

  -- Returns the number of pairs of the map.
  function Cardinality (
   Map : in Attribute_Handle_Value_Map)
  return Natural;

  -- Returns the Value at a given Rank.
  function Get_Value (
   Map : in Attribute_Handle_Value_Map;
   Rank : in Positive)
  return Encoded_Value;

   -- exception:
   -- Constraint_Error if Rank > Cardinality.

  -- Returns the Value for a given Attribute_Handle.
  -- Returns an empty Value if not found.
  function Get_Value (
   Map : in Attribute_Handle_Value_Map;
   Handle : in Attribute_Handle)
  return Encoded_Value;

  -- Returns the Attribute_Handle at a given Rank.
  function Get_Attribute (
   Map : in Attribute_Handle_Value_Map;
   Rank : in Positive)
  return Attribute_Handle;

   -- exception:
   -- Constraint_Error if Rank > Cardinality.

  -- Returns the Attribute_Handle_Set of the map.
  function Get_Attributes (
   Map : in Attribute_Handle_Value_Map)
  return Attribute_Handle_Set;

--
-- RTI_Ambassador type
--

  type RTI_Ambassador is limited private;

  -- Implementation requirement:
  -- No storage associated with an RTI_Ambassador object shall be lost
  -- upon scope exit.

--
```

```
-- Federate_Ambassador type
--

 type Federate_Ambassador is abstract tagged limited private;

 -- Implementation requirement:
 -- No storage associated with a Federate_Ambassador object shall be lost
 -- upon scope exit.

--
-- Federation Management services
--

 -- 4.2
 procedure Create_Federation_Execution (
  Ambassador : in RTI_Ambassador;
  Federation_Execution_Name : in Wide_String;
  Full_Path_To_FDD : in Wide_String);

  -- exceptions:
  -- Federation_Execution_Already_Exists
  -- Could_Not_Open_FDD
  -- Error_Reading_FDD
  -- RTI_Internal_Error

 -- 4.3
 procedure Destroy_Federation_Execution (
  Ambassador : in RTI_Ambassador;
  Federation_Execution_Name : in Wide_String);

  -- exceptions:
  -- Federates_Currently_Joined
  -- Federation_Execution_Does_Not_Exist
  -- RTI_Internal_Error

 -- 4.4
 procedure Join_Federation_Execution (
  Ambassador : in RTI_Ambassador;
  Federate_Type : in Wide_String;
  Federation_Execution_Name : in Wide_String;
  An_Intance_Of_Federate_Ambassador : in Federate_Ambassador'Class;
  An_Intance_Of_Logical_Time : in Logical_Time'Class;
  An_Intance_Of_Logical_Time_Interval : in Logical_Time_Interval'Class;
  Federate : out Federate_Handle);

  -- exceptions:
  -- Federate_Already_Execution_Member
  -- Federation_Execution_Does_Not_Exist
  -- Save_In_Progress
  -- Restore_In_Progress
  -- RTI_Internal_Error

 -- 4.5
 type Resign_Action is (
  Unconditionally_Divest_Attributes,
  Delete_Objects,
  Cancel_Pending_Ownership_Acquisitions,
  Delete_Objects_Then_Divest,
  Cancel_Then_Delete_Then_Divest,
  No_Action);

 procedure Resign_Federation_Execution (
  Ambassador : in RTI_Ambassador;
  Action : in Resign_Action);

  -- exceptions:
  -- Ownership_Acquisition_Pending
  -- Federate_Owns_Attributes
  -- Invalid_Resign_Action
  -- Federate_Not_Execution_Member
  -- RTI_Internal_Error
```



```
-- 4.6
procedure Register_Federation_Synchronization_Point (
 Ambassador : in RTI_Ambassador;
 Label : in Wide_String;
 Tag : in User_Supplied_Tag);

 -- exceptions:
 -- Federate_Not_Execution_Member
 -- Save_In_Progress
 -- Restore_In_Progress
 -- RTI_Internal_Error

procedure Register_Federation_Synchronization_Point (
 Ambassador : in RTI_Ambassador;
 Label : in Wide_String;
 Tag : in User_Supplied_Tag;
 Synchronization_Set : in Federate_Handle_Set);

 -- exceptions:
 -- Federate_Not_Execution_Member
 -- Save_In_Progress
 -- Restore_In_Progress
 -- RTI_Internal_Error

-- 4.7 +
procedure Synchronization_Point_Registration_Succeeded (
 Ambassador : in Federate_Ambassador;
 Label : in Wide_String) is abstract;

 -- exceptions:
 -- Federate_Internal_Error

 type Synchronization_Failure_Reason is (
 Synchronization_Point_Label_Is_Not_Unique,
 Synchronization_Set_Member_Is_Not_Joined);

procedure Synchronization_Point_Registration_Failed (
 Ambassador : in Federate_Ambassador;
 Label : in Wide_String;
 Reason: in Synchronization_Failure_Reason) is abstract;

 -- exceptions:
 -- Federate_Internal_Error

-- 4.8 +
procedure Announce_Synchronization_Point (
 Ambassador : in Federate_Ambassador;
 Label : in Wide_String;
 Tag : in User_Supplied_Tag) is abstract;

 -- exceptions:
 -- Federate_Internal_Error

-- 4.9
procedure Synchronization_Point_Achieved (
 Ambassador : in RTI_Ambassador;
 Label : in Wide_String);

 -- exceptions:
 -- Synchronization_Point_Not_Announced
 -- Federate_Not_Execution_Member
 -- Save_In_Progress
 -- Restore_In_Progress
 -- RTI_Internal_Error

-- 4.10 +
procedure Federation_Synchronized (
 Ambassador : in Federate_Ambassador;
 Label : in Wide_String) is abstract;

 -- exceptions:
 -- Federate_Internal_Error
```

```
-- 4.11
procedure Request_Federation_Save (
 Ambassador : in RTI_Ambassador;
 Label : in Wide_String);

 -- exceptions:
 -- Federate_Not_Execution_Member
 -- Save_In_Progress
 -- Restore_In_Progress
 -- RTI_Internal_Error

procedure Request_Federation_Save (
 Ambassador : in RTI_Ambassador;
 Label : in Wide_String;
 Time : in Logical_Time'Class);

 -- exceptions:
 -- Logical_Time_Already_Passed
 -- Invalid_Logical_Time
 -- Federate_Unable_To_Use_Time
 -- Federate_Not_Execution_Member
 -- Save_In_Progress
 -- Restore_In_Progress
 -- RTI_Internal_Error

-- 4.12 +
procedure Initiate_Federate_Save (
 Ambassador : in Federate_Ambassador;
 Label : in Wide_String) is abstract;

 -- exceptions:
 -- Unable_To_Perform_Save
 -- Federate_Internal_Error

procedure Initiate_Federate_Save (
 Ambassador : in Federate_Ambassador;
 Label : in Wide_String;
 Time_Stamp : in Logical_Time'Class) is abstract;

 -- exceptions:
 -- Invalid_Logical_Time
 -- Unable_To_Perform_Save
 -- Federate_Internal_Error

-- 4.13
procedure Federate_Save_Begun (
 Ambassador : in RTI_Ambassador);

 -- exceptions:
 -- Save_Not_Initiated
 -- Federate_Not_Execution_Member
 -- Restore_In_Progress
 -- RTI_Internal_Error

-- 4.14
procedure Federate_Save_Complete (
 Ambassador : in RTI_Ambassador);

 -- exceptions:
 -- Federate_Has_Not_Begun_Save
 -- Federate_Not_Execution_Member
 -- Restore_In_Progress
 -- RTI_Internal_Error

procedure Federate_Save_Not_Complete (
 Ambassador : in RTI_Ambassador);

 -- exceptions:
 -- Federate_Has_Not_Begun_Save
 -- Federate_Not_Execution_Member
 -- Restore_In_Progress
 -- RTI_Internal_Error
```



```
-- 4.15 +
procedure Federation_Saved (
 Ambassador : in Federate_Ambassador) is abstract;

 -- exceptions:
 -- Federate_Internal_Error

 type Save_Failure_Reason is (
 RTI_Unable_To_Save,
 Federate_Reported_Failure,
 Federate_Resigned,
 RTI_Detected_Failure,
 Save_Time_Cannot_Be_Honored);

procedure Federation_Not_Saved (
 Ambassador : in Federate_Ambassador;
 Reason: in Save_Failure_Reason) is abstract;

 -- exceptions:
 -- Federate_Internal_Error

-- 4.16
procedure Query_Federation_Save_Status (
 Ambassador : in RTI_Ambassador);

 -- exceptions:
 -- Federate_Not_Execution_Member
 -- Save_Not_In_Progress
 -- Restore_In_Progress
 -- RTI_Internal_Error

-- 4.17 +
type Save_Status is (
 No_Save_In_Progress,
 Federate_Instructed_To_Save,
 Federate_Saving,
 Federate_Waiting_For_Federation_To_Save);

 type Federate_Save_Status is record
 Federate : Federate_Handle;
 Status : Save_Status;
end record;

type Federate_Save_Status_List is array (Positive range <>)
 of Federate_Save_Status;

procedure Federation_Save_Status_Response (
 Ambassador : in Federate_Ambassador;
 Joined_Federates_Save_Status : in Federate_Save_Status_List) is abstract;

 -- exceptions:
 -- Federate_Internal_Error

-- 4.18
procedure Request_Federation_Restore (
 Ambassador : in RTI_Ambassador;
 Label : in Wide_String);

 -- exceptions:
 -- Federate_Not_Execution_Member
 -- Save_In_Progress
 -- Restore_In_Progress
 -- RTI_Internal_Error

-- 4.19 +
procedure Request_Federation_Restore_Succeeded (
 Ambassador : in Federate_Ambassador;
 Label : in Wide_String) is abstract;

 -- exceptions:
 -- Federate_Internal_Error
```

```
 procedure Request_Federation_Restore_Failed (
  Ambassador : in Federate_Ambassador;
  Label : in Wide_String) is abstract;

  -- exceptions:
  -- Federate_Internal_Error

 -- 4.20 +
 procedure Federation_Restore_Begun (
  Ambassador : in Federate_Ambassador) is abstract;

  -- exceptions:
  -- Federate_Internal_Error

 -- 4.21 +
 procedure Initiate_Federate_Restore (
  Ambassador : in Federate_Ambassador;
  Label : in Wide_String;
  Federate : in Federate_Handle) is abstract;

  -- exceptions:
  -- Specified_Save_Label_Does_Not_Exist
  -- Could_Not_Initiate_Restore
  -- Federate_Internal_Error

 -- 4.22
 procedure Federate_Restore_Complete (
  Ambassador : in RTI_Ambassador);

  -- exceptions:
  -- Restore_Not_Requested
  -- Federate_Not_Execution_Member
  -- Save_In_Progress
  -- RTI_Internal_Error

 procedure Federate_Restore_Not_Complete (
  Ambassador : in RTI_Ambassador);

  -- exceptions:
  -- Restore_Not_Requested
  -- Federate_Not_Execution_Member
  -- Save_In_Progress
  -- RTI_Internal_Error

 -- 4.23 +
 procedure Federation_Restored (
  Ambassador : in Federate_Ambassador) is abstract;

  -- exceptions:
  -- Federate_Internal_Error

 type Restore_Failure_Reason is (
  RTI_Unable_To_Restore,
  Federate_Reported_Failure,
  Federate_Resigned,
  RTI_Detected_Failure);

 procedure Federation_Not_Restored (
  Ambassador : in Federate_Ambassador;
  Reason: in Restore_Failure_Reason) is abstract;

  -- exceptions:
  -- Federate_Internal_Error

 -- 4.24
 procedure Query_Federation_Restore_Status (
  Ambassador : in RTI_Ambassador);

  -- exceptions:
  -- Federate_Not_Execution_Member
  -- Save_In_Progress
  -- Restore_Not_In_Progress
```



```
   -- RTI_Internal_Error

  -- 4.25 +
  type Restore_Status is (
   No_Restore_In_Progress,
   Federate_Restore_Request_Pending,
   Federate_Waiting_For_Restore_To_Begin,
   Federate_Prepared_To_Restore,
   Federate_Restoring,
   Federate_Waiting_For_Federation_To_Restore);

  type Federate_Restore_Status is record
   Federate : Federate_Handle;
   Status : Restore_Status;
  end record;

  type Federate_Restore_Status_List is array (Positive range <>)
   of Federate_Restore_Status;

  procedure Federation_Restore_Status_Response (
   Ambassador : in Federate_Ambassador;
   Joined_Federates_Restore_Status : in Federate_Restore_Status_List)
   is abstract;

   -- exceptions:
   -- Federate_Internal_Error

--
-- Declaration Management services
--

  -- 5.2
  procedure Publish_Object_Class_Attributes (
   Ambassador : in RTI_Ambassador;
   Object_Class : in Object_Class_Handle;
   Attributes : in Attribute_Handle_Set);

   -- exceptions:
   -- Object_Class_Not_Defined
   -- Attribute_Not_Defined
   -- Federate_Not_Execution_Member
   -- Save_In_Progress
   -- Restore_In_Progress
   -- RTI_Internal_Error

  -- 5.3
  procedure Unpublish_Object_Class (
   Ambassador : in RTI_Ambassador;
   Object_Class : in Object_Class_Handle);

   -- exceptions:
   -- Object_Class_Not_Defined
   -- Ownership_Acquisition_Pending
   -- Federate_Not_Execution_Member
   -- Save_In_Progress
   -- Restore_In_Progress
   -- RTI_Internal_Error

  procedure Unpublish_Object_Class_Attributes (
   Ambassador : in RTI_Ambassador;
   Object_Class : in Object_Class_Handle;
   Attributes : in Attribute_Handle_Set);

   -- exceptions:
   -- Object_Class_Not_Defined
   -- Attribute_Not_Defined
   -- Ownership_Acquisition_Pending
   -- Federate_Not_Execution_Member
   -- Save_In_Progress
   -- Restore_In_Progress
   -- RTI_Internal_Error
```

```
 -- 5.4
 procedure Publish_Interaction_Class (
  Ambassador : in RTI_Ambassador;
  Interaction_Class : in Interaction_Class_Handle);

  -- exceptions:
  -- Interaction_Class_Not_Defined
  -- Federate_Not_Execution_Member
  -- Save_In_Progress
  -- Restore_In_Progress
  -- RTI_Internal_Error

 -- 5.5
 procedure Unpublish_Interaction_Class (
  Ambassador : in RTI_Ambassador;
  Interaction_Class : in Interaction_Class_Handle);

  -- exceptions:
  -- Interaction_Class_Not_Defined
  -- Federate_Not_Execution_Member
  -- Save_In_Progress
  -- Restore_In_Progress
  -- RTI_Internal_Error

 -- 5.6
 procedure Subscribe_Object_Class_Attributes (
  Ambassador : in RTI_Ambassador;
  Object_Class : in Object_Class_Handle;
  Attributes : in Attribute_Handle_Set;
  Active : in Boolean := True);

  -- exceptions:
  -- Object_Class_Not_Defined
  -- Attribute_Not_Defined
  -- Federate_Not_Execution_Member
  -- Save_In_Progress
  -- Restore_In_Progress
  -- RTI_Internal_Error

 -- 5.7
 procedure Unsubscribe_Object_Class (
  Ambassador : in RTI_Ambassador;
  Object_Class : in Object_Class_Handle);

  -- exceptions:
  -- Object_Class_Not_Defined
  -- Federate_Not_Execution_Member
  -- Save_In_Progress
  -- Restore_In_Progress
  -- RTI_Internal_Error

 procedure Unsubscribe_Object_Class_Attributes (
  Ambassador : in RTI_Ambassador;
  Object_Class : in Object_Class_Handle;
  Attributes : in Attribute_Handle_Set);

  -- exceptions:
  -- Object_Class_Not_Defined
  -- Attribute_Not_Defined
  -- Federate_Not_Execution_Member
  -- Save_In_Progress
  -- Restore_In_Progress
  -- RTI_Internal_Error

 -- 5.8
 procedure Subscribe_Interaction_Class (
  Ambassador : in RTI_Ambassador;
  Interaction_Class : in Interaction_Class_Handle;
  Active : in Boolean := True);

  -- exceptions:
  -- Interaction_Class_Not_Defined
```



```
   -- Federate_Service_Invocations_Are_Being_Reported_Via_MOM
   -- Federate_Not_Execution_Member
   -- Save_In_Progress
   -- Restore_In_Progress
   -- RTI_Internal_Error

  -- 5.9
  procedure Unsubscribe_Interaction_Class (
   Ambassador : in RTI_Ambassador;
   Interaction_Class : in Interaction_Class_Handle);

   -- exceptions:
   -- Interaction_Class_Not_Defined
   -- Federate_Not_Execution_Member
   -- Save_In_Progress
   -- Restore_In_Progress
   -- RTI_Internal_Error

  -- 5.10 +
  procedure Start_Registration_For_Object_Class (
   Ambassador : in Federate_Ambassador;
   Object_Class : in Object_Class_Handle) is abstract;

   -- exceptions:
   -- Object_Class_Not_Published
   -- Federate_Internal_Error

  -- 5.11 +
  procedure Stop_Registration_For_Object_Class (
   Ambassador : in Federate_Ambassador;
   Object_Class : in Object_Class_Handle) is abstract;

   -- exceptions:
   -- Object_Class_Not_Published
   -- Federate_Internal_Error

  -- 5.12 +
  procedure Turn_Interactions_On (
   Ambassador : in Federate_Ambassador;
   Interaction_Class : in Interaction_Class_Handle) is abstract;

   -- exceptions:
   -- Interaction_Class_Not_Published
   -- Federate_Internal_Error

  -- 5.13 +
  procedure Turn_Interactions_Off (
   Ambassador : in Federate_Ambassador;
   Interaction_Class : in Interaction_Class_Handle) is abstract;

   -- exceptions:
   -- Interaction_Class_Not_Published
   -- Federate_Internal_Error

--
-- Object Management services
--

  -- 6.2
  procedure Reserve_Object_Instance_Name (
   Ambassador : in RTI_Ambassador;
   Object_Instance_Name : in Wide_String);

   -- exceptions:
   -- Illegal_Name
   -- Federate_Not_Execution_Member
   -- Save_In_Progress
   -- Restore_In_Progress
   -- RTI_Internal_Error

  -- 6.3 +
  procedure Object_Instance_Name_Reservation_Succeeded (
```

```
  Ambassador : in Federate_Ambassador;
  Object_Instance_Name : in Wide_String) is abstract;

  -- exceptions:
  -- Unknown_Name
  -- Federate_Internal_Error

 procedure Object_Instance_Name_Reservation_Failed (
  Ambassador : in Federate_Ambassador;
  Object_Instance_Name : in Wide_String) is abstract;

  -- exceptions:
  -- Unknown_Name
  -- Federate_Internal_Error

 -- 6.4
 procedure Register_Object_Instance (
  Ambassador : in RTI_Ambassador;
  Object_Class : in Object_Class_Handle;
  Object_Instance : out Object_Instance_Handle);

  -- exceptions:
  -- Object_Class_Not_Defined
  -- Object_Class_Not_Published
  -- Federate_Not_Execution_Member
  -- Save_In_Progress
  -- Restore_In_Progress
  -- RTI_Internal_Error

 procedure Register_Object_Instance (
  Ambassador : in RTI_Ambassador;
  Object_Class : in Object_Class_Handle;
  Object_Instance_Name : in Wide_String;
  Object_Instance : out Object_Instance_Handle);

  -- exceptions:
  -- Object_Class_Not_Defined
  -- Object_Class_Not_Published
  -- Object_Instance_Name_Not_Reserved
  -- Object_Instance_Name_In_Use
  -- Federate_Not_Execution_Member
  -- Save_In_Progress
  -- Restore_In_Progress
  -- RTI_Internal_Error

 -- 6.5 +
 procedure Discover_Object_Instance (
  Ambassador : in Federate_Ambassador;
  Object_Instance : in Object_Instance_Handle;
  Object_Class : in Object_Class_Handle;
  Object_Instance_Name : in Wide_String) is abstract;

  -- exceptions:
  -- Could_Not_Discover
  -- Object_Class_Not_Recognized
  -- Federate_Internal_Error

 -- 6.6
 procedure Update_Attribute_Values (
  Ambassador : in RTI_Ambassador;
  Object_Instance : in Object_Instance_Handle;
  Attribute_Values : in Attribute_Handle_Value_Map;
  Tag : in User_Supplied_Tag);

  -- exceptions:
  -- Object_Instance_Not_Known
  -- Attribute_Not_Defined
  -- Attribute_Not_Owned
  -- Federate_Not_Execution_Member
  -- Save_In_Progress
  -- Restore_In_Progress
  -- RTI_Internal_Error
```



```
procedure Update_Attribute_Values (
 Ambassador : in RTI_Ambassador;
 Object_Instance : in Object_Instance_Handle;
 Attribute_Values : in Attribute_Handle_Value_Map;
 Tag : in User_Supplied_Tag;
 Time : in Logical_Time'Class;
 Message_Retraction : out Message_Retraction_Handle;
 Message_Retraction_Is_Valid : out Boolean);

 -- exceptions:
 -- Object_Instance_Not_Known
 -- Attribute_Not_Defined
 -- Attribute_Not_Owned
 -- Invalid_Logical_Time
 -- Federate_Not_Execution_Member
 -- Save_In_Progress
 -- Restore_In_Progress
 -- RTI_Internal_Error

-- 6.7 +
procedure Reflect_Attribute_Values (
 Ambassador : in Federate_Ambassador;
 Object_Instance : in Object_Instance_Handle;
 Attribute_Values : in Attribute_Handle_Value_Map;
 Tag : in User_Supplied_Tag;
 Sent_Order : in Order_Type;
 Transportation : in Transportation_Type) is abstract;

 -- exceptions:
 -- Object_Instance_Not_Known
 -- Attribute_Not_Recognized
 -- Attribute_Not_Subscribed
 -- Federate_Internal_Error

procedure Reflect_Attribute_Values (
 Ambassador : in Federate_Ambassador;
 Object_Instance : in Object_Instance_Handle;
 Attribute_Values : in Attribute_Handle_Value_Map;
 Tag : in User_Supplied_Tag;
 Sent_Order : in Order_Type;
 Transportation : in Transportation_Type;
 Sent_Regions : in Region_Handle_Set) is abstract;

 -- exceptions:
 -- Object_Instance_Not_Known
 -- Attribute_Not_Recognized
 -- Attribute_Not_Subscribed
 -- Federate_Internal_Error

procedure Reflect_Attribute_Values (
 Ambassador : in Federate_Ambassador;
 Object_Instance : in Object_Instance_Handle;
 Attribute_Values : in Attribute_Handle_Value_Map;
 Tag : in User_Supplied_Tag;
 Sent_Order : in Order_Type;
 Transportation : in Transportation_Type;
 Time : in Logical_Time'Class;
 Received_Order : in Order_Type) is abstract;

 -- exceptions:
 -- Object_Instance_Not_Known
 -- Attribute_Not_Recognized
 -- Attribute_Not_Subscribed
 -- Federate_Internal_Error

procedure Reflect_Attribute_Values (
 Ambassador : in Federate_Ambassador;
 Object_Instance : in Object_Instance_Handle;
 Attribute_Values : in Attribute_Handle_Value_Map;
 Tag : in User_Supplied_Tag;
 Sent_Order : in Order_Type;
 Transportation : in Transportation_Type;
```

```
  Time : in Logical_Time'Class;
  Received_Order : in Order_Type;
  Sent_Regions : in Region_Handle_Set) is abstract;

  -- exceptions:
  -- Object_Instance_Not_Known
  -- Attribute_Not_Recognized
  -- Attribute_Not_Subscribed
  -- Federate_Internal_Error

 procedure Reflect_Attribute_Values (
  Ambassador : in Federate_Ambassador;
  Object_Instance : in Object_Instance_Handle;
  Attribute_Values : in Attribute_Handle_Value_Map;
  Tag : in User_Supplied_Tag;
  Sent_Order : in Order_Type;
  Transportation : in Transportation_Type;
  Time : in Logical_Time'Class;
  Received_Order : in Order_Type;
  Message_Retraction : in Message_Retraction_Handle) is abstract;

  -- exceptions:
  -- Object_Instance_Not_Known
  -- Attribute_Not_Recognized
  -- Attribute_Not_Subscribed
  -- Invalid_Logical_Time
  -- Federate_Internal_Error

 procedure Reflect_Attribute_Values (
  Ambassador : in Federate_Ambassador;
  Object_Instance : in Object_Instance_Handle;
  Attribute_Values : in Attribute_Handle_Value_Map;
  Tag : in User_Supplied_Tag;
  Sent_Order : in Order_Type;
  Transportation : in Transportation_Type;
  Time : in Logical_Time'Class;
  Received_Order : in Order_Type;
  Message_Retraction : in Message_Retraction_Handle;
  Sent_Regions : in Region_Handle_Set) is abstract;

  -- exceptions:
  -- Object_Instance_Not_Known
  -- Attribute_Not_Recognized
  -- Attribute_Not_Subscribed
  -- Invalid_Logical_Time
  -- Federate_Internal_Error

 -- 6.8
 procedure Send_Interaction (
  Ambassador : in RTI_Ambassador;
  Interaction_Class : in Interaction_Class_Handle;
  Parameter_Values : in Parameter_Handle_Value_Map;
  Tag : in User_Supplied_Tag);

  -- exceptions:
  -- Interaction_Class_Not_Published
  -- Interaction_Class_Not_Defined
  -- Interaction_Parameter_Not_Defined
  -- Federate_Not_Execution_Member
  -- Save_In_Progress
  -- Restore_In_Progress
  -- RTI_Internal_Error

 procedure Send_Interaction (
  Ambassador : in RTI_Ambassador;
  Interaction_Class : in Interaction_Class_Handle;
  Parameter_Values : in Parameter_Handle_Value_Map;
  Tag : in User_Supplied_Tag;
  Time : in Logical_Time'Class;
  Message_Retraction : out Message_Retraction_Handle;
  Message_Retraction_Is_Valid : out Boolean);
```



```
  -- exceptions:
  -- Interaction_Class_Not_Published
  -- Interaction_Class_Not_Defined
  -- Interaction_Parameter_Not_Defined
  -- Invalid_Logical_Time
  -- Federate_Not_Execution_Member
  -- Save_In_Progress
  -- Restore_In_Progress
  -- RTI_Internal_Error

-- 6.9 +
procedure Receive_Interaction (
 Ambassador : in Federate_Ambassador;
 Interaction_Class : in Interaction_Class_Handle;
 Parameter_Values : in Parameter_Handle_Value_Map;
 Tag : in User_Supplied_Tag;
 Sent_Order : in Order_Type;
 Transportation : in Transportation_Type) is abstract;

-- exceptions:
-- Interaction_Class_Not_Recognized
-- Interaction_Parameter_Not_Recognized
-- Interaction_Class_Not_Subscribed
-- Federate_Internal_Error

procedure Receive_Interaction (
 Ambassador : in Federate_Ambassador;
 Interaction_Class : in Interaction_Class_Handle;
 Parameter_Values : in Parameter_Handle_Value_Map;
 Tag : in User_Supplied_Tag;
 Sent_Order : in Order_Type;
 Transportation : in Transportation_Type;
 Sent_Regions : in Region_Handle_Set) is abstract;

 -- exceptions:
 -- Interaction_Class_Not_Recognized
 -- Interaction_Parameter_Not_Recognized
 -- Interaction_Class_Not_Subscribed
 -- Federate_Internal_Error

procedure Receive_Interaction (
 Ambassador : in Federate_Ambassador;
 Interaction_Class : in Interaction_Class_Handle;
 Parameter_Values : in Parameter_Handle_Value_Map;
 Tag : in User_Supplied_Tag;
 Sent_Order : in Order_Type;
 Transportation : in Transportation_Type;
 Time : in Logical_Time'Class;
 Received_Order : in Order_Type) is abstract;

 -- exceptions:
 -- Interaction_Class_Not_Recognized
 -- Interaction_Parameter_Not_Recognized
 -- Interaction_Class_Not_Subscribed
 -- Federate_Internal_Error

procedure Receive_Interaction (
 Ambassador : in Federate_Ambassador;
 Interaction_Class : in Interaction_Class_Handle;
 Parameter_Values : in Parameter_Handle_Value_Map;
 Tag : in User_Supplied_Tag;
 Sent_Order : in Order_Type;
 Transportation : in Transportation_Type;
 Time : in Logical_Time'Class;
 Received_Order : in Order_Type;
 Sent_Regions : in Region_Handle_Set) is abstract;

 -- exceptions:
 -- Interaction_Class_Not_Recognized
 -- Interaction_Parameter_Not_Recognized
 -- Interaction_Class_Not_Subscribed
 -- Federate_Internal_Error
```

```
 procedure Receive_Interaction (
  Ambassador : in Federate_Ambassador;
  Interaction_Class : in Interaction_Class_Handle;
  Parameter_Values : in Parameter_Handle_Value_Map;
  Tag : in User_Supplied_Tag;
  Sent_Order : in Order_Type;
  Transportation : in Transportation_Type;
  Time : in Logical_Time'Class;
  Received_Order : in Order_Type;
  Message_Retraction : in Message_Retraction_Handle) is abstract;

  -- exceptions:
  -- Interaction_Class_Not_Recognized
  -- Interaction_Parameter_Not_Recognized
  -- Interaction_Class_Not_Subscribed
  -- Invalid_Logical_Time
  -- Federate_Internal_Error

 procedure Receive_Interaction (
  Ambassador : in Federate_Ambassador;
  Interaction_Class : in Interaction_Class_Handle;
  Parameter_Values : in Parameter_Handle_Value_Map;
  Tag : in User_Supplied_Tag;
  Sent_Order : in Order_Type;
  Transportation : in Transportation_Type;
  Time : in Logical_Time'Class;
  Received_Order : in Order_Type;
  Message_Retraction : in Message_Retraction_Handle;
  Sent_Regions : in Region_Handle_Set) is abstract;

  -- exceptions:
  -- Interaction_Class_Not_Recognized
  -- Interaction_Parameter_Not_Recognized
  -- Interaction_Class_Not_Subscribed
  -- Invalid_Logical_Time
  -- Federate_Internal_Error

 -- 6.10
 procedure Delete_Object_Instance (
  Ambassador : in RTI_Ambassador;
  Object_Instance : in Object_Instance_Handle;
  Tag : in User_Supplied_Tag);

  -- exceptions:
  -- Delete_Privilege_Not_Held
  -- Object_Instance_Not_Known
  -- Federate_Not_Execution_Member
  -- Save_In_Progress
  -- Restore_In_Progress
  -- RTI_Internal_Error

 procedure Delete_Object_Instance (
  Ambassador : in RTI_Ambassador;
  Object_Instance : in Object_Instance_Handle;
  Tag : in User_Supplied_Tag;
  Time : in Logical_Time'Class;
  Message_Retraction : out Message_Retraction_Handle;
  Message_Retraction_Is_Valid : out Boolean);

  -- exceptions:
  -- Delete_Privilege_Not_Held
  -- Object_Instance_Not_Known
  -- Invalid_Logical_Time
  -- Federate_Not_Execution_Member
  -- Save_In_Progress
  -- Restore_In_Progress
  -- RTI_Internal_Error

 -- 6.11 +
 procedure Remove_Object_Instance (
  Ambassador : in Federate_Ambassador;
  Object_Instance : in Object_Instance_Handle;
```



```
  Tag : in User_Supplied_Tag;
  Sent_Order : in Order_Type) is abstract;

  -- exceptions:
  -- Object_Instance_Not_Known
  -- Federate_Internal_Error

 procedure Remove_Object_Instance (
  Ambassador : in Federate_Ambassador;
  Object_Instance : in Object_Instance_Handle;
  Tag : in User_Supplied_Tag;
  Sent_Order : in Order_Type;
  Time : in Logical_Time'Class;
  Received_Order : in Order_Type) is abstract;

  -- exceptions:
  -- Object_Instance_Not_Known
  -- Federate_Internal_Error
 procedure Remove_Object_Instance (
  Ambassador : in Federate_Ambassador;
  Object_Instance : in Object_Instance_Handle;
  Tag : in User_Supplied_Tag;
  Sent_Order : in Order_Type;
  Time : in Logical_Time'Class;
  Received_Order : in Order_Type;
  Message_Retraction : in Message_Retraction_Handle) is abstract;

  -- exceptions:
  -- Object_Instance_Not_Known
  -- Invalid_Logical_Time
  -- Federate_Internal_Error

 -- 6.12
 procedure Local_Delete_Object_Instance (
  Ambassador : in RTI_Ambassador;
  Object_Instance : in Object_Instance_Handle);

  -- exceptions:
  -- Object_Instance_Not_Known
  -- Federate_Owns_Attributes
  -- Ownership_Acquisition_Pending
  -- Federate_Not_Execution_Member
  -- Save_In_Progress
  -- Restore_In_Progress
  -- RTI_Internal_Error

 -- 6.13
 procedure Change_Attribute_Transportation_Type (
  Ambassador : in RTI_Ambassador;
  Object_Instance : in Object_Instance_Handle;
  Attributes : in Attribute_Handle_Set;
  Transportation : in Transportation_Type);

  -- exceptions:
  -- Object_Instance_Not_Known
  -- Attribute_Not_Defined
  -- Attribute_Not_Owned
  -- Federate_Not_Execution_Member
  -- Save_In_Progress
  -- Restore_In_Progress
  -- RTI_Internal_Error

 -- 6.14
 procedure Change_Interaction_Transportation_Type (
  Ambassador : in RTI_Ambassador;
  Interaction_Class : in Interaction_Class_Handle;
  Transportation : in Transportation_Type);

  -- exceptions:
  -- Interaction_Class_Not_Defined
  -- Interaction_Class_Not_Published
  -- Federate_Not_Execution_Member
```

```
   -- Save_In_Progress
   -- Restore_In_Progress
   -- RTI_Internal_Error

  -- 6.15 +
  procedure Attributes_In_Scope (
   Ambassador : in Federate_Ambassador;
   Object_Instance : in Object_Instance_Handle;
   Attributes : in Attribute_Handle_Set) is abstract;

   -- exceptions:
   -- Object_Instance_Not_Known
   -- Attribute_Not_Recognized
   -- Attribute_Not_Subscribed
   -- Federate_Internal_Error

  -- 6.16 +
  procedure Attributes_Out_Of_Scope (
   Ambassador : in Federate_Ambassador;
   Object_Instance : in Object_Instance_Handle;
   Attributes : in Attribute_Handle_Set) is abstract;

   -- exceptions:
   -- Object_Instance_Not_Known
   -- Attribute_Not_Recognized
   -- Attribute_Not_Subscribed
   -- Federate_Internal_Error

  -- 6.17
  procedure Request_Attribute_Value_Update (
   Ambassador : in RTI_Ambassador;
   Object_Instance : in Object_Instance_Handle;
   Attributes : in Attribute_Handle_Set;
   Tag : in User_Supplied_Tag);

   -- exceptions:
   -- Object_Instance_Not_Known
   -- Attribute_Not_Defined
   -- Federate_Not_Execution_Member
   -- Save_In_Progress
   -- Restore_In_Progress
   -- RTI_Internal_Error

  procedure Request_Attribute_Value_Update (
   Ambassador : in RTI_Ambassador;
   Object_Class : in Object_Class_Handle;
   Attributes : in Attribute_Handle_Set;
   Tag : in User_Supplied_Tag);

   -- exceptions:
   -- Object_Class_Not_Defined
   -- Attribute_Not_Defined
   -- Federate_Not_Execution_Member
   -- Save_In_Progress
   -- Restore_In_Progress
   -- RTI_Internal_Error

  -- 6.18 +
  procedure Provide_Attribute_Value_Update (
   Ambassador : in Federate_Ambassador;
   Object_Instance : in Object_Instance_Handle;
   Attributes : in Attribute_Handle_Set;
   Tag : in User_Supplied_Tag) is abstract;

   -- exceptions:
   -- Object_Instance_Not_Known
   -- Attribute_Not_Recognized
   -- Attribute_Not_Owned
   -- Federate_Internal_Error

  -- 6.19 +
  procedure Turn_Updates_On_For_Object_Instance (
```



```
   Ambassador : in Federate_Ambassador;
   Object_Instance : in Object_Instance_Handle;
   Attributes : in Attribute_Handle_Set) is abstract;

   -- exceptions:
   -- Object_Instance_Not_Known
   -- Attribute_Not_Recognized
   -- Attribute_Not_Owned
   -- Federate_Internal_Error

 -- 6.20 +
 procedure Turn_Updates_Off_For_Object_Instance (
   Ambassador : in Federate_Ambassador;
   Object_Instance : in Object_Instance_Handle;
   Attributes : in Attribute_Handle_Set) is abstract;

   -- exceptions:
   -- Object_Instance_Not_Known
   -- Attribute_Not_Recognized
   -- Attribute_Not_Owned
   -- Federate_Internal_Error

--
-- Ownership Management services
--

 -- 7.2
 procedure Unconditional_Attribute_Ownership_Divestiture (
   Ambassador : in RTI_Ambassador;
   Object_Instance : in Object_Instance_Handle;
   Attributes : in Attribute_Handle_Set);

   -- exceptions:
   -- Object_Instance_Not_Known
   -- Attribute_Not_Defined
   -- Attribute_Not_Owned
   -- Federate_Not_Execution_Member
   -- Save_In_Progress
   -- Restore_In_Progress
   -- RTI_Internal_Error

 -- 7.3
 procedure Negotiated_Attribute_Ownership_Divestiture (
   Ambassador : in RTI_Ambassador;
   Object_Instance : in Object_Instance_Handle;
   Attributes : in Attribute_Handle_Set;
   Tag : in User_Supplied_Tag);

   -- exceptions:
   -- Object_Instance_Not_Known
   -- Attribute_Not_Defined
   -- Attribute_Not_Owned
   -- Attribute_Already_Being_Divested
   -- Federate_Not_Execution_Member
   -- Save_In_Progress
   -- Restore_In_Progress
   -- RTI_Internal_Error

 -- 7.4 +
 procedure Request_Attribute_Ownership_Assumption (
   Ambassador : in Federate_Ambassador;
   Object_Instance : in Object_Instance_Handle;
   Offered_Attributes : in Attribute_Handle_Set;
   Tag : in User_Supplied_Tag) is abstract;

   -- exceptions:
   -- Object_Instance_Not_Known
   -- Attribute_Not_Recognized
   -- Attribute_Already_Owned
   -- Attribute_Not_Published
   -- Federate_Internal_Error
```

```
 -- 7.5 +
 procedure Request_Divestiture_Confirmation (
  Ambassador : in Federate_Ambassador;
  Object_Instance : in Object_Instance_Handle;
  Offered_Attributes : in Attribute_Handle_Set) is abstract;

  -- exceptions:
  -- Object_Instance_Not_Known
  -- Attribute_Not_Recognized
  -- Attribute_Not_Owned
  -- Attribute_Divestiture_Was_Not_Requested
  -- Federate_Internal_Error

 -- 7.6
 procedure Confirm_Divestiture (
  Ambassador : in RTI_Ambassador;
  Object_Instance : in Object_Instance_Handle;
  Attributes : in Attribute_Handle_Set;
  Tag : in User_Supplied_Tag);

  -- exceptions:
  -- Object_Instance_Not_Known
  -- Attribute_Not_Defined
  -- Attribute_Not_Owned
  -- Attribute_Divestiture_Was_Not_Requested
  -- Federate_Not_Execution_Member
  -- Save_In_Progress
  -- Restore_In_Progress
  -- RTI_Internal_Error

 -- 7.7 +
 procedure Attribute_Ownership_Acquisition_Notification (
  Ambassador : in Federate_Ambassador;
  Object_Instance : in Object_Instance_Handle;
  Secured_Attributes : in Attribute_Handle_Set;
  Tag : in User_Supplied_Tag) is abstract;

  -- exceptions:
  -- Object_Instance_Not_Known
  -- Attribute_Not_Recognized
  -- Attribute_Acquisition_Was_Not_Requested
  -- Attribute_Already_Owned
  -- Attribute_Not_Published
  -- Federate_Internal_Error

 -- 7.8
 procedure Attribute_Ownership_Acquisition (
  Ambassador : in RTI_Ambassador;
  Object_Instance : in Object_Instance_Handle;
  Desired_Attributes : in Attribute_Handle_Set;
  Tag : in User_Supplied_Tag);

  -- exceptions:
  -- Object_Instance_Not_Known
  -- Object_Class_Not_Published
  -- Attribute_Not_Defined
  -- Attribute_Not_Published
  -- Federate_Owns_Attributes
  -- Federate_Not_Execution_Member
  -- Save_In_Progress
  -- Restore_In_Progress
  -- RTI_Internal_Error

 -- 7.9
 procedure Attribute_Ownership_Acquisition_If_Available (
  Ambassador : in RTI_Ambassador;
  Object_Instance : in Object_Instance_Handle;
  Desired_Attributes : in Attribute_Handle_Set);

  -- exceptions:
  -- Object_Instance_Not_Known
  -- Object_Class_Not_Published
```



```
  -- Attribute_Not_Defined
  -- Attribute_Not_Published
  -- Federate_Owns_Attributes
  -- Attribute_Already_Being_Acquired
  -- Federate_Not_Execution_Member
  -- Save_In_Progress
  -- Restore_In_Progress
  -- RTI_Internal_Error

 -- 7.10 +
 procedure Attribute_Ownership_Unavailable (
  Ambassador : in Federate_Ambassador;
  Object_Instance : in Object_Instance_Handle;
  Attributes : in Attribute_Handle_Set) is abstract;

  -- exceptions:
  -- Object_Instance_Not_Known
  -- Attribute_Not_Recognized
  -- Attribute_Already_Owned
  -- Attribute_Acquisition_Was_Not_Requested
  -- Federate_Internal_Error

 -- 7.11 +
 procedure Request_Attribute_Ownership_Release (
  Ambassador : in Federate_Ambassador;
  Object_Instance : in Object_Instance_Handle;
  Candidate_Attributes : in Attribute_Handle_Set;
  Tag : in User_Supplied_Tag) is abstract;

  -- exceptions:
  -- Object_Instance_Not_Known
  -- Attribute_Not_Recognized
  -- Attribute_Not_Owned
  -- Federate_Internal_Error

 -- 7.12
 function Attribute_Ownership_Divestiture_If_Wanted (
  Ambassador : in RTI_Ambassador;
  Object_Instance : in Object_Instance_Handle;
  Attributes : in Attribute_Handle_Set)
 return Attribute_Handle_Set; -- actually divested attributes

  -- exceptions:
  -- Object_Instance_Not_Known
  -- Attribute_Not_Defined
  -- Attribute_Not_Owned
  -- Federate_Not_Execution_Member
  -- Save_In_Progress
  -- Restore_In_Progress
  -- RTI_Internal_Error

 -- 7.13
 procedure Cancel_Negotiated_Attribute_Ownership_Divestiture (
  Ambassador : in RTI_Ambassador;
  Object_Instance : in Object_Instance_Handle;
  Attributes : in Attribute_Handle_Set);

  -- exceptions:
  -- Object_Instance_Not_Known
  -- Attribute_Not_Defined
  -- Attribute_Not_Owned
  -- Attribute_Divestiture_Was_Not_Requested
  -- Federate_Not_Execution_Member
  -- Save_In_Progress
  -- Restore_In_Progress
  -- RTI_Internal_Error

 -- 7.14
 procedure Cancel_Attribute_Ownership_Acquisition (
  Ambassador : in RTI_Ambassador;
  Object_Instance : in Object_Instance_Handle;
  Attributes : in Attribute_Handle_Set);
```

```
  -- exceptions:
  -- Object_Instance_Not_Known
  -- Attribute_Not_Defined
  -- Attribute_Already_Owned
  -- Attribute_Acquisition_Was_Not_Requested
  -- Federate_Not_Execution_Member
  -- Save_In_Progress
  -- Restore_In_Progress
  -- RTI_Internal_Error

 -- 7.15 +
 procedure Confirm_Attribute_Ownership_Acquisition_Cancellation (
  Ambassador : in Federate_Ambassador;
  Object_Instance : in Object_Instance_Handle;
  Attributes : in Attribute_Handle_Set) is abstract;

  -- exceptions:
  -- Object_Instance_Not_Known
  -- Attribute_Not_Recognized
  -- Attribute_Already_Owned
  -- Attribute_Acquisition_Was_Not_Canceled
  -- Federate_Internal_Error

 -- 7.16
 procedure Query_Attribute_Ownership (
  Ambassador : in RTI_Ambassador;
  Object_Instance : in Object_Instance_Handle;
  Attribute : in Attribute_Handle);

  -- exceptions:
  -- Object_Instance_Not_Known
  -- Attribute_Not_Defined
  -- Federate_Not_Execution_Member
  -- Save_In_Progress
  -- Restore_In_Progress
  -- RTI_Internal_Error

 -- 7.17 +
 procedure Inform_Attribute_Ownership (
  Ambassador : in Federate_Ambassador;
  Object_Instance : in Object_Instance_Handle;
  Attribute : in Attribute_Handle;
  Owner : in Federate_Handle) is abstract;

  -- exceptions:
  -- Object_Instance_Not_Known
  -- Attribute_Not_Recognized
  -- Federate_Internal_Error

 procedure Attribute_Is_Not_Owned (
  Ambassador : in Federate_Ambassador;
  Object_Instance : in Object_Instance_Handle;
  Attribute : in Attribute_Handle) is abstract;

  -- exceptions:
  -- Object_Instance_Not_Known
  -- Attribute_Not_Recognized
  -- Federate_Internal_Error

 procedure Attribute_Is_Owned_By_RTI (
  Ambassador : in Federate_Ambassador;
  Object_Instance : in Object_Instance_Handle;
  Attribute : in Attribute_Handle) is abstract;

  -- exceptions:
  -- Object_Instance_Not_Known
  -- Attribute_Not_Recognized
  -- Federate_Internal_Error

 -- 7.18
 function Is_Attribute_Owned_By_Federate (
  Ambassador : in RTI_Ambassador;
```



```
   Object_Instance : in Object_Instance_Handle;
   Attribute : in Attribute_Handle)
  return Boolean;

   -- exceptions:
   -- Object_Instance_Not_Known
   -- Attribute_Not_Defined
   -- Federate_Not_Execution_Member
   -- Save_In_Progress
   -- Restore_In_Progress
   -- RTI_Internal_Error

--
-- Time Management services
--

  -- 8.2
  procedure Enable_Time_Regulation (
   Ambassador : in RTI_Ambassador;
   Lookahead : in Logical_Time_Interval'Class);

   -- exceptions:
   -- Time_Regulation_Already_Enabled
   -- Invalid_Lookahead
   -- In_Time_Advancing_State
   -- Request_For_Time_Regulation_Pending
   -- Federate_Not_Execution_Member
   -- Save_In_Progress
   -- Restore_In_Progress
   -- RTI_Internal_Error

  -- 8.3 +
  procedure Time_Regulation_Enabled (
   Ambassador : in Federate_Ambassador;
   Federate_Time : in Logical_Time'Class) is abstract;

   -- exceptions:
   -- Invalid_Logical_Time
   -- No_Request_To_Enable_Time_Regulation_Was_Pending
   -- Federate_Internal_Error

  -- 8.4
  procedure Disable_Time_Regulation (
   Ambassador : in RTI_Ambassador);

   -- exceptions:
   -- Time_Regulation_Is_Not_Enabled
   -- Federate_Not_Execution_Member
   -- Save_In_Progress
   -- Restore_In_Progress
   -- RTI_Internal_Error

  -- 8.5
  procedure Enable_Time_Constrained (
   Ambassador : in RTI_Ambassador);

   -- exceptions:
   -- Time_Constrained_Already_Enabled
   -- In_Time_Advancing_State
   -- Request_For_Time_Constrained_Pending
   -- Federate_Not_Execution_Member
   -- Save_In_Progress
   -- Restore_In_Progress
   -- RTI_Internal_Error

  -- 8.6 +
  procedure Time_Constrained_Enabled (
   Ambassador : in Federate_Ambassador;
   Federate_Time : in Logical_Time'Class) is abstract;

   -- exceptions:
   -- Invalid_Logical_Time
```

```
  -- No_Request_To_Enable_Time_Constrained_Was_Pending
  -- Federate_Internal_Error

 -- 8.7
 procedure Disable_Time_Constrained (
  Ambassador : in RTI_Ambassador);

  -- exceptions:
  -- Time_Constrained_Is_Not_Enabled
  -- Federate_Not_Execution_Member
  -- Save_In_Progress
  -- Restore_In_Progress
  -- RTI_Internal_Error

 -- 8.8
 procedure Time_Advance_Request (
  Ambassador : in RTI_Ambassador;
  Time : in Logical_Time'Class);

  -- exceptions:
  -- Invalid_Logical_Time
  -- Logical_Time_Already_Passed
  -- In_Time_Advancing_State
  -- Request_For_Time_Regulation_Pending
  -- Request_For_Time_Constrained_Pending
  -- Federate_Not_Execution_Member
  -- Save_In_Progress
  -- Restore_In_Progress
  -- RTI_Internal_Error

 -- 8.9
 procedure Time_Advance_Request_Available (
  Ambassador : in RTI_Ambassador;
  Time : in Logical_Time'Class);

  -- exceptions:
  -- Invalid_Logical_Time
  -- Logical_Time_Already_Passed
  -- In_Time_Advancing_State
  -- Request_For_Time_Regulation_Pending
  -- Request_For_Time_Constrained_Pending
  -- Federate_Not_Execution_Member
  -- Save_In_Progress
  -- Restore_In_Progress
  -- RTI_Internal_Error

 -- 8.10
 procedure Next_Message_Request (
  Ambassador : in RTI_Ambassador;
  Time : in Logical_Time'Class);

  -- exceptions:
  -- Invalid_Logical_Time
  -- Logical_Time_Already_Passed
  -- In_Time_Advancing_State
  -- Request_For_Time_Regulation_Pending
  -- Request_For_Time_Constrained_Pending
  -- Federate_Not_Execution_Member
  -- Save_In_Progress
  -- Restore_In_Progress
  -- RTI_Internal_Error

 -- 8.11
 procedure Next_Message_Request_Available (
  Ambassador : in RTI_Ambassador;
  Time : in Logical_Time'Class);

  -- exceptions:
  -- Invalid_Logical_Time
  -- Logical_Time_Already_Passed
  -- In_Time_Advancing_State
  -- Request_For_Time_Regulation_Pending
```



```
  -- Request_For_Time_Constrained_Pending
  -- Federate_Not_Execution_Member
  -- Save_In_Progress
  -- Restore_In_Progress
  -- RTI_Internal_Error

-- 8.12
procedure Flush_Queue_Request (
 Ambassador : in RTI_Ambassador;
 Time : in Logical_Time'Class);

 -- exceptions:
 -- Invalid_Logical_Time
 -- Logical_Time_Already_Passed
 -- In_Time_Advancing_State
 -- Request_For_Time_Regulation_Pending
 -- Request_For_Time_Constrained_Pending
 -- Federate_Not_Execution_Member
 -- Save_In_Progress
 -- Restore_In_Progress
 -- RTI_Internal_Error

-- 8.13 +
procedure Time_Advance_Grant (
 Ambassador : in Federate_Ambassador;
 Time : in Logical_Time'Class) is abstract;

 -- exceptions:
 -- Invalid_Logical_Time
 -- Joined_Federate_Is_Not_In_Time_Advancing_State
 -- Federate_Internal_Error

-- 8.14
procedure Enable_Asynchronous_Delivery (
 Ambassador : in RTI_Ambassador);

 -- exceptions:
 -- Asynchronous_Delivery_Already_Enabled
 -- Federate_Not_Execution_Member
 -- Save_In_Progress
 -- Restore_In_Progress
 -- RTI_Internal_Error

-- 8.15
procedure Disable_Asynchronous_Delivery (
 Ambassador : in RTI_Ambassador);

 -- exceptions:
 -- Asynchronous_Delivery_Already_Disabled
 -- Federate_Not_Execution_Member
 -- Save_In_Progress
 -- Restore_In_Progress
 -- RTI_Internal_Error

-- 8.16
procedure Query_GALT (
 Ambassador : in RTI_Ambassador;
 Time : out Logical_Time'Class;
 Time_Is_Valid : out Boolean);

 -- exceptions:
 -- Federate_Not_Execution_Member
 -- Save_In_Progress
 -- Restore_In_Progress
 -- RTI_Internal_Error

-- 8.17
function Query_Logical_Time (
 Ambassador : in RTI_Ambassador)
return Logical_Time'Class;
```

```
  -- exceptions:
  -- Federate_Not_Execution_Member
  -- Save_In_Progress
  -- Restore_In_Progress
  -- RTI_Internal_Error

 -- 8.18
 procedure Query_LITS (
  Ambassador : in RTI_Ambassador;
  Time : out Logical_Time'Class;
  Time_Is_Valid : out Boolean);

  -- exceptions:
  -- Federate_Not_Execution_Member
  -- Save_In_Progress
  -- Restore_In_Progress
  -- RTI_Internal_Error

 -- 8.19
 procedure Modify_Lookahead (
  Ambassador : in RTI_Ambassador;
  Lookahead : in Logical_Time_Interval'Class);

  -- exceptions:
  -- Time_Regulation_Is_Not_Enabled
  -- Invalid_Lookahead
  -- In_Time_Advancing_State
  -- Federate_Not_Execution_Member
  -- Save_In_Progress
  -- Restore_In_Progress
  -- RTI_Internal_Error

 -- 8.20
 function Query_Lookahead (
  Ambassador : in RTI_Ambassador)
 return Logical_Time_Interval'Class;

  -- exceptions:
  -- Time_Regulation_Is_Not_Enabled
  -- Federate_Not_Execution_Member
  -- Save_In_Progress
  -- Restore_In_Progress
  -- RTI_Internal_Error

 -- 8.21
 procedure Retract (
  Ambassador : in RTI_Ambassador;
  Message_Retraction : in Message_Retraction_Handle);

  -- exceptions:
  -- Invalid_Message_Retraction_Handle
  -- Time_Regulation_Is_Not_Enabled
  -- Message_Can_No_Longer_Be_Retracted
  -- Federate_Not_Execution_Member
  -- Save_In_Progress
  -- Restore_In_Progress
  -- RTI_Internal_Error

 -- 8.22 +
 procedure Request_Retraction (
  Ambassador : in Federate_Ambassador;
  Message_Retraction : in Message_Retraction_Handle) is abstract;

  -- exceptions:
  -- Federate_Internal_Error

 -- 8.23
 procedure Change_Attribute_Order_Type (
  Ambassador : in RTI_Ambassador;
  Object_Instance : in Object_Instance_Handle;
  Attributes : in Attribute_Handle_Set;
  Order : in Order_Type);
```



```
   -- exceptions:
   -- Object_Instance_Not_Known
   -- Attribute_Not_Defined
   -- Attribute_Not_Owned
   -- Federate_Not_Execution_Member
   -- Save_In_Progress
   -- Restore_In_Progress
   -- RTI_Internal_Error

  -- 8.24
  procedure Change_Interaction_Order_Type (
   Ambassador : in RTI_Ambassador;
   Interaction_Class : in Interaction_Class_Handle;
   Order : in Order_Type);

   -- exceptions:
   -- Interaction_Class_Not_Defined
   -- Interaction_Class_Not_Published
   -- Federate_Not_Execution_Member
   -- Save_In_Progress
   -- Restore_In_Progress
   -- RTI_Internal_Error

--
-- Data Distribution Management services
--

  -- 9.2
  procedure Create_Region (
   Ambassador : in RTI_Ambassador;
   Dimensions : in Dimension_Handle_Set;
   Region : out Region_Handle);

   -- exceptions:
   -- Invalid_Dimension_Handle
   -- Federate_Not_Execution_Member
   -- Save_In_Progress
   -- Restore_In_Progress
   -- RTI_Internal_Error

  -- 9.3
  procedure Commit_Region_Modifications (
   Ambassador : in RTI_Ambassador;
   Regions : in Region_Handle_Set);

   -- exceptions:
   -- Invalid_Region
   -- Region_Not_Created_By_This_Federate
   -- Federate_Not_Execution_Member
   -- Save_In_Progress
   -- Restore_In_Progress
   -- RTI_Internal_Error

  -- 9.4
  procedure Delete_Region (
   Ambassador : in RTI_Ambassador;
   Region : in Region_Handle);

   -- exceptions:
   -- Invalid_Region
   -- Region_Not_Created_By_This_Federate
   -- Region_In_Use_For_Update_Or_Subscription
   -- Federate_Not_Execution_Member
   -- Save_In_Progress
   -- Restore_In_Progress
   -- RTI_Internal_Error

  -- 9.5
  procedure Register_Object_Instance_With_Regions (
   Ambassador : in RTI_Ambassador;
   Object_Class : in Object_Class_Handle;
   Attributes_And_Regions : in Attributes_Regions_Pairs;
```

```
  Object_Instance : out Object_Instance_Handle);

  -- exceptions:
  -- Object_Class_Not_Defined
  -- Object_Class_Not_Published
  -- Attribute_Not_Defined
  -- Attribute_Not_Published
  -- Invalid_Region
  -- Region_Not_Created_By_This_Federate
  -- Invalid_Region_Context
  -- Federate_Not_Execution_Member
  -- Save_In_Progress
  -- Restore_In_Progress
  -- RTI_Internal_Error

 procedure Register_Object_Instance_With_Regions (
  Ambassador : in RTI_Ambassador;
  Object_Class : in Object_Class_Handle;
  Attributes_And_Regions : in Attributes_Regions_Pairs;
  Object_Instance_Name : in Wide_String;
  Object_Instance : out Object_Instance_Handle);

  -- exceptions:
  -- Object_Class_Not_Defined
  -- Object_Class_Not_Published
  -- Attribute_Not_Defined
  -- Attribute_Not_Published
  -- Invalid_Region
  -- Region_Not_Created_By_This_Federate
  -- Invalid_Region_Context
  -- Object_Instance_Name_Not_Reserved
  -- Object_Instance_Name_In_Use
  -- Federate_Not_Execution_Member
  -- Save_In_Progress
  -- Restore_In_Progress
  -- RTI_Internal_Error

 -- 9.6
 procedure Associate_Regions_For_Updates (
  Ambassador : in RTI_Ambassador;
  Object_Instance : in Object_Instance_Handle;
  Attributes_And_Regions : in Attributes_Regions_Pairs);

  -- exceptions:
  -- Object_Instance_Not_Known
  -- Attribute_Not_Defined
  -- Invalid_Region
  -- Region_Not_Created_By_This_Federate
  -- Invalid_Region_Context
  -- Federate_Not_Execution_Member
  -- Save_In_Progress
  -- Restore_In_Progress
  -- RTI_Internal_Error

 -- 9.7
 procedure Unassociate_Regions_For_Updates (
  Ambassador : in RTI_Ambassador;
  Object_Instance : in Object_Instance_Handle;
  Attributes_And_Regions : in Attributes_Regions_Pairs);

  -- exceptions:
  -- Object_Instance_Not_Known
  -- Attribute_Not_Defined
  -- Invalid_Region
  -- Region_Not_Created_By_This_Federate
  -- Federate_Not_Execution_Member
  -- Save_In_Progress
  -- Restore_In_Progress
  -- RTI_Internal_Error

 -- 9.8
 procedure Subscribe_Object_Class_Attributes_With_Regions (
```



```
  Ambassador : in RTI_Ambassador;
  Object_Class : in Object_Class_Handle;
  Attributes_And_Regions : in Attributes_Regions_Pairs;
  Active : in Boolean := True);

  -- exceptions:
  -- Object_Class_Not_Defined
  -- Attribute_Not_Defined
  -- Invalid_Region
  -- Region_Not_Created_By_This_Federate
  -- Invalid_Region_Context
  -- Federate_Not_Execution_Member
  -- Save_In_Progress
  -- Restore_In_Progress
  -- RTI_Internal_Error

-- 9.9
procedure Unsubscribe_Object_Class_Attributes_With_Regions (
  Ambassador : in RTI_Ambassador;
  Object_Class : in Object_Class_Handle;
  Attributes_And_Regions : in Attributes_Regions_Pairs);

  -- exceptions:
  -- Object_Class_Not_Defined
  -- Attribute_Not_Defined
  -- Invalid_Region
  -- Region_Not_Created_By_This_Federate
  -- Federate_Not_Execution_Member
  -- Save_In_Progress
  -- Restore_In_Progress
  -- RTI_Internal_Error

-- 9.10
procedure Subscribe_Interaction_Class_With_Regions (
  Ambassador : in RTI_Ambassador;
  Interaction_Class : in Interaction_Class_Handle;
  Regions : in Region_Handle_Set;
  Active : in Boolean := True);

  -- exceptions:
  -- Interaction_Class_Not_Defined
  -- Invalid_Region
  -- Region_Not_Created_By_This_Federate
  -- Invalid_Region_Context
  -- Federate_Service_Invocations_Are_Being_Reported_Via_MOM
  -- Federate_Not_Execution_Member
  -- Save_In_Progress
  -- Restore_In_Progress
  -- RTI_Internal_Error

-- 9.11
procedure Unsubscribe_Interaction_Class_With_Regions (
  Ambassador : in RTI_Ambassador;
  Interaction_Class : in Interaction_Class_Handle;
  Regions : in Region_Handle_Set);

  -- exceptions:
  -- Interaction_Class_Not_Defined
  -- Invalid_Region
  -- Region_Not_Created_By_This_Federate
  -- Federate_Not_Execution_Member
  -- Save_In_Progress
  -- Restore_In_Progress
  -- RTI_Internal_Error

-- 9.12
procedure Send_Interaction_With_Regions (
  Ambassador : in RTI_Ambassador;
  Interaction_Class : in Interaction_Class_Handle;
  Parameter_Values : in Parameter_Handle_Value_Map;
  Regions : in Region_Handle_Set;
  Tag : in User_Supplied_Tag);
```

```
   -- exceptions:
   -- Interaction_Class_Not_Defined
   -- Interaction_Class_Not_Published
   -- Interaction_Parameter_Not_Defined
   -- Invalid_Region
   -- Region_Not_Created_By_This_Federate
   -- Invalid_Region_Context
   -- Federate_Not_Execution_Member
   -- Save_In_Progress
   -- Restore_In_Progress
   -- RTI_Internal_Error

  procedure Send_Interaction_With_Regions (
   Ambassador : in RTI_Ambassador;
   Interaction_Class : in Interaction_Class_Handle;
   Parameter_Values : in Parameter_Handle_Value_Map;
   Regions : in Region_Handle_Set;
   Tag : in User_Supplied_Tag;
   Time : in Logical_Time'Class;
   Message_Retraction : out Message_Retraction_Handle;
   Message_Retraction_Is_Valid : out Boolean);

   -- exceptions:
   -- Interaction_Class_Not_Defined
   -- Interaction_Class_Not_Published
   -- Interaction_Parameter_Not_Defined
   -- Invalid_Region
   -- Region_Not_Created_By_This_Federate
   -- Invalid_Region_Context
   -- Invalid_Logical_Time
   -- Federate_Not_Execution_Member
   -- Save_In_Progress
   -- Restore_In_Progress
   -- RTI_Internal_Error

  -- 9.13
  procedure Request_Attribute_Value_Update_With_Regions (
   Ambassador : in RTI_Ambassador;
   Object_Class : in Object_Class_Handle;
   Attributes_And_Regions : in Attributes_Regions_Pairs;
   Tag : in User_Supplied_Tag);

   -- exceptions:
   -- Object_Class_Not_Defined
   -- Attribute_Not_Defined
   -- Invalid_Region
   -- Region_Not_Created_By_This_Federate
   -- Invalid_Region_Context
   -- Federate_Not_Execution_Member
   -- Save_In_Progress
   -- Restore_In_Progress
   -- RTI_Internal_Error\

--
-- RTI Support services
--

  -- 10.2
  function Get_Object_Class_Handle (
   Ambassador : in RTI_Ambassador;
   Object_Class_Name : in Wide_String)
  return Object_Class_Handle;

   -- exceptions:
   -- Name_Not_Found
   -- Federate_Not_Execution_Member
   -- RTI_Internal_Error

  -- 10.3
  function Get_Object_Class_Name (
   Ambassador : in RTI_Ambassador;
   Object_Class : in Object_Class_Handle)
```



```
return Wide_String;

 -- exceptions:
 -- Invalid_Object_Class_Handle
 -- Federate_Not_Execution_Member
 -- RTI_Internal_Error

-- 10.4
function Get_Attribute_Handle (
 Ambassador : in RTI_Ambassador;
 Which_Class : in Object_Class_Handle;
 Attribute_Name : in Wide_String)
return Attribute_Handle;

 -- exceptions:
 -- Invalid_Object_Class_Handle
 -- Name_Not_Found
 -- Federate_Not_Execution_Member
 -- RTI_Internal_Error

-- 10.5
function Get_Attribute_Name (
 Ambassador : in RTI_Ambassador;
 Which_Class : in Object_Class_Handle;
 Attribute : in Attribute_Handle)
return Wide_String;

 -- exceptions:
 -- Invalid_Object_Class_Handle
 -- Invalid_Attribute_Handle
 -- Attribute_Not_Defined
 -- Federate_Not_Execution_Member
 -- RTI_Internal_Error

-- 10.6
function Get_Interaction_Class_Handle (
 Ambassador : in RTI_Ambassador;
 Interaction_Class_Name : in Wide_String)
return Interaction_Class_Handle;

 -- exceptions:
 -- Name_Not_Found
 -- Federate_Not_Execution_Member
 -- RTI_Internal_Error

-- 10.7
function Get_Interaction_Class_Name (
 Ambassador : in RTI_Ambassador;
 Interaction_Class : in Interaction_Class_Handle)
return Wide_String;

 -- exceptions:
 -- Invalid_Interaction_Class_Handle
 -- Federate_Not_Execution_Member
 -- RTI_Internal_Error

-- 10.8
function Get_Parameter_Handle (
 Ambassador : in RTI_Ambassador;
 Which_Class : in Interaction_Class_Handle;
 Parameter_Name : in Wide_String)
return Parameter_Handle;

 -- exceptions:
 -- Invalid_Interaction_Class_Handle
 -- Name_Not_Found
 -- Federate_Not_Execution_Member
 -- RTI_Internal_Error

-- 10.9
function Get_Parameter_Name (
 Ambassador : in RTI_Ambassador;
```

```
  Which_Class : in Interaction_Class_Handle;
  Parameter : in Parameter_Handle)
 return Wide_String;

  -- exceptions:
  -- Invalid_Interaction_Class_Handle
  -- Invalid_Parameter_Handle
  -- Interaction_Parameter_Not_Defined
  -- Federate_Not_Execution_Member
  -- RTI_Internal_Error

 -- 10.10
 function Get_Object_Instance_Handle (
  Ambassador : in RTI_Ambassador;
  Object_Instance_Name : in Wide_String)
 return Object_Instance_Handle;

  -- exceptions:
  -- Object_Instance_Not_Known
  -- Federate_Not_Execution_Member
  -- RTI_Internal_Error

 -- 10.11
 function Get_Object_Instance_Name (
  Ambassador : in RTI_Ambassador;
  Object_Instance : in Object_Instance_Handle)
 return Wide_String;

  -- exceptions:
  -- Object_Instance_Not_Known
  -- Federate_Not_Execution_Member
  -- RTI_Internal_Error

 -- 10.12
 function Get_Dimension_Handle (
  Ambassador : in RTI_Ambassador;
  Dimension_Name : in Wide_String)
 return Dimension_Handle;

  -- exceptions:
  -- Name_Not_Found
  -- Federate_Not_Execution_Member
  -- RTI_Internal_Error

 -- 10.13
 function Get_Dimension_Name (
  Ambassador : in RTI_Ambassador;
  Dimension : in Dimension_Handle)
 return Wide_String;

  -- exceptions:
  -- Invalid_Dimension_Handle
  -- Federate_Not_Execution_Member
  -- RTI_Internal_Error

 -- 10.14
 function Get_Dimension_Upper_Bound (
  Ambassador : in RTI_Ambassador;
  Dimension : in Dimension_Handle)
 return HLA_DDM_Coordinate;

  -- exceptions:
  -- Invalid_Dimension_Handle
  -- Federate_Not_Execution_Member
  -- RTI_Internal_Error

 -- 10.15
 function Get_Available_Dimensions_For_Class_Attribute (
  Ambassador : in RTI_Ambassador;
  Object_Class : in Object_Class_Handle;
  Attribute : in Attribute_Handle)
 return Dimension_Handle_Set;
```



```
 -- exceptions:
 -- Invalid_Object_Class_Handle
 -- Invalid_Attribute_Handle
 -- Attribute_Not_Defined
 -- Federate_Not_Execution_Member
 -- RTI_Internal_Error

-- 10.16
function Get_Known_Object_Class_Handle (
 Ambassador : in RTI_Ambassador;
 Object_Instance : in Object_Instance_Handle)
return Object_Class_Handle;

 -- exceptions:
 -- Object_Instance_Not_Known
 -- Federate_Not_Execution_Member
 -- RTI_Internal_Error

-- 10.17
function Get_Available_Dimensions_For_Interaction_Class (
 Ambassador : in RTI_Ambassador;
 Interaction_Class : in Interaction_Class_Handle)
return Dimension_Handle_Set;

 -- exceptions:
 -- Invalid_Interaction_Class_Handle
 -- Federate_Not_Execution_Member
 -- RTI_Internal_Error

-- 10.18
function Get_Transportation_Type (
 Ambassador : in RTI_Ambassador;
 Transportation_Name : in Wide_String)
return Transportation_Type;

 -- exceptions:
 -- Invalid_Transportation_Name
 -- Federate_Not_Execution_Member
 -- RTI_Internal_Error

-- 10.19
function Get_Transportation_Name (
 Ambassador : in RTI_Ambassador;
 Transportation : in Transportation_Type)
return Wide_String;

 -- exceptions:
 -- Invalid_Transportation_Type
 -- Federate_Not_Execution_Member
 -- RTI_Internal_Error

-- 10.20
function Get_Order_Type (
 Ambassador : in RTI_Ambassador;
 Order_Name : in Wide_String)
return Order_Type;

 -- exceptions:
 -- Invalid_Order_Name
 -- Federate_Not_Execution_Member
 -- RTI_Internal_Error

-- 10.21
function Get_Order_Name (
 Ambassador : in RTI_Ambassador;
 Order : in Order_Type)
return Wide_String;

 -- exceptions:
 -- Invalid_Order_Type
 -- Federate_Not_Execution_Member
 -- RTI_Internal_Error
```

```
 -- 10.22
 procedure Enable_Object_Class_Relevance_Advisory_Switch (
  Ambassador : in RTI_Ambassador);

  -- exceptions:
  -- Object_Class_Relevance_Advisory_Switch_Is_On
  -- Federate_Not_Execution_Member
  -- Save_In_Progress
  -- Restore_In_Progress
  -- RTI_Internal_Error

 -- 10.23
 procedure Disable_Object_Class_Relevance_Advisory_Switch (
  Ambassador : in RTI_Ambassador);

  -- exceptions:
  -- Object_Class_Relevance_Advisory_Switch_Is_Off
  -- Federate_Not_Execution_Member
  -- Save_In_Progress
  -- Restore_In_Progress
  -- RTI_Internal_Error

 -- 10.24
 procedure Enable_Attribute_Relevance_Advisory_Switch (
  Ambassador : in RTI_Ambassador);

  -- exceptions:
  -- Attribute_Relevance_Advisory_Switch_Is_On
  -- Federate_Not_Execution_Member
  -- Save_In_Progress
  -- Restore_In_Progress
  -- RTI_Internal_Error

 -- 10.25
 procedure Disable_Attribute_Relevance_Advisory_Switch (
  Ambassador : in RTI_Ambassador);

  -- exceptions:
  -- Attribute_Relevance_Advisory_Switch_Is_Off
  -- Federate_Not_Execution_Member
  -- Save_In_Progress
  -- Restore_In_Progress
  -- RTI_Internal_Error

 -- 10.26
 procedure Enable_Attribute_Scope_Advisory_Switch (
  Ambassador : in RTI_Ambassador);

  -- exceptions:
  -- Attribute_Scope_Advisory_Switch_Is_On
  -- Federate_Not_Execution_Member
  -- Save_In_Progress
  -- Restore_In_Progress
  -- RTI_Internal_Error

 -- 10.27
 procedure Disable_Attribute_Scope_Advisory_Switch (
  Ambassador : in RTI_Ambassador);

  -- exceptions:
  -- Attribute_Scope_Advisory_Switch_Is_Off
  -- Federate_Not_Execution_Member
  -- Save_In_Progress
  -- Restore_In_Progress
  -- RTI_Internal_Error

 -- 10.28
 procedure Enable_Interaction_Relevance_Advisory_Switch (
  Ambassador : in RTI_Ambassador);

  -- exceptions:
  -- Interaction_Relevance_Advisory_Switch_Is_On
```



```
  -- Federate_Not_Execution_Member
  -- Save_In_Progress
  -- Restore_In_Progress
  -- RTI_Internal_Error

-- 10.29
procedure Disable_Interaction_Relevance_Advisory_Switch (
 Ambassador : in RTI_Ambassador);

  -- exceptions:
  -- Interaction_Relevance_Advisory_Switch_Is_Off
  -- Federate_Not_Execution_Member
  -- Save_In_Progress
  -- Restore_In_Progress
  -- RTI_Internal_Error

-- 10.30
function Get_Dimension_Handle_Set (
 Ambassador : in RTI_Ambassador;
 Region : in Region_Handle)
return Dimension_Handle_Set;

  -- exceptions:
  -- Invalid_Region
  -- Federate_Not_Execution_Member
  -- Save_In_Progress
  -- Restore_In_Progress
  -- RTI_Internal_Error

-- 10.31
function Get_Range_Bounds (
 Ambassador : in RTI_Ambassador;
 Region : in Region_Handle;
 Dimension : in Dimension_Handle)
return Range_Bounds;

  -- exceptions:
  -- Invalid_Region
  -- Region_Does_Not_Contain_Specified_Dimension
  -- Federate_Not_Execution_Member
  -- Save_In_Progress
  -- Restore_In_Progress
  -- RTI_Internal_Error

-- 10.32
procedure Set_Range_Bounds (
 Ambassador : in RTI_Ambassador;
 Region : in Region_Handle;
 Dimension : in Dimension_Handle;
 Bounds : in Range_Bounds);

  -- exceptions:
  -- Invalid_Region
  -- Region_Not_Created_By_This_Federate
  -- Region_Does_Not_Contain_Specified_Dimension
  -- Invalid_Range_Bound
  -- Federate_Not_Execution_Member
  -- Save_In_Progress
  -- Restore_In_Progress
  -- RTI_Internal_Error

-- 10.33
function Normalize_Federate_Handle (
 Ambassador : in RTI_Ambassador;
 Federate : in Federate_Handle)
return HLA_DDM_Coordinate;

  -- exceptions:
  -- Invalid_Federate_Handle
  -- Federate_Not_Execution_Member
  -- RTI_Internal_Error
```

```
-- 10.34
type Service_Group is (
 Federation_Management,
 Declaration_Management,
 Object_Management,
 Ownership_Management,
 Time_Management,
 Data_Distribution_Management,
 Support_Services);

function Normalize_Service_Group (
 Ambassador : in RTI_Ambassador;
 Group : in Service_Group)
 return HLA_DDM_Coordinate;

 -- exceptions:
 -- Federate_Not_Execution_Member
 -- RTI_Internal_Error

-- 10.35
type RTI_Arguments is array (Positive range <>) of
 Ada.Strings.Wide_Unbounded.Unbounded_Wide_String;

function Initialize_RTI (
 Ambassador : in RTI_Ambassador;
 Arguments : in RTI_Arguments)
return RTI_Arguments; -- arguments unused by the RTI

 -- exceptions:
 -- Initialize_Previously_Invoked
 -- Bad_Initialization_Parameter
 -- RTI_Internal_Error

-- 10.36
procedure Finalize_RTI (
 Ambassador : in RTI_Ambassador);

 -- exceptions:
 -- Initialize_Never_Invoked
 -- Some_Federate_Is_Joined_To_An_Execution
 -- RTI_Internal_Error

-- 10.37
procedure Evoke_Callback (
 Ambassador : in RTI_Ambassador;
 Minimum_Wait_Time : in Duration;
 Pending_Callback_Exists : out Boolean);

 -- exceptions:
 -- Federate_Not_Execution_Member
 -- RTI_Internal_Error

-- 10.38
procedure Evoke_Multiple_Callbacks (
 Ambassador : in RTI_Ambassador;
 Minimum_Wait_Time : in Duration;
 Maximum_Wait_Time : in Duration;
 Pending_Callback_Exists : out Boolean);

 -- exceptions:
 -- Federate_Not_Execution_Member
 -- RTI_Internal_Error

-- 10.39
procedure Enable_Callbacks (
 Ambassador : in RTI_Ambassador);

 -- exceptions:
 -- Federate_Not_Execution_Member
 -- Save_In_Progress
 -- Restore_In_Progress
 -- RTI_Internal_Error
```



```
 -- 10.40
 procedure Disable_Callbacks (
  Ambassador : in RTI_Ambassador);

  -- exceptions:
  -- Federate_Not_Execution_Member
  -- Save_In_Progress
  -- Restore_In_Progress
  -- RTI_Internal_Error

private
 ... -- not specified by the Standard

end <Some_Supplier>_RTI;
```

A.2 Examples

This subclause is non-normative. It provides examples of integer 64 based implementations of the HLA time ADTs.

```
-- File: default_logical_time.ads

-- ============================================================================
-- IEEE 1516.1 High Level Architecture Federate Interface Specification
-- Ada 95 API
-- ============================================================================


with <some-supplier>_RTI;

package Default_Logical_Time is


 -- This is the default concrete implementation for the Logical Time Interval
 -- and the Logical Time of the federate.
 -- Default Logical Time Interval implementation is a 64 bits 2's complement
 -- integer type.
 -- Zero is equal to 0.
 -- Epsilon is equal to 1.
 -- Encoded Default Logical Time Interval shall insure interoperability
 -- between targeted platforms.
 -- Default Logical Time implementation is a 64 bits 2's complement integer
 -- type.
 -- Initial is equal to 0.
 -- Final is equal to 2**63 -1.
 -- Encoded Default Logical Time shall insure interoperability between
 -- targeted platforms.

--
-- Integer_64_Time_Interval implementation
--

 type Integer_64_Time_Interval is new RTI.Logical_Time_Interval with private;
 -- Concrete implementation of abstract services declared in RTI package:
 procedure Set_To_Zero (
  Time_Interval : out Integer_64_Time_Interval);
 procedure Set_To_Epsilon (
  Time_Interval : out Integer_64_Time_Interval);
 procedure Set_To (
  Time_Interval : out Integer_64_Time_Interval;
  Value : in Integer_64_Time_Interval);
 function "-" (
  L, R : in Integer_64_Time_Interval)
 return Integer_64_Time_Interval;

  -- exception:
  -- Constraint_Error
```

```
 function Is_Zero (
  Time_Interval : in Integer_64_Time_Interval)
 return Boolean;

 function Is_Epsilon (
  Time_Interval : in Integer_64_Time_Interval)
 return Boolean;

 function "<=" (
  L, R : in Integer_64_Time_Interval)
 return Boolean;

 function "<" (
  L, R : in Integer_64_Time_Interval)
 return Boolean;

 function ">=" (
  L, R : in Integer_64_Time_Interval)
 return Boolean;

 function ">" (
  L, R : in Integer_64_Time_Interval)
 return Boolean;

 function "=" (
  L, R : in Integer_64_Time_Interval)
 return Boolean;

 function Image (
  Time_Interval : in Integer_64_Time_Interval)
 return Wide_String;

 function Encode (
  Time_Interval : in Integer_64_Time_Interval)
 return Rti.Encoded_Value;

 function Decode (
  Value : in Rti.Encoded_Value)
 return Integer_64_Time_Interval;

  -- exception:
  -- Data_Error

--
-- Integer_64_Time implementation
--

 type Integer_64_Time is new RTI.Logical_Time with private;

 -- Concrete implementation of abstract services declared in RTI package:

 procedure Set_To_Initial (
  Time : out Integer_64_Time);

 procedure Set_To_Final (
  Time : out Integer_64_Time);

 procedure Set_To (
  Time : out Integer_64_Time;
  Value : in Integer_64_Time);

 function "+" (
  L : in Integer_64_Time;
  R : in RTI.Logical_Time_Interval'Class)
 return Integer_64_Time;

  -- exception:
  -- Constraint_Error

 function "-" (
  L : in Integer_64_Time;
  R : in RTI.Logical_Time_Interval'Class)
```



```
 return Integer_64_Time;

  -- exception:
  -- Constraint_Error

 function "-" (
  L, R : in Integer_64_Time)
 return RTI.Logical_Time_Interval'Class;

 function Is_Initial (
  Time : in Integer_64_Time)
 return Boolean;

 function Is_Final (
  Time : in Integer_64_Time)
 return Boolean;

 function "<=" (
  L, R : in Integer_64_Time)
 return Boolean;

 function "<" (
  L, R : in Integer_64_Time)
 return Boolean;

 function ">=" (
  L, R : in Integer_64_Time)
 return Boolean;

 function ">" (
  L, R : in Integer_64_Time)
 return Boolean;

 function "=" (
  L, R : in Integer_64_Time)
 return Boolean;

 function Image (
  Time : in Integer_64_Time)
 return Wide_String;

 function Encode (
  Time : in Integer_64_Time)
 return Rti.Encoded_Value;

 function Decode (
  Value : in Rti.Encoded_Value)
 return Integer_64_Time;

  -- exception:
  -- Data_Error

private

 type Underlying_Representation is range -(2**63)..2**63 -1;

 -- Integer_64_Time_Interval implementation:

 type Time_Interval_Implementation is new Underlying_Representation;

 Zero : constant Time_Interval_Implementation := 0;
 Epsilon : constant Time_Interval_Implementation := 1;

 type Integer_64_Time_Interval is new RTI.Logical_Time_Interval with record
  Value : Time_Interval_Implementation := Zero;
 end record;

 -- Integer_64_Time implementation:

 type Time_Implementation is new Underlying_Representation
```

```
   range 0..Underlying_Representation'Last;

  Initial : constant Time_Implementation := 0;
  Final : constant Time_Implementation := Time_Implementation'Last;

  type Integer_64_Time is new RTI.Logical_Time with record
   Value : Time_Implementation := Initial;
   end record;

end Default_Logical_Time;
```

Annex B

(normative)

Java application programmer's interface

B.1 API

The following files shall comprise the Java HLA API.

```
//File: RTIambassador.java

package hla.rti;

/**
Memory Management Conventions for Parameters

All Java parameters, including object references, are passed by value.
Therefore there is no need to specify further conventions for primitive types.

Unless otherwise noted, reference parameters adhere to the following convention:
The referenced object is created (or acquired) by the caller. The callee must
copy during the call anything it wishes to save beyond the completion of the
call.

Unless otherwise noted, a reference returned from a method represents a new
object created by the callee. The caller is free to modify the object whose
reference is returned.


*/

/**
 * The RTI presents this interface to the federate.
 * RTI implementer must implement this.
*/

public interface RTIambassador {

////////////////////////////////////
// Federation Management Services //
////////////////////////////////////

//4.2
public void createFederationExecution (
 String          federationExecutionName,
 java.net.URL fdd)
throws
 FederationExecutionAlreadyExists,
 CouldNotOpenFDD,
 ErrorReadingFDD,
 RTIinternalError;

//4.3
public void destroyFederationExecution (
 String federationExecutionName)
throws
 FederatesCurrentlyJoined,
 FederationExecutionDoesNotExist,
 RTIinternalError;
```

```
//4.4
public FederateHandle
joinFederationExecution(
 String              federateType,
 String              federationExecutionName,
 FederateAmbassador  federateReference,
 MobileFederateServices serviceReferences)
throws
 FederateAlreadyExecutionMember,
 FederationExecutionDoesNotExist,
 SaveInProgress,
 RestoreInProgress,
 RTIinternalError;

//4.5
public void resignFederationExecution (ResignAction resignAction)
throws
 OwnershipAcquisitionPending,
 FederateOwnsAttributes,
 FederateNotExecutionMember,
 RTIinternalError;

//4.6
public void registerFederationSynchronizationPoint (
 String synchronizationPointLabel,
 byte[] userSuppliedTag)
throws
 FederateNotExecutionMember,
 SaveInProgress,
 RestoreInProgress,
 RTIinternalError;

public void registerFederationSynchronizationPoint (
 String              synchronizationPointLabel,
 byte[]              userSuppliedTag,
 FederateHandleSet   synchronizationSet)
throws
 FederateNotExecutionMember,
 SaveInProgress,
 RestoreInProgress,
 RTIinternalError;

//4.9
public void synchronizationPointAchieved (
 String synchronizationPointLabel)
throws
 SynchronizationPointLabelNotAnnounced,
 FederateNotExecutionMember,
 SaveInProgress,
 RestoreInProgress,
 RTIinternalError;

// 4.11
public void requestFederationSave (
 String label)
throws
 FederateNotExecutionMember,
 SaveInProgress,
 RestoreInProgress,
 RTIinternalError;

public void requestFederationSave (
 String              label,
 LogicalTime         theTime)
throws
 LogicalTimeAlreadyPassed,
 InvalidLogicalTime,
 FederateUnableToUseTime,
 FederateNotExecutionMember,
 SaveInProgress,
 RestoreInProgress,
 RTIinternalError;
```



```
// 4.13
public void federateSaveBegun ()
throws
 SaveNotInitiated,
 FederateNotExecutionMember,
 RestoreInProgress,
 RTIinternalError;

// 4.14
public void federateSaveComplete ()
throws
 FederateHasNotBegunSave,
 FederateNotExecutionMember,
 RestoreInProgress,
 RTIinternalError;

public void federateSaveNotComplete ()
throws
 FederateHasNotBegunSave,
 FederateNotExecutionMember,
 RestoreInProgress,
 RTIinternalError;

// 4.16
public void queryFederationSaveStatus ()
throws
 FederateNotExecutionMember,
 SaveNotInProgress,
 RestoreInProgress,
 RTIinternalError;

// 4.18
public void requestFederationRestore (
 String label)
throws
 FederateNotExecutionMember,
 SaveInProgress,
 RestoreInProgress,
 RTIinternalError;

// 4.22
public void federateRestoreComplete ()
throws
 RestoreNotRequested,
 FederateNotExecutionMember,
 SaveInProgress,
 RTIinternalError;

public void federateRestoreNotComplete ()
throws
 RestoreNotRequested,
 FederateNotExecutionMember,
 SaveInProgress,
 RTIinternalError;

// 4.24
public void queryFederationRestoreStatus ()
throws
 FederateNotExecutionMember,
 SaveInProgress,
 RestoreNotInProgress,
 RTIinternalError;

////////////////////////////////////
// Declaration Management Services //
////////////////////////////////////

// 5.2
public void publishObjectClassAttributes (
 ObjectClassHandle   theClass,
 AttributeHandleSet  attributeList)
throws
```

```
 ObjectClassNotDefined,
 AttributeNotDefined,
 FederateNotExecutionMember,
 SaveInProgress,
 RestoreInProgress,
 RTIinternalError;

// 5.3
public void unpublishObjectClass (
 ObjectClassHandle   theClass)
throws
 ObjectClassNotDefined,
 OwnershipAcquisitionPending,
 FederateNotExecutionMember,
 SaveInProgress,
 RestoreInProgress,
 RTIinternalError;

public void unpublishObjectClassAttributes (
 ObjectClassHandle   theClass,
 AttributeHandleSet  attributeList)
throws
 ObjectClassNotDefined,
 AttributeNotDefined,
 OwnershipAcquisitionPending,
 FederateNotExecutionMember,
 SaveInProgress,
 RestoreInProgress,
 RTIinternalError;

// 5.4
public void publishInteractionClass (
 InteractionClassHandle theInteraction)
throws
 InteractionClassNotDefined,
 FederateNotExecutionMember,
 SaveInProgress,
 RestoreInProgress,
 RTIinternalError;

// 5.5
public void unpublishInteractionClass (
 InteractionClassHandle theInteraction)
throws
 InteractionClassNotDefined,
 FederateNotExecutionMember,
 SaveInProgress,
 RestoreInProgress,
 RTIinternalError;

// 5.6
public void subscribeObjectClassAttributes (
 ObjectClassHandle   theClass,
 AttributeHandleSet  attributeList)
throws
 ObjectClassNotDefined,
 AttributeNotDefined,
 FederateNotExecutionMember,
 SaveInProgress,
 RestoreInProgress,
 RTIinternalError;

public void subscribeObjectClassAttributesPassively (
 ObjectClassHandle   theClass,
 AttributeHandleSet  attributeList)
throws
 ObjectClassNotDefined,
 AttributeNotDefined,
 FederateNotExecutionMember,
 SaveInProgress,
 RestoreInProgress,
 RTIinternalError;
```



```
// 5.7
public void unsubscribeObjectClass (
 ObjectClassHandle   theClass)
throws
 ObjectClassNotDefined,
 FederateNotExecutionMember,
 SaveInProgress,
 RestoreInProgress,
 RTIinternalError;

public void unsubscribeObjectClassAttributes (
 ObjectClassHandle   theClass,
 AttributeHandleSet  attributeList)
throws
 ObjectClassNotDefined,
 AttributeNotDefined,
 FederateNotExecutionMember,
 SaveInProgress,
 RestoreInProgress,
 RTIinternalError;

// 5.8
public void subscribeInteractionClass (
 InteractionClassHandle theClass)
throws
 InteractionClassNotDefined,
 FederateServiceInvocationsAreBeingReportedViaMOM,
 FederateNotExecutionMember,
 SaveInProgress,
 RestoreInProgress,
 RTIinternalError;

public void subscribeInteractionClassPassively (
 InteractionClassHandle theClass)
throws
 InteractionClassNotDefined,
 FederateServiceInvocationsAreBeingReportedViaMOM,
 FederateNotExecutionMember,
 SaveInProgress,
 RestoreInProgress,
 RTIinternalError;

// 5.9
public void unsubscribeInteractionClass (
 InteractionClassHandle theClass)
throws
 InteractionClassNotDefined,
 FederateNotExecutionMember,
 SaveInProgress,
 RestoreInProgress,
 RTIinternalError;

////////////////////////////////
// Object Management Services //
////////////////////////////////

// 6.2
public void reserveObjectInstanceName (
 String theObjectName)
throws
 IllegalName,
 FederateNotExecutionMember,
 SaveInProgress,
 RestoreInProgress,
 RTIinternalError;

// 6.4
public ObjectInstanceHandle
registerObjectInstance (
 ObjectClassHandle   theClass)
throws
 ObjectClassNotDefined,
```

```
 ObjectClassNotPublished,
 FederateNotExecutionMember,
 SaveInProgress,
 RestoreInProgress,
 RTIinternalError;

public ObjectInstanceHandle
registerObjectInstance (
 ObjectClassHandle   theClass,
 String theObjectName)
throws
 ObjectClassNotDefined,
 ObjectClassNotPublished,
 ObjectInstanceNameNotReserved,
 ObjectInstanceNameInUse,
 FederateNotExecutionMember,
 SaveInProgress,
 RestoreInProgress,
 RTIinternalError;

// 6.6
public void updateAttributeValues (
 ObjectInstanceHandletheObject,
 AttributeHandleValueMap theAttributes,
 byte[]userSuppliedTag)
throws
 ObjectInstanceNotKnown,
 AttributeNotDefined,
 AttributeNotOwned,
 FederateNotExecutionMember,
 SaveInProgress,
 RestoreInProgress,
 RTIinternalError;

public MessageRetractionReturn
updateAttributeValues (
 ObjectInstanceHandle    theObject,
 AttributeHandleValueMaptheAttributes,
 byte[]              userSuppliedTag,
 LogicalTimetheTime)
throws
 ObjectInstanceNotKnown,
 AttributeNotDefined,
 AttributeNotOwned,
 InvalidLogicalTime,
 FederateNotExecutionMember,
 SaveInProgress,
 RestoreInProgress,
 RTIinternalError;

// 6.8
public void sendInteraction (
 InteractionClassHandle  theInteraction,
 ParameterHandleValueMap theParameters,
 byte[]                  userSuppliedTag)
throws
 InteractionClassNotPublished,
 InteractionClassNotDefined,
 InteractionParameterNotDefined,
 FederateNotExecutionMember,
 SaveInProgress,
 RestoreInProgress,
 RTIinternalError;

public MessageRetractionReturn
sendInteraction (
 InteractionClassHandle theInteraction,
 ParameterHandleValueMaptheParameters,
 byte[]                 userSuppliedTag,
 LogicalTimetheTime)
throws
 InteractionClassNotPublished,
```



```
 InteractionClassNotDefined,
 InteractionParameterNotDefined,
 InvalidLogicalTime,
 FederateNotExecutionMember,
 SaveInProgress,
 RestoreInProgress,
 RTIinternalError;

// 6.10
public void deleteObjectInstance (
 ObjectInstanceHandle objectHandle,
 byte[]                userSuppliedTag)
throws
 DeletePrivilegeNotHeld,
 ObjectInstanceNotKnown,
 FederateNotExecutionMember,
 SaveInProgress,
 RestoreInProgress,
 RTIinternalError;

public MessageRetractionReturn
deleteObjectInstance (
 ObjectInstanceHandle objectHandle,
 byte[]               userSuppliedTag,
 LogicalTime          theTime)
throws
 DeletePrivilegeNotHeld,
 ObjectInstanceNotKnown,
 InvalidLogicalTime,
 FederateNotExecutionMember,
 SaveInProgress,
 RestoreInProgress,
 RTIinternalError;

// 6.12
public void localDeleteObjectInstance (
 ObjectInstanceHandle objectHandle)
throws
 ObjectInstanceNotKnown,
 FederateOwnsAttributes,
 OwnershipAcquisitionPending,
 FederateNotExecutionMember,
 SaveInProgress,
 RestoreInProgress,
 RTIinternalError;

// 6.13
public void changeAttributeTransportationType (
 ObjectInstanceHandletheObject,
 AttributeHandleSet  theAttributes,
 TransportationType  theType)
throws
 ObjectInstanceNotKnown,
 AttributeNotDefined,
 AttributeNotOwned,
 FederateNotExecutionMember,
 SaveInProgress,
 RestoreInProgress,
 RTIinternalError;

// 6.14
public void changeInteractionTransportationType (
 InteractionClassHandle theClass,
 TransportationType  theType)
throws
 InteractionClassNotDefined,
 InteractionClassNotPublished,
 FederateNotExecutionMember,
 SaveInProgress,
 RestoreInProgress,
 RTIinternalError;
```

```
// 6.17
public void requestAttributeValueUpdate (
 ObjectInstanceHandle theObject,
 AttributeHandleSet  theAttributes,
 byte[]              userSuppliedTag)
throws
 ObjectInstanceNotKnown,
 AttributeNotDefined,
 FederateNotExecutionMember,
 SaveInProgress,
 RestoreInProgress,
 RTIinternalError;

public void requestAttributeValueUpdate (
 ObjectClassHandle   theClass,
 AttributeHandleSet  theAttributes,
 byte[]              userSuppliedTag)
throws
 ObjectClassNotDefined,
 AttributeNotDefined,
 FederateNotExecutionMember,
 SaveInProgress,
 RestoreInProgress,
 RTIinternalError;

///////////////////////////////////
// Ownership Management Services //
///////////////////////////////////

// 7.2
public void unconditionalAttributeOwnershipDivestiture (
 ObjectInstanceHandle theObject,
 AttributeHandleSet  theAttributes)
throws
 ObjectInstanceNotKnown,
 AttributeNotDefined,
 AttributeNotOwned,
 FederateNotExecutionMember,
 SaveInProgress,
 RestoreInProgress,
 RTIinternalError;

// 7.3
public void negotiatedAttributeOwnershipDivestiture (
 ObjectInstanceHandle theObject,
 AttributeHandleSet  theAttributes,
 byte[]              userSuppliedTag)
throws
 ObjectInstanceNotKnown,
 AttributeNotDefined,
 AttributeNotOwned,
 AttributeAlreadyBeingDivested,
 FederateNotExecutionMember,
 SaveInProgress,
 RestoreInProgress,
 RTIinternalError;

// 7.6
public void confirmDivestiture (
 ObjectInstanceHandle theObject,
 AttributeHandleSet  theAttributes,
 byte[]              userSuppliedTag)
throws
 ObjectInstanceNotKnown,
 AttributeNotDefined,
 AttributeNotOwned,
 AttributeDivestitureWasNotRequested,
 FederateNotExecutionMember,
 SaveInProgress,
 RestoreInProgress,
 RTIinternalError;
```



```
// 7.8
public void attributeOwnershipAcquisition (
 ObjectInstanceHandle theObject,
 AttributeHandleSet  desiredAttributes,
 byte[]              userSuppliedTag)
throws
 ObjectInstanceNotKnown,
 ObjectClassNotPublished,
 AttributeNotDefined,
 AttributeNotPublished,
 FederateOwnsAttributes,
 FederateNotExecutionMember,
 SaveInProgress,
 RestoreInProgress,
 RTIinternalError;

// 7.9
public void attributeOwnershipAcquisitionIfAvailable (
 ObjectInstanceHandle theObject,
 AttributeHandleSet  desiredAttributes)
throws
 ObjectInstanceNotKnown,
 ObjectClassNotPublished,
 AttributeNotDefined,
 AttributeNotPublished,
 FederateOwnsAttributes,
 AttributeAlreadyBeingAcquired,
 FederateNotExecutionMember,
 SaveInProgress,
 RestoreInProgress,
 RTIinternalError;

// 7.12
public AttributeHandleSet
attributeOwnershipDivestitureIfWanted (
 ObjectInstanceHandle theObject,
 AttributeHandleSet  theAttributes)
throws
 ObjectInstanceNotKnown,
 AttributeNotDefined,
 AttributeNotOwned,
 FederateNotExecutionMember,
 SaveInProgress,
 RestoreInProgress,
 RTIinternalError;

// 7.13
public void cancelNegotiatedAttributeOwnershipDivestiture (
 ObjectInstanceHandle theObject,
 AttributeHandleSet  theAttributes)
throws
 ObjectInstanceNotKnown,
 AttributeNotDefined,
 AttributeNotOwned,
 AttributeDivestitureWasNotRequested,
 FederateNotExecutionMember,
 SaveInProgress,
 RestoreInProgress,
 RTIinternalError;

// 7.14
public void cancelAttributeOwnershipAcquisition (
 ObjectInstanceHandle theObject,
 AttributeHandleSet  theAttributes)
throws
 ObjectInstanceNotKnown,
 AttributeNotDefined,
 AttributeAlreadyOwned,
 AttributeAcquisitionWasNotRequested,
 FederateNotExecutionMember,
 SaveInProgress,
 RestoreInProgress,
```

```
 RTIinternalError;

// 7.16
public void queryAttributeOwnership (
 ObjectInstanceHandle theObject,
 AttributeHandle     theAttribute)
throws
 ObjectInstanceNotKnown,
 AttributeNotDefined,
 FederateNotExecutionMember,
 SaveInProgress,
 RestoreInProgress,
 RTIinternalError;

// 7.18
public boolean
isAttributeOwnedByFederate (
 ObjectInstanceHandle theObject,
 AttributeHandle     theAttribute)
throws
 ObjectInstanceNotKnown,
 AttributeNotDefined,
 FederateNotExecutionMember,
 SaveInProgress,
 RestoreInProgress,
 RTIinternalError;

//////////////////////////////
// Time Management Services //
//////////////////////////////

// 8.2
public void enableTimeRegulation (
 LogicalTimeInterval theLookahead)
throws
 TimeRegulationAlreadyEnabled,
 InvalidLookahead,
 InTimeAdvancingState,
 RequestForTimeRegulationPending,
 FederateNotExecutionMember,
 SaveInProgress,
 RestoreInProgress,
 RTIinternalError;

// 8.4
public void disableTimeRegulation ()
throws
 TimeRegulationIsNotEnabled,
 FederateNotExecutionMember,
 SaveInProgress,
 RestoreInProgress,
 RTIinternalError;

// 8.5
public void enableTimeConstrained ()
throws
 TimeConstrainedAlreadyEnabled,
 InTimeAdvancingState,
 RequestForTimeConstrainedPending,
 FederateNotExecutionMember,
 SaveInProgress,
 RestoreInProgress,
 RTIinternalError;

// 8.7
public void disableTimeConstrained ()
throws
 TimeConstrainedIsNotEnabled,
 FederateNotExecutionMember,
 SaveInProgress,
 RestoreInProgress,
 RTIinternalError;
```



```
// 8.8
public void timeAdvanceRequest (
 LogicalTime     theTime)
throws
 InvalidLogicalTime,
 LogicalTimeAlreadyPassed,
 InTimeAdvancingState,
 RequestForTimeRegulationPending,
 RequestForTimeConstrainedPending,
 FederateNotExecutionMember,
 SaveInProgress,
 RestoreInProgress,
 RTIinternalError;

// 8.9
public void timeAdvanceRequestAvailable (
 LogicalTime     theTime)
throws
 InvalidLogicalTime,
 LogicalTimeAlreadyPassed,
 InTimeAdvancingState,
 RequestForTimeRegulationPending,
 RequestForTimeConstrainedPending,
 FederateNotExecutionMember,
 SaveInProgress,
 RestoreInProgress,
 RTIinternalError;

// 8.10
public void nextMessageRequest (
 LogicalTime     theTime)
throws
 InvalidLogicalTime,
 LogicalTimeAlreadyPassed,
 InTimeAdvancingState,
 RequestForTimeRegulationPending,
 RequestForTimeConstrainedPending,
 FederateNotExecutionMember,
 SaveInProgress,
 RestoreInProgress,
 RTIinternalError;

// 8.11
public void nextMessageRequestAvailable (
 LogicalTime     theTime)
throws
 InvalidLogicalTime,
 LogicalTimeAlreadyPassed,
 InTimeAdvancingState,
 RequestForTimeRegulationPending,
 RequestForTimeConstrainedPending,
 FederateNotExecutionMember,
 SaveInProgress,
 RestoreInProgress,
 RTIinternalError;

// 8.12
public void flushQueueRequest (
 LogicalTime     theTime)
throws
 InvalidLogicalTime,
 LogicalTimeAlreadyPassed,
 InTimeAdvancingState,
 RequestForTimeRegulationPending,
 RequestForTimeConstrainedPending,
 FederateNotExecutionMember,
 SaveInProgress,
 RestoreInProgress,
 RTIinternalError;

// 8.14
public void enableAsynchronousDelivery()
```

```
throws
 AsynchronousDeliveryAlreadyEnabled,
 FederateNotExecutionMember,
 SaveInProgress,
 RestoreInProgress,
 RTIinternalError;

// 8.15
public void disableAsynchronousDelivery()
throws
 AsynchronousDeliveryAlreadyDisabled,
 FederateNotExecutionMember,
 SaveInProgress,
 RestoreInProgress,
 RTIinternalError;

// 8.16
public TimeQueryReturn
queryGALT ()
throws
 FederateNotExecutionMember,
 SaveInProgress,
 RestoreInProgress,
 RTIinternalError;

// 8.17
public LogicalTime queryLogicalTime ()
throws
 FederateNotExecutionMember,
 SaveInProgress,
 RestoreInProgress,
 RTIinternalError;

// 8.18
public TimeQueryReturn
queryLITS ()
throws
 FederateNotExecutionMember,
 SaveInProgress,
 RestoreInProgress,
 RTIinternalError;

// 8.19
public void modifyLookahead (
 LogicalTimeInterval theLookahead)
throws
 TimeRegulationIsNotEnabled,
 InvalidLookahead,
 InTimeAdvancingState,
 FederateNotExecutionMember,
 SaveInProgress,
 RestoreInProgress,
 RTIinternalError;

// 8.20
public LogicalTimeInterval queryLookahead ()
throws
 TimeRegulationIsNotEnabled,
 FederateNotExecutionMember,
 SaveInProgress,
 RestoreInProgress,
 RTIinternalError;

// 8.21
public void retract (
 MessageRetractionHandle theHandle)
throws
 InvalidMessageRetractionHandle,
 TimeRegulationIsNotEnabled,
 MessageCanNoLongerBeRetracted,
 FederateNotExecutionMember,
 SaveInProgress,
```



```
 RestoreInProgress,
 RTIinternalError;

// 8.23
public void changeAttributeOrderType (
 ObjectInstanceHandle theObject,
 AttributeHandleSet  theAttributes,
 OrderType            theType)
throws
 ObjectInstanceNotKnown,
 AttributeNotDefined,
 AttributeNotOwned,
 FederateNotExecutionMember,
 SaveInProgress,
 RestoreInProgress,
 RTIinternalError;

// 8.24
public void changeInteractionOrderType (
 InteractionClassHandle theClass,
 OrderType           theType)
throws
 InteractionClassNotDefined,
 InteractionClassNotPublished,
 FederateNotExecutionMember,
 SaveInProgress,
 RestoreInProgress,
 RTIinternalError;

///////////////////////////////////
// Data Distribution Management //
///////////////////////////////////

// 9.2
public RegionHandle
createRegion (DimensionHandleSet dimensions)
throws
 InvalidDimensionHandle,
 FederateNotExecutionMember,
 SaveInProgress,
 RestoreInProgress,
 RTIinternalError;

// 9.3
public void commitRegionModifications (
 RegionHandleSet regions)
throws
 InvalidRegion,
 RegionNotCreatedByThisFederate,
 FederateNotExecutionMember,
 SaveInProgress,
 RestoreInProgress,
 RTIinternalError;

// 9.4
public void deleteRegion (
 RegionHandle    theRegion)
throws
 InvalidRegion,
 RegionNotCreatedByThisFederate,
 RegionInUseForUpdateOrSubscription,
 FederateNotExecutionMember,
 SaveInProgress,
 RestoreInProgress,
 RTIinternalError;

//9.5
public ObjectInstanceHandle
registerObjectInstanceWithRegions (
 ObjectClassHandle            theClass,
 AttributeSetRegionSetPairListattributesAndRegions)
throws
```

```
 ObjectClassNotDefined,
 ObjectClassNotPublished,
 AttributeNotDefined,
 AttributeNotPublished,
 InvalidRegion,
 RegionNotCreatedByThisFederate,
 InvalidRegionContext,
 FederateNotExecutionMember,
 SaveInProgress,
 RestoreInProgress,
 RTIinternalError;

public ObjectInstanceHandle
registerObjectInstanceWithRegions (
 ObjectClassHandle            theClass,
 AttributeSetRegionSetPairList attributesAndRegions,
 String                       theObject)
throws
 ObjectClassNotDefined,
 ObjectClassNotPublished,
 AttributeNotDefined,
 AttributeNotPublished,
 InvalidRegion,
 RegionNotCreatedByThisFederate,
 InvalidRegionContext,
 ObjectInstanceNameNotReserved,
 ObjectInstanceNameInUse,
 FederateNotExecutionMember,
 SaveInProgress,
 RestoreInProgress,
 RTIinternalError;

// 9.6
public void associateRegionsForUpdates (
 ObjectInstanceHandle         theObject,
 AttributeSetRegionSetPairListattributesAndRegions)
throws
 ObjectInstanceNotKnown,
 AttributeNotDefined,
 InvalidRegion,
 RegionNotCreatedByThisFederate,
 InvalidRegionContext,
 FederateNotExecutionMember,
 SaveInProgress,
 RestoreInProgress,
 RTIinternalError;

// 9.7
public void unassociateRegionsForUpdates (
 ObjectInstanceHandle         theObject,
 AttributeSetRegionSetPairListattributesAndRegions)
throws
 ObjectInstanceNotKnown,
 AttributeNotDefined,
 InvalidRegion,
 RegionNotCreatedByThisFederate,
 FederateNotExecutionMember,
 SaveInProgress,
 RestoreInProgress,
 RTIinternalError;

// 9.8
public void subscribeObjectClassAttributesWithRegions (
 ObjectClassHandle            theClass,
 AttributeSetRegionSetPairList attributesAndRegions)
throws
 ObjectClassNotDefined,
 AttributeNotDefined,
 InvalidRegion,
 RegionNotCreatedByThisFederate,
 InvalidRegionContext,
 FederateNotExecutionMember,
```



```
 SaveInProgress,
 RestoreInProgress,
 RTIinternalError;

public void subscribeObjectClassAttributesPassivelyWithRegions (
 ObjectClassHandle            theClass,
 AttributeSetRegionSetPairListattributesAndRegions)
throws
 ObjectClassNotDefined,
 AttributeNotDefined,
 InvalidRegion,
 RegionNotCreatedByThisFederate,
 InvalidRegionContext,
 FederateNotExecutionMember,
 SaveInProgress,
 RestoreInProgress,
 RTIinternalError;

// 9.9
public void unsubscribeObjectClassAttributesWithRegions (
 ObjectClassHandle            theClass,
 AttributeSetRegionSetPairListattributesAndRegions)
throws
 ObjectClassNotDefined,
 AttributeNotDefined,
 InvalidRegion,
 RegionNotCreatedByThisFederate,
 FederateNotExecutionMember,
 SaveInProgress,
 RestoreInProgress,
 RTIinternalError;

// 9.10
public void subscribeInteractionClassWithRegions (
 InteractionClassHandletheClass,
 RegionHandleSet     regions)
throws
 InteractionClassNotDefined,
 InvalidRegion,
 RegionNotCreatedByThisFederate,
 InvalidRegionContext,
 FederateServiceInvocationsAreBeingReportedViaMOM,
 FederateNotExecutionMember,
 SaveInProgress,
 RestoreInProgress,
 RTIinternalError;

public void subscribeInteractionClassPassivelyWithRegions (
 InteractionClassHandle theClass,
 RegionHandleSet     regions)
throws
 InteractionClassNotDefined,
 InvalidRegion,
 RegionNotCreatedByThisFederate,
 InvalidRegionContext,
 FederateServiceInvocationsAreBeingReportedViaMOM,
 FederateNotExecutionMember,
 SaveInProgress,
 RestoreInProgress,
 RTIinternalError;

// 9.11
public void unsubscribeInteractionClassWithRegions (
 InteractionClassHandle theClass,
 RegionHandleSet     regions)
throws
 InteractionClassNotDefined,
 InvalidRegion,
 RegionNotCreatedByThisFederate,
 FederateNotExecutionMember,
 SaveInProgress,
 RestoreInProgress,
```

```
 RTIinternalError;

//9.12
public void sendInteractionWithRegions (
 InteractionClassHandletheInteraction,
 ParameterHandleValueMap theParameters,
 RegionHandleSet     regions,
 byte[]              userSuppliedTag)
throws
 InteractionClassNotDefined,
 InteractionClassNotPublished,
 InteractionParameterNotDefined,
 InvalidRegion,
 RegionNotCreatedByThisFederate,
 InvalidRegionContext,
 FederateNotExecutionMember,
 SaveInProgress,
 RestoreInProgress,
 RTIinternalError;

public MessageRetractionReturn
sendInteractionWithRegions (
 InteractionClassHandle  theInteraction,
 ParameterHandleValueMap theParameters,
 RegionHandleSet     regions,
 byte[]              userSuppliedTag,
 LogicalTime         theTime)
throws
 InteractionClassNotDefined,
 InteractionClassNotPublished,
 InteractionParameterNotDefined,
 InvalidRegion,
 RegionNotCreatedByThisFederate,
 InvalidRegionContext,
 InvalidLogicalTime,
 FederateNotExecutionMember,
 SaveInProgress,
 RestoreInProgress,
 RTIinternalError;

// 9.13
public void requestAttributeValueUpdateWithRegions (
 ObjectClassHandletheClass,
 AttributeSetRegionSetPairList attributesAndRegions,
 byte[]userSuppliedTag)
throws
 ObjectClassNotDefined,
 AttributeNotDefined,
 InvalidRegion,
 RegionNotCreatedByThisFederate,
 InvalidRegionContext,
 FederateNotExecutionMember,
 SaveInProgress,
 RestoreInProgress,
 RTIinternalError;

//////////////////////////
// RTI Support Services //
//////////////////////////

// 10.2
public ObjectClassHandle
getObjectClassHandle (
 String theName)
throws
 NameNotFound,
 FederateNotExecutionMember,
 RTIinternalError;

// 10.3
public String
getObjectClassName (
```



```
 ObjectClassHandle theHandle)
throws
 InvalidObjectClassHandle,
 FederateNotExecutionMember,
 RTIinternalError;

// 10.4
public AttributeHandle
getAttributeHandle (
 ObjectClassHandle whichClass,
 String theName)
throws
 InvalidObjectClassHandle,
 NameNotFound,
 FederateNotExecutionMember,
 RTIinternalError;

// 10.5
public String
getAttributeName (
 ObjectClassHandle whichClass,
 AttributeHandletheHandle)
throws
 InvalidObjectClassHandle,
 InvalidAttributeHandle,
 AttributeNotDefined,
 FederateNotExecutionMember,
 RTIinternalError;

// 10.6
public InteractionClassHandle
getInteractionClassHandle (
 String          theName)
throws
 NameNotFound,
 FederateNotExecutionMember,
 RTIinternalError;

// 10.7
public String
getInteractionClassName (
 InteractionClassHandle theHandle)
throws
 InvalidInteractionClassHandle,
 FederateNotExecutionMember,
 RTIinternalError;

// 10.8
public ParameterHandle
getParameterHandle (
 InteractionClassHandle whichClass,
 String               theName)
throws
 InvalidInteractionClassHandle,
 NameNotFound,
 FederateNotExecutionMember,
 RTIinternalError;

// 10.9
public String
getParameterName (
 InteractionClassHandle whichClass,
 ParameterHandle     theHandle)
throws
 InvalidInteractionClassHandle,
 InvalidParameterHandle,
 InteractionParameterNotDefined,
 FederateNotExecutionMember,
 RTIinternalError;

// 10.10
public ObjectInstanceHandle
```

```
getObjectInstanceHandle (
 String       theName)
throws
 ObjectInstanceNotKnown,
 FederateNotExecutionMember,
 RTIinternalError;

// 10.11
public String
getObjectInstanceName (
 ObjectInstanceHandle theHandle)
throws
 ObjectInstanceNotKnown,
 FederateNotExecutionMember,
 RTIinternalError;

// 10.12
public DimensionHandle
getDimensionHandle (
 String           theName)
throws
 NameNotFound,
 ederateNotExecutionMember,
 RTIinternalError;

// 10.13
public String
getDimensionName (
 DimensionHandle     theHandle)
throws
 InvalidDimensionHandle,
 FederateNotExecutionMember,
 RTIinternalError;

// 10.14
public long
getDimensionUpperBound (
 DimensionHandle     theHandle)
throws
 InvalidDimensionHandle,
 FederateNotExecutionMember,
 RTIinternalError;

// 10.15
public DimensionHandleSet
getAvailableDimensionsForClassAttribute (
 ObjectClassHandle whichClass,
 AttributeHandle     theHandle)
throws
 InvalidObjectClassHandle,
 InvalidAttributeHandle,
 AttributeNotDefined,
 FederateNotExecutionMember,
 RTIinternalError;

// 10.16
public ObjectClassHandle
getKnownObjectClassHandle (
 ObjectInstanceHandle theObject)
throws
 ObjectInstanceNotKnown,
 FederateNotExecutionMember,
 RTIinternalError;

// 10.17
public DimensionHandleSet
getAvailableDimensionsForInteractionClass (
 InteractionClassHandle theHandle)
throws
 InvalidInteractionClassHandle,
 FederateNotExecutionMember,
 RTIinternalError;
```



```
// 10.18
public TransportationType
getTransportationType (
 String              theName)
throws
 InvalidTransportationName,
 FederateNotExecutionMember,
 RTIinternalError;

// 10.19
public String
getTransportationName (
 TransportationType  theType)
throws
 InvalidTransportationType,
 FederateNotExecutionMember,
 RTIinternalError;

// 10.20
public OrderType
getOrderType (
 String     theName)
throws
 InvalidOrderName,
 FederateNotExecutionMember,
 RTIinternalError;

// 10.21
public String
getOrderName (
 OrderType    theHandle)
throws
 InvalidOrderType,
 FederateNotExecutionMember,
 RTIinternalError;

// 10.22
public void enableObjectClassRelevanceAdvisorySwitch()
throws
 FederateNotExecutionMember,
 ObjectClassRelevanceAdvisorySwitchIsOn,
 SaveInProgress,
 RestoreInProgress,
 RTIinternalError;

// 10.23
public void disableObjectClassRelevanceAdvisorySwitch()
throws
 ObjectClassRelevanceAdvisorySwitchIsOff,
 FederateNotExecutionMember,
 SaveInProgress,
 RestoreInProgress,
 RTIinternalError;

// 10.24
public void enableAttributeRelevanceAdvisorySwitch()
throws
 AttributeRelevanceAdvisorySwitchIsOn,
 FederateNotExecutionMember,
 SaveInProgress,
 RestoreInProgress,
 RTIinternalError;

// 10.25
public void disableAttributeRelevanceAdvisorySwitch()
throws
 AttributeRelevanceAdvisorySwitchIsOff,
 FederateNotExecutionMember,
 SaveInProgress,
 RestoreInProgress,
 RTIinternalError;
```

```
// 10.26
public void enableAttributeScopeAdvisorySwitch()
throws
 AttributeScopeAdvisorySwitchIsOn,
 FederateNotExecutionMember,
 SaveInProgress,
 RestoreInProgress,
 RTIinternalError;

// 10.27
public void disableAttributeScopeAdvisorySwitch()
throws
 AttributeScopeAdvisorySwitchIsOff,
 FederateNotExecutionMember,
 SaveInProgress,
 RestoreInProgress,
 RTIinternalError;

// 10.28
public void enableInteractionRelevanceAdvisorySwitch()
throws
 InteractionRelevanceAdvisorySwitchIsOn,
 FederateNotExecutionMember,
 SaveInProgress,
 RestoreInProgress,
 RTIinternalError;

// 10.29
public void disableInteractionRelevanceAdvisorySwitch()
throws
 InteractionRelevanceAdvisorySwitchIsOff,
 FederateNotExecutionMember,
 SaveInProgress,
 RestoreInProgress,
 RTIinternalError;

// 10.30
public DimensionHandleSet
 getDimensionHandleSet(RegionHandle region)
throws
 InvalidRegion,
 FederateNotExecutionMember,
 SaveInProgress,
 RestoreInProgress,
 RTIinternalError;

// 10.31
public RangeBounds
 getRangeBounds(
 RegionHandle region,
 DimensionHandle dimension)
throws
 InvalidRegion,
 RegionDoesNotContainSpecifiedDimension,
 FederateNotExecutionMember,
 SaveInProgress,
 RestoreInProgress,
 RTIinternalError;

// 10.32
public void setRangeBounds(
 RegionHandle region,
 DimensionHandle dimension,
 RangeBounds bounds)
throws
 InvalidRegion,
 RegionNotCreatedByThisFederate,
 RegionDoesNotContainSpecifiedDimension,
 InvalidRangeBound,
 FederateNotExecutionMember,
 SaveInProgress,
 RestoreInProgress,
```



```
 RTIinternalError;

// 10.33
public long
normalizeFederateHandle(
 FederateHandle federateHandle)
throws
 InvalidFederateHandle,
 FederateNotExecutionMember,
 RTIinternalError;

// 10.34
public long
normalizeServiceGroup(
 ServiceGroup group)
throws
 FederateNotExecutionMember,
 RTIinternalError;

// 10.35
public java.util.Properties
initializeRTI(
 java.util.Properties properties)
throws
 InitializePreviouslyInvoked,
 BadInitializationParameter,
 RTIinternalError;

// 10.36
public void finalizeRTI()
throws
 InitializeNeverInvoked,
 SomeFederateJoinedToAnExecution,
 RTIinternalError;

// 10.37
public boolean
evokeCallback(
 double seconds)
throws
 FederateNotExecutionMember,
 RTIinternalError;

// 10.38
public boolean
evokeMultipleCallbacks(
 double minimumTime,
 double maximumTime)
throws
 FederateNotExecutionMember,
 RTIinternalError;

// 10.39
public void enableCallbacks()
throws
 FederateNotExecutionMember,
 SaveInProgress,
 RestoreInProgress,
 RTIinternalError;

// 10.40
public void disableCallbacks()
throws
 FederateNotExecutionMember,
 SaveInProgress,
 RestoreInProgress,
 RTIinternalError;
```

```
//API-specific services
public AttributeHandleFactory.             getAttributeHandleFactory();
public AttributeHandleSetFactory           getAttributeHandleSetFactory();
public AttributeHandleValueMapFactory      getAttributeHandleValueMapFactory();
public AttributeSetRegionSetPairListFactorygetAttributeSetRegionSetPairListFactory();
public DimensionHandleFactory              getDimensionHandleFactory();
public DimensionHandleSetFactory           getDimensionHandleSetFactory();
public FederateHandleFactory               getFederateHandleFactory();
public FederateHandleSetFactory            getFederateHandleSetFactory();
public InteractionClassHandleFactory       getInteractionClassHandleFactory();
public ObjectClassHandleFactory            getObjectClassHandleFactory();
public ObjectInstanceHandleFactory         getObjectInstanceHandleFactory();
public ParameterHandleFactory              getParameterHandleFactory();
public ParameterHandleValueMapFactory      getParameterHandleValueMapFactory();
public RegionHandleSetFactory              getRegionHandleSetFactory();

public String getHLAversion();
}
//end RTIambassador


//File: FederateAmbassador.java

package hla.rti;

/**
* Federate must implement this interface.
*/

public interface FederateAmbassador {


////////////////////////////////////
// Federation Management Services //
////////////////////////////////////

//4.7
public void synchronizationPointRegistrationSucceeded(
 String synchronizationPointLabel)
throws
 FederateInternalError;

public void synchronizationPointRegistrationFailed(
 String               synchronizationPointLabel,
 SynchronizationPointFailureReason reason)
throws
 FederateInternalError;

//4.8
public void announceSynchronizationPoint(
 String synchronizationPointLabel,
 byte[] userSuppliedTag)
throws
 FederateInternalError;

//4.10
public void federationSynchronized(
 String synchronizationPointLabel)
throws
 FederateInternalError;

//4.12
public void initiateFederateSave(
 String label)
throws
 UnableToPerformSave,
 FederateInternalError;

public void initiateFederateSave(
 Stringlabel,
 LogicalTime time)
throws
 InvalidLogicalTime,
 UnableToPerformSave,
 FederateInternalError;
```



```
// 4.15
public void federationSaved ()
throws
 FederateInternalError;

public void federationNotSaved (
 SaveFailureReason   reason)
throws
 FederateInternalError;

// 4.17
public void federationSaveStatusResponse (
 FederateHandleSaveStatusPair[] response)
throws
 FederateInternalError;

// 4.19
public void requestFederationRestoreSucceeded (
 String label)
throws
 FederateInternalError;

public void requestFederationRestoreFailed (
 String label)
throws
 FederateInternalError;

// 4.20
public void federationRestoreBegun ()
throws
 FederateInternalError;

// 4.21
public void initiateFederateRestore (
 String       label,
 FederateHandle federateHandle)
throws
 SpecifiedSaveLabelDoesNotExist,
 CouldNotInitiateRestore,
 FederateInternalError;

// 4.23
public void federationRestored ()
throws
 FederateInternalError;

public void federationNotRestored (
 RestoreFailureReason reason)
throws
 FederateInternalError;

// 4.25
public void federationRestoreStatusResponse (
 FederateHandleRestoreStatusPair[] response)
throws
 FederateInternalError;

////////////////////////////////////
// Declaration Management Services //
////////////////////////////////////

// 5.10
public void startRegistrationForObjectClass (
 ObjectClassHandle   theClass)
throws
 ObjectClassNotPublished,
 FederateInternalError;

// 5.11
public void stopRegistrationForObjectClass (
  ObjectClassHandle theClass)
throws
```

```
 ObjectClassNotPublished,
 FederateInternalError;

// 5.12
public void turnInteractionsOn (
 InteractionClassHandle theHandle)
throws
 InteractionClassNotPublished,
 FederateInternalError;

// 5.13
public void turnInteractionsOff (
 InteractionClassHandle theHandle)
throws
 InteractionClassNotPublished,
 FederateInternalError;

////////////////////////////////
// Object Management Services //
////////////////////////////////

// 6.3
public void objectInstanceNameReservationSucceded (
 String objectName)
throws
 UnknownName,
 FederateInternalError;

public void objectInstanceNameReservationFailed (
 String objectName)
throws
 UnknownName,
 FederateInternalError;

// 6.5
public void discoverObjectInstance (
 ObjectInstanceHandletheObject,
 ObjectClassHandle   theObjectClass,
 StringobjectName)
throws
 CouldNotDiscover,
 ObjectClassNotRecognized,
 FederateInternalError;

// 6.7
public void reflectAttributeValues (
 ObjectInstanceHandle         theObject,
 AttributeHandleValueMap      theAttributes,
 byte[]                       userSuppliedTag,
 OrderType                    sentOrdering,
 TransportationType           theTransport)
throws
 ObjectInstanceNotKnown,
 AttributeNotRecognized,
 AttributeNotSubscribed,
 FederateInternalError;

public void reflectAttributeValues (
 ObjectInstanceHandle         theObject,
 AttributeHandleValueMap      theAttributes,
 byte[]                       userSuppliedTag,
 OrderType                    sentOrdering,
 TransportationType           theTransport,
 RegionHandleSet              sentRegions)
throws
 ObjectInstanceNotKnown,
 AttributeNotRecognized,
 AttributeNotSubscribed,
 FederateInternalError;
```



```
public void reflectAttributeValues (
 ObjectInstanceHandle         theObject,
 AttributeHandleValueMap      theAttributes,
 byte[]                       userSuppliedTag,
 OrderType                    sentOrdering,
 TransportationType           theTransport,
 LogicalTime                  theTime,
 OrderType                    receivedOrdering)
throws
 ObjectInstanceNotKnown,
 AttributeNotRecognized,
 AttributeNotSubscribed,
 FederateInternalError;

public void reflectAttributeValues (
 ObjectInstanceHandle         theObject,
 AttributeHandleValueMap      theAttributes,
 byte[]                       userSuppliedTag,
 OrderType                    sentOrdering,
 TransportationType           theTransport,
 LogicalTime                  theTime,
 OrderType                    receivedOrdering,
 RegionHandleSet              sentRegions)
throws
 ObjectInstanceNotKnown,
 AttributeNotRecognized,
 AttributeNotSubscribed,
 FederateInternalError;

public void reflectAttributeValues (
 ObjectInstanceHandle         theObject,
 AttributeHandleValueMap      theAttributes,
 byte[]                       userSuppliedTag,
 OrderType                    sentOrdering,
 TransportationType           theTransport,
 LogicalTime                  theTime,
 OrderType                    receivedOrdering,
 MessageRetractionHandle      retractionHandle)
throws
 ObjectInstanceNotKnown,
 AttributeNotRecognized,
 AttributeNotSubscribed,
 InvalidLogicalTime,
 FederateInternalError;

public void reflectAttributeValues (
 ObjectInstanceHandle         theObject,
 AttributeHandleValueMap      theAttributes,
 byte[]                       userSuppliedTag,
 OrderType                    sentOrdering,
 TransportationType           theTransport,
 LogicalTime                  theTime,
 OrderType                    receivedOrdering,
 MessageRetractionHandle      retractionHandle,
 RegionHandleSet              sentRegions)
throws
 ObjectInstanceNotKnown, AttributeNotRecognized,
 AttributeNotSubscribed,
 InvalidLogicalTime,
 FederateInternalError;

// 6.9
public void receiveInteraction (
 InteractionClassHandle       interactionClass,
 ParameterHandleValueMap      theParameters,
 byte[]                       userSuppliedTag,
 OrderType                    sentOrdering,
 TransportationType           theTransport)
throws
 InteractionClassNotRecognized,
 InteractionParameterNotRecognized,
 InteractionClassNotSubscribed,
```

```
 FederateInternalError;

public void receiveInteraction (
 InteractionClassHandle       interactionClass,
 ParameterHandleValueMap      theParameters,
 byte[]                       userSuppliedTag,
 OrderType                    sentOrdering,
 TransportationType           theTransport,
 RegionHandleSet              sentRegions)
throws
 InteractionClassNotRecognized,
 InteractionParameterNotRecognized,
 InteractionClassNotSubscribed,
 FederateInternalError;

public void receiveInteraction (
 InteractionClassHandle       interactionClass,
 ParameterHandleValueMap      theParameters,
 byte[]                       userSuppliedTag,
 OrderType                    sentOrdering,
 TransportationType           theTransport,
 LogicalTime                  theTime,
 OrderType                    receivedOrdering)
throws
 InteractionClassNotRecognized,
 InteractionParameterNotRecognized,
 InteractionClassNotSubscribed,
 FederateInternalError;

public void receiveInteraction (
 InteractionClassHandle       interactionClass,
 ParameterHandleValueMap      theParameters,
 byte[]                       userSuppliedTag,
 OrderType                    sentOrdering,
 TransportationType           theTransport,
 LogicalTime                  theTime,
 OrderType                    receivedOrdering,
 RegionHandleSet              regions)
throws
 InteractionClassNotRecognized,
 InteractionParameterNotRecognized,
 InteractionClassNotSubscribed,
 FederateInternalError;

public void receiveInteraction (
 InteractionClassHandle       interactionClass,
 ParameterHandleValueMap      theParameters,
 byte[]                       userSuppliedTag,
 OrderType                    sentOrdering,
 TransportationType           theTransport,
 LogicalTime                  theTime,
 OrderType                    receivedOrdering,
 MessageRetractionHandle      messageRetractionHandle)
throws
 InteractionClassNotRecognized,
 InteractionParameterNotRecognized,
 InteractionClassNotSubscribed,
 InvalidLogicalTime,
 FederateInternalError;

public void receiveInteraction (
 InteractionClassHandle       interactionClass,
 ParameterHandleValueMap      theParameters,
 byte[]                       userSuppliedTag,
 OrderType                    sentOrdering,
 TransportationType           theTransport,
 LogicalTime                  theTime,
 OrderType                    receivedOrdering,
 MessageRetractionHandle      messageRetractionHandle,
 RegionHandleSet              sentRegions)
throws
 InteractionClassNotRecognized,
```



```
 InteractionParameterNotRecognized,
 InteractionClassNotSubscribed,
 InvalidLogicalTime,
 FederateInternalError;

// 6.11
public void removeObjectInstance (
 ObjectInstanceHandle theObject,
 byte[]               userSuppliedTag,
 OrderType            sentOrdering)
throws
 ObjectInstanceNotKnown,
 FederateInternalError;

public void removeObjectInstance (
 ObjectInstanceHandle         theObject,
 byte[]                       userSuppliedTag,
 OrderType                    sentOrdering,
 LogicalTime                  theTime,
 OrderType                    receivedOrdering)
throws
 ObjectInstanceNotKnown,
 FederateInternalError;

public void removeObjectInstance (
 ObjectInstanceHandle         theObject,
 byte[]                       userSuppliedTag,
 OrderType                    sentOrdering,
 LogicalTime                  theTime,
 OrderType                    receivedOrdering,
 MessageRetractionHandle      retractionHandle)
throws
 ObjectInstanceNotKnown,
 InvalidLogicalTime,
 FederateInternalError;

// 6.15
public void attributesInScope (
 ObjectInstanceHandle theObject,
 AttributeHandleSet  theAttributes)
throws
 ObjectInstanceNotKnown,
 AttributeNotRecognized,
 AttributeNotSubscribed,
 FederateInternalError;

// 6.16
public void attributesOutOfScope (
 ObjectInstanceHandle theObject,
 AttributeHandleSet  theAttributes)
throws
 ObjectInstanceNotKnown,
 AttributeNotRecognized,
 AttributeNotSubscribed,
 FederateInternalError;

// 6.18
public void provideAttributeValueUpdate (
 ObjectInstanceHandle theObject,
 AttributeHandleSet  theAttributes,
 byte[]              userSuppliedTag)
throws
 ObjectInstanceNotKnown,
 AttributeNotRecognized,
 AttributeNotOwned,
 FederateInternalError;

// 6.19
public void turnUpdatesOnForObjectInstance (
 ObjectInstanceHandle         theObject,
 AttributeHandleSet           theAttributes)
throws
```

```
 ObjectInstanceNotKnown,
 AttributeNotRecognized,
 AttributeNotOwned,
 FederateInternalError;

// 6.20
public void turnUpdatesOffForObjectInstance (
 ObjectInstanceHandle         theObject,
 AttributeHandleSet           theAttributes)
throws
 ObjectInstanceNotKnown,
 AttributeNotRecognized,
 AttributeNotOwned,
 FederateInternalError;

///////////////////////////////////
// Ownership Management Services //
///////////////////////////////////

// 7.4
public void requestAttributeOwnershipAssumption (
 ObjectInstanceHandle         theObject,
 AttributeHandleSet           offeredAttributes,
 byte[]                       userSuppliedTag)
throws
 ObjectInstanceNotKnown,
 AttributeNotRecognized,
 AttributeAlreadyOwned,
 AttributeNotPublished,
 FederateInternalError;

// 7.5
public void requestDivestitureConfirmation (
 ObjectInstanceHandle theObject,
 AttributeHandleSet  offeredAttributes)
throws
 ObjectInstanceNotKnown,
 AttributeNotRecognized,
 AttributeNotOwned,
 AttributeDivestitureWasNotRequested,
 FederateInternalError;

// 7.7
public void attributeOwnershipAcquisitionNotification (
 ObjectInstanceHandle theObject,
 AttributeHandleSet  securedAttributes,
 byte[]              userSuppliedTag)
throws
 ObjectInstanceNotKnown,
 AttributeNotRecognized,
 AttributeAcquisitionWasNotRequested,
 AttributeAlreadyOwned,
 AttributeNotPublished,
 FederateInternalError;

// 7.10
public void attributeOwnershipUnavailable (
 ObjectInstanceHandle theObject,
 AttributeHandleSet  theAttributes)
throws
 ObjectInstanceNotKnown,
 AttributeNotRecognized,
 AttributeAlreadyOwned,
 AttributeAcquisitionWasNotRequested,
 FederateInternalError;

// 7.11
public void requestAttributeOwnershipRelease (
 ObjectInstanceHandle theObject,
 AttributeHandleSet  candidateAttributes,
 byte[]              userSuppliedTag)
throws
```



```
 ObjectInstanceNotKnown,
 AttributeNotRecognized,
 AttributeNotOwned,
 FederateInternalError;

// 7.15
public void confirmAttributeOwnershipAcquisitionCancellation (
 ObjectInstanceHandle theObject,
 AttributeHandleSet  theAttributes)
throws
 ObjectInstanceNotKnown,
 AttributeNotRecognized,
 AttributeAlreadyOwned,
 AttributeAcquisitionWasNotCanceled,
 FederateInternalError;

// 7.17
public void informAttributeOwnership (
 ObjectInstanceHandle theObject,
 AttributeHandle     theAttribute,
 FederateHandle      theOwner)
throws
 ObjectInstanceNotKnown,
 AttributeNotRecognized,
 FederateInternalError;

public void attributeIsNotOwned (
 ObjectInstanceHandle theObject,
 AttributeHandle     theAttribute)
throws
 ObjectInstanceNotKnown,
 AttributeNotRecognized,
 FederateInternalError;

public void attributeIsOwnedByRTI (
 ObjectInstanceHandle theObject,
 AttributeHandle     theAttribute)
throws
  ObjectInstanceNotKnown,
  AttributeNotRecognized,
  FederateInternalError;

//////////////////////////////
// Time Management Services //
//////////////////////////////

// 8.3
public void timeRegulationEnabled (
 LogicalTime     time)
throws
 InvalidLogicalTime,
 NoRequestToEnableTimeRegulationWasPending,
 FederateInternalError;

// 8.6
public void timeConstrainedEnabled (
 LogicalTime     time)
throws
 InvalidLogicalTime,
 NoRequestToEnableTimeConstrainedWasPending,
 FederateInternalError;

// 8.13
public void timeAdvanceGrant (
 LogicalTime     theTime)
throws
 InvalidLogicalTime,
 JoinedFederateIsNotInTimeAdvancingState,
 FederateInternalError;
```

```
// 8.22
public void requestRetraction (
 MessageRetractionHandle theHandle)
throws
 FederateInternalError;
}
//end FederateAmbassador



//File: AsynchronousDeliveryAlreadyDisabled.java

package hla.rti;

/**
 * Public exception class AsynchronousDeliveryAlreadyDisabled
*/
public final class AsynchronousDeliveryAlreadyDisabled extends RTIexception {
 public AsynchronousDeliveryAlreadyDisabled(String msg) {
  super(msg);
 }
}



//File: AttributeHandle.java

package hla.rti;

/**
 * Type-safe handle for an attribute. Generally these are created by the
 * RTI and passed to the user.
 */

public interface AttributeHandle extends java.io.Serializable {

/**
 * @return true if this refers to the same attribute as other handle
 */
public boolean equals(Object otherAttributeHandle);

/**
 * @return int. All instances that refer to the same attribute should return the
 * same hashcode.
 */
public int hashCode();

 public int encodedLength();
 public void encode(byte[] buffer, int offset);

 public String toString();

}
//end AttributeHandle



//File: AttributeHandleFactory.java

package hla.rti;

/**
 * The factory is used only (outside RTI) to create AttributeHandle
 * received as an attribute value or parameter value.
 */

public interface AttributeHandleFactory extends java.io.Serializable {
   public AttributeHandle decode(byte[] buffer, int offset)
    throws CouldNotDecode, FederateNotExecutionMember;
```



```
}
//end AttributeHandleFactory



//File: AttributeHandleSet.java

package hla.rti;

/**
All Set operations are required, none are optional.
add() and remove() should throw IllegalArgumentException if the argument is not
an AttributeHandle.
addAll(), removeAll() and retainAll() should throw IllegalArgumentException if
the argument is not an AttributeHandleSet
*/

public interface AttributeHandleSet
 extends java.util.Set, Cloneable, java.io.Serializable {
}

//end AttributeHandleSet



//File: AttributeHandleSetFactory.java
package hla.rti;

public interface AttributeHandleSetFactory extends java.io.Serializable {
 /**
  * return hla.rti.AttributeHandleSet newly created
  */
 public AttributeHandleSet create();
}
//end AttributeHandleSetFactory



//File: AttributeHandleValueMap.java
package hla.rti;

/**
Keys are AttributeHandles; values are byte[].
All operations are required, none optional.
Null mappings are not allowed.
put(), putAll(), and remove() should throw IllegalArgumentException to enforce
types of keys and mappings.
 */
public interface AttributeHandleValueMap
  extends java.util.Map, Cloneable, java.io.Serializable {
}
//end AttributeHandleValueMap



//File: AttributeHandleValueMapFactory.java
package hla.rti;
 /**
  * Factory for AttributeHandleValueMap instances.
  */
 public interface AttributeHandleValueMapFactory extends java.io.Serializable {
 /**
  * Creates a new AttributeHandleValueMap instance with specified initial
      capacity.
  */
 public AttributeHandleValueMap create ( int capacity);
}
//end AttributeHandleValueMapFactory
```

```
//File: AttributeSetRegionSetPairList.java

package hla.rti;

/**
 * This packages the attributes supplied to the RTI for various DDM services with
 * the regions to be used with the attributes.
 * Elements are AttributeRegionAssociations.
 * All operations are required, none optional.
 * add(), addAll(), and set() should throw IllegalArgumentException to enforce
 * type of elements.
 */
public interface AttributeSetRegionSetPairList
 extends java.util.List, Cloneable, java.io.Serializable {
}

//end AttributeSetRegionSetPairList



//File: AttributeSetRegionSetPairListFactory.java
package hla.rti;
 /**
  * Factory for AttributeHandleValuePairSet instances.
  */
 public interface AttributeSetRegionSetPairListFactory extends
   java.io.Serializable {

 /**
  * Creates a new AttributeHandleValuePairSet instance with specified initial
     capacity.
  */
 public AttributeSetRegionSetPairList create ( int capacity);
}
//end AttributeSetRegionSetPairListFactory



//File: DimensionHandle.java

package hla.rti;

/**
 * Type-safe handle for a dimension. Generally these are created by the
 * RTI and passed to the user.
 */

public interface DimensionHandle extends java.io.Serializable {

 /**
  * @return true if this refers to the same dimension as other handle
  */
 public boolean equals(Object otherDimensionHandle);

 /**
  * @return int. All instances that refer to the same dimension should return the
  * same hascode.
  */
public int hashCode();

 public int encodedLength();
 public void encode(byte[] buffer, int offset);
 public String toString();

}
//end DimensionHandle
```



```
//File: DimensionHandleFactory.java

package hla.rti;

 /**
  * The factory is used only (outside RTI) to create DimensionHandle
  * received as an attribute value or parameter value.
  */

public interface DimensionHandleFactory extends java.io.Serializable {
 public DimensionHandle decode(byte[] buffer, int offset)
  throws CouldNotDecode, FederateNotExecutionMember
}
//end DimensionHandleFactory



//File: DimensionHandleSet.java

package hla.rti;

/**
All Set operations are required, none are optional.
add() and remove() should throw IllegalArgumentException if the argument is not
a DimensionHandle.
addAll(), removeAll() and retainAll() should throw IllegalArgumentException if
the argument is not a DimensionHandleSet
*/

public interface DimensionHandleSet
 extends java.util.Set, Cloneable, java.io.Serializable {
}
//end DimensionHandleSet



//File: DimensionHandleSetFactory.java
package hla.rti;

public interface DimensionHandleSetFactory extends java.io.Serializable {

 /**
  * return hla.rti.DimensionHandleSet newly created
  */
 public DimensionHandleSet create();
}
//end DimensionHandleSetFactory



//File: FederateHandle.java

package hla.rti;
/**
 * Type-safe handle for a federate handle. Generally these are created by the
 * RTI and passed to the user.
 */

public interface FederateHandle extends java.io.Serializable {

 /**
  * @return true if this refers to the same federate as other handle
  */

public boolean equals(Object otherFederateHandle);

 /**
  * @return int. All instances that refer to the same federate should return the
  * same hashcode.
```

```
   */
public int hashCode();

 public int encodedLength();
 public void encode(byte[] buffer, int offset);

 public String toString();

}
//end FederateHandle



//File: FederateHandleFactory.java

package hla.rti;

 /**
  * The factory is used only (outside RTI) to create FederateHandle
  * received as an attribute value or parameter value.
  */

public interface FederateHandleFactory extends java.io.Serializable {
 public FederateHandle decode(byte[] buffer, int offset)
  throws CouldNotDecode, FederateNotExecutionMember;
}
//end FederateHandleFactory



//File: FederateHandleRestoreStatusPair.java

/**
 * Array of these records returned by (4.25) federationRestoreStatusResponse
 */

package hla.rti;

public final class FederateHandleRestoreStatusPair {
 public FederateHandle handle;
 public RestoreStatus  status;
}
//end FederateHandleRestoreStatusPair



//File: FederateHandleSaveStatusPair.java

/**
 * Array of these records returned by (4.17) federationSaveStatusResponse
 */

package hla.rti;

public final class FederateHandleSaveStatusPair {
 public FederateHandle handle;
 public SaveStatus     status;
}
//end FederateHandleSaveStatusPair
```



```
//File: FederateHandleSet.java

package hla.rti;

/**
All Set operations are required, none are optional.
add() and remove() should throw IllegalArgumentException if the argument is not
a FederateHandleHandle.
addAll(), removeAll() and retainAll() should throw IllegalArgumentException if
the argument is not a FederateHandleSet
*/

public interface FederateHandleSet
 extends java.util.Set, java.io.Serializable, Cloneable {
}
//end FederateHandleSet



//File: FederateHandleSetFactory.java
package hla.rti;

public interface FederateHandleSetFactory extends java.io.Serializable {

 /**
  * return hla.rti.FederateHandleSet newly created
  */
 public FederateHandleSet create();
}
//end FederateHandleSetFactory



//File: InteractionClassHandle.java

package hla.rti;

/**
 * Type-safe handle for an interaction class. Generally these are created by the
 * RTI and passed to the user.
 */

public interface InteractionClassHandle extends java.io.Serializable {

 /**
  * @return true if this refers to the same interaction class as other handle
  */
 public boolean equals(Object otherInteractionClassHandle);

 /**
  * @return int. All instances that refer to the same interaction class should
  * return the same hashcode.
  */
public int hashCode();

 public int encodedLength();
 public void encode(byte[] buffer, int offset);

 public String toString();

}
//end InteractionClassHandle



//File: InteractionClassHandleFactory.java

package hla.rti;
```

```
/**
 * The factory is used only (outside RTI) to create InteractionClassHandle
 * received as an attribute value or parameter value.
 */

public interface InteractionClassHandleFactory extends java.io.Serializable {
 public InteractionClassHandle decode(byte[] buffer, int offset)
  throws CouldNotDecode, FederateNotExecutionMember;
}
//end InteractionClassHandleFactory



//File: LogicalTime.java

package hla.rti;
/**
 * LogicalTime declares an interface to an immutable time value
 */

public interface LogicalTime extends Comparable, java.io.Serializable
{
 public boolean isInitial();
 public boolean isFinal();

 /**
  * Returns a LogicalTime whose value is (this + val).
  */
 public LogicalTime add(LogicalTimeInterval val)
 throws IllegalTimeArithmetic;

 /**
  * Returns a LogicalTime whose value is (this - val).
  */
 public LogicalTime subtract(LogicalTimeInterval val)
 throws IllegalTimeArithmetic;

 /**
  * Returns a LogicalTimeInterval whose value is the time
  * interval between this and val.
  */
 public LogicalTimeInterval distance(LogicalTime val);

 public int compareTo(Object other);

 /**
  * Returns true iff this and other represent the same logical time
  * Supports standard Java mechanisms.
  */
 public boolean equals(Object other);

 /**
  * Two LogicalTimes for which equals() is true should yield
  * same hash code
  */
 public int hashCode();

 public String toString();
 public int encodedLength();
 public void encode(byte[] buffer, int offset);

}//end LogicalTime



// File: LogicalTimeFactory.java
package hla.rti;

public interface LogicalTimeFactory extends java.io.Serializable {
```



```
 public LogicalTime decode(byte[] buffer, int offset)
  throws CouldNotDecode;
 public LogicalTime makeInitial();
 public LogicalTime makeFinal();
}



//File: LogicalTimeInterval.java
package hla.rti;

/**
 * LogicalTimeInterval declares an interface to an immutable time interval value
 */

public interface LogicalTimeInterval extends Comparable, java.io.Serializable
{
 public boolean isZero();
 public boolean isEpsilon();

 /**
   * Returns a LogicalTimeInterval whose value is (this - subtrahend).
   */
 public LogicalTimeInterval subtract(LogicalTimeInterval subtrahend);

 public int compareTo(Object other);

 /**
   * Returns true iff this and other represent the same time interval.
   */
 public boolean equals(Object other);

 /**
   * Two LogicalTimeIntervals for which equals() is true should yield
   * same hash code
   */
 public int hashCode();

 public String toString();
 public int encodedLength();
 public void encode(byte[] buffer, int offset);
}
//end LogicalTimeInterval



// File: LogicalTimeIntervalFactory.java
package hla.rti;

public interface LogicalTimeIntervalFactory extends java.io.Serializable {

 public LogicalTimeInterval decode(byte[] buffer, int offset)
  throws CouldNotDecode;
 public LogicalTimeInterval makeZero();
 public LogicalTimeInterval makeEpsilon();
}



//File: MessageRetractionHandle.java

package hla.rti;

/**
 * The user can do nothing with these but employ them as keys.
  * Implementers should provide equals, hashCode and toString
  * rather than settling for the defaults.
  *
  */
```

```
public interface MessageRetractionHandle extends java.io.Serializable {

 /**
  * @return true if this refers to the same Message as other handle
  */
 public boolean equals(Object otherMRHandle);

 /**
  * @return int. All instances that refer to the same Message should return the
  * same hashcode.
  */
 public int hashCode();

 public String toString();
}
//end MessageRetractionHandle



//File: MessageRetractionReturn.java

/**
 * Record returned by updateAttributeValues, sendInteraction, and deleteObject
 */

package hla.rti;

public final class MessageRetractionReturn {
 public boolean    retractionHandleIsValid;
 public MessageRetractionHandle handle;
}
//end MessageRetractionReturn



//File: MobileFederateServices.java

package hla.rti;

/**
  * Conveys the interfaces for all services that a federate
  * must supply and which may not execute in the federate's
  * space.
  *
  */
public final class MobileFederateServices {
 public hla.rti.LogicalTimeFactory _timeFactory;
 public hla.rti.LogicalTimeIntervalFactory _intervalFactory;

/**
  * @param timeFactory hla.rti.LogicalTimeFactory
  * @param intervalFactory hla.rti.LogicalTimeIntervalFactory
  */
public MobileFederateServices (
 LogicalTimeFactory timeFactory,
 LogicalTimeIntervalFactory intervalFactory)
{
   _timeFactory = timeFactory;
   _intervalFactory = intervalFactory;
}
}

//end MobileFederateServices
```



```
//File: ObjectClassHandle.java

package hla.rti;

/**
  * Type-safe handle for an object class. Generally these are created by the
  * RTI and passed to the user.
  */

public interface ObjectClassHandle extends java.io.Serializable {

 /**
  * @return true if this refers to the same object class as other handle
  */
 public boolean equals(Object otherObjectClassHandle);

 /**
  * @return int. All instances that refer to the same object class should return the
  * same hashcode.
  */

 public int hashCode();


  public int encodedLength();
  public void encode(byte[] buffer, int offset);

 public String toString();

}
//end ObjectClassHandle



//File: ObjectClassHandleFactory.java

package hla.rti;

/**
 * The factory is used only (outside RTI) to create ObjectClassHandle
 * received as an attribute value or parameter value.
 */

public interface ObjectClassHandleFactory extends java.io.Serializable {
 public ObjectClassHandle decode(byte[] buffer, int offset)
  throws CouldNotDecode, FederateNotExecutionMember;
}
//end ObjectClassHandleFactory



//File: ObjectInstanceHandle.java

package hla.rti;

/**
  * Type-safe handle for an object instance. Generally these are created by the
  * RTI and passed to the user.
  */

public interface ObjectInstanceHandle extends java.io.Serializable {

 /**
  * @return true if this refers to the same instance as other handle
  */
 public boolean equals(Object otherObjectInstanceHandle);

 /**
  * @return int. All instances that refer to the same instance should return the
  * same hascode.
```

```
   */
  public int hashCode();

  public int encodedLength();
  public void encode(byte[] buffer, int offset);

  public String toString();

}
//end ObjectInstanceHandle



//File: ObjectInstanceHandleFactory.java

package hla.rti;

/**
   * The factory is used only (outside RTI) to create ObjectInstanceHandle
   * received as an attribute value or parameter value.
   */

public interface ObjectInstanceHandleFactory extends java.io.Serializable {
  public ObjectInstanceHandle decode(byte[] buffer, int offset)
    throws CouldNotDecode, FederateNotExecutionMember;
}
//end ObjectInstanceHandleFactory



// File: OrderType.java
package hla.rti;

public final class OrderType implements java.io.Serializable {
  private int _value; //each instance's value
  private static final int _lowestValue = 1;
  private static int _nextToAssign = _lowestValue; //begins at lowest

  /**
  This is the only public constructor. Each user-defined instance of a OrderType
  must be initialized with one of the defined static values.
  * @param otherOrderTypeValue must be a defined static value or another instance.
  */
  public OrderType(OrderType otherOrderTypeValue) {
    _value = otherOrderTypeValue._value;
  }

  /**
  Private to class
  */
  private OrderType() {
    _value = _nextToAssign++;
  }

  OrderType(int value)
  throws RTIinternalError
  {
    _value = value;
   if (value < _lowestValue || value >= _nextToAssign) throw new
     RTIinternalError("OrderType: illegal value " + value);
  }

  /**
  * @return String with value "OrderType(n)" where n is value
  */
  public String toString() {
   return "OrderType(" + _value + ")";
  }

  /**
```



```
 Allows comparison with other instance of same type.
 * @return true if supplied object is of type OrderType and has same value;
 false otherwise
 */
 public boolean equals(Object otherOrderTypeValue) {
  if (otherOrderTypeValue instanceof OrderType)
   return _value == ((OrderType)otherOrderTypeValue)._value;
  else return false;
 }

 public int hashCode() {
  return _value;
 }

 public int encodedLength() {
  return1;
 }

 public void encode(byte[] buffer, int offset) {
  buffer[offset] = (byte)_value;
 }

 public static OrderType decode(byte[] buffer, int offset)
  throws CouldNotDecode
 {
  int val = buffer[offset];
  OrderType neo;
  try {
   neo = new OrderType(val);
   }
  catch (RTIinternalError e) {
   throw new CouldNotDecode(e.getMessage());
   }
  return neo;
  }

  static public final OrderType RECEIVE
   = new OrderType();
  static public final OrderType TIMESTAMP
   = new OrderType();
 }



//File: ParameterHandle.java

package hla.rti;

/**
 * Type-safe handle for a parameter. Generally these are created by the
 * RTI and passed to the user.
 */

public interface ParameterHandle extends java.io.Serializable {

 /**
  * @return true if this refers to the same parameter as other handle
  */
 public boolean equals(Object otherParameterHandle);

 /**
  * @return int. All instances that refer to the same parameter should return the
  * same hascode.
  */
 public int hashCode();

  public int encodedLength();
  public void encode(byte[] buffer, int offset);

  public String toString();
```

```
}
//end ParameterHandle



//File: ParameterHandleFactory.java

package hla.rti;

/**
 * The factory is used only (outside RTI) to create ParameterHandle
 * received as an attribute value or parameter value.
 */

public interface ParameterHandleFactory extends java.io.Serializable {
  public ParameterHandle decode(byte[] buffer, int offset)
   throws CouldNotDecode, FederateNotExecutionMember;
}
//end ParameterHandleFactory



//File: ParameterHandleValueMap.java
package hla.rti;

/**
Keys are ParameterHandles; values are byte[].
All operations are required, none optional.
Null mappings are not allowed.
put(), putAll(), and remove() should throw IllegalArgumentException to enforce
types of keys and mappings.
 */
public interface ParameterHandleValueMap
 extends java.util.Map, Cloneable, java.io.Serializable {
}
//end ParameterHandleValueMap



//File: ParameterHandleValueMapFactory.java
package hla.rti;

/**
 * Factory for ParameterHandleValueMap instances.
 */
public interface ParameterHandleValueMapFactory extends java.io.Serializable {

 /**
  * Createsa new ParameterHandleValueMap instance with specified initial
     capacity.
  */
 public ParameterHandleValueMap create ( int capacity);
}
//end ParameterHandleValueMapFactory



//File: RangeBounds.java

/**
 * Record returned by (10.31) getRangeBounds
 */

package hla.rti;

public final class RangeBounds {

 public long lower;
```



```
 public long upper;
}
//end RangeBounds



//File: RegionHandle.java

package hla.rti;

public interface RegionHandle extends java.io.Serializable {

 /**
  * @return true if this refers to the same Region as other handle
  */
 public boolean equals(Object otherRegionHandle);

 /**
  * @return int. All instances that refer to the same Region should return the
  * same hashcode.
  */
 public int hashCode();

 public String toString();

}
//end RegionHandle



//File: RegionHandleSet.java

package hla.rti;

/**
All Set operations are required, none are optional.
add() and remove() should throw IllegalArgumentException if the argument is not
a RegionHandle.
addAll(), removeAll() and retainAll() should throw IllegalArgumentException if
the argument is not a RegionHandleSet
*/

public interface RegionHandleSet
 extends java.util.Set, Cloneable, java.io.Serializable {
}
//end RegionHandleSet



//File: RegionHandleSetFactory.java
package hla.rti;

public interface RegionHandleSetFactory extends java.io.Serializable {

 /**
  * return hla.rti.RegionHandleSet newly created
  */
 public RegionHandleSet create();
}
//end RegionHandleSetFactory
```

```
// File: ResignAction.java

package hla.rti;

/**
  An enumerated type (not a Java Enumeration!)
  * @see hla.rti.RTIambassador#resignFederationExecution
*/

public final class ResignAction implements java.io.Serializable {
 private int _value; //each instance's value
 private static final int _lowestValue = 1;
 private static int _nextToAssign = _lowestValue; //begins at lowest
 /**
 This is the only public constructor. Each user-defined instance of a ResignAction
 must be initialized with one of the defined static values.
 * @param otherResignActionValue must be a defined static value or another
      instance.
 */
 public ResignAction(ResignAction otherResignActionValue) {
  _value = otherResignActionValue._value;
 }

 /**
 Private to class
 */
 private ResignAction() {
  _value = _nextToAssign++;
 }

 ResignAction(int value)
 throws RTIinternalError
 {
  _value = value;
  if (value < _lowestValue || value >= _nextToAssign) throw new
   RTIinternalError("ResignAction: illegal value " + value);
 }

 /**
 * @return String with value "ResignAction(n)" where n is value
 */
 public String toString() {
  return "ResignAction(" + _value + ")";
 }

 /**
 Allows comparison with other instance of same type.
 * @return true if supplied object is of type ResignAction and has same value;
 false otherwise
 */
 public boolean equals(Object otherResignActionValue) {
  if (otherResignActionValue instanceof ResignAction)
   return _value == ((ResignAction)otherResignActionValue)._value;
  else return false;
 }

 public int hashCode() {
  return _value;
 }

 static public final ResignAction UNCONDITIONALLY_DIVEST_ATTRIBUTES
  = new ResignAction();
 static public final ResignAction DELETE_OBJECTS
  = new ResignAction();
 static public final ResignAction CANCEL_PENDING_OWNERSHIP_ACQUISITIONS
  = new ResignAction();
 static public final ResignAction DELETE_OBJECTS_THEN_DIVEST
  = new ResignAction();
 static public final ResignAction CANCEL_THEN_DELETE_THEN_DIVEST
  = new ResignAction();
 static public final ResignAction NO_ACTION
  = new ResignAction();
}
```



```
// File: RestoreFailureReason.java

package hla.rti;

/**
  An enumerated type (not a Java Enumeration!)
  * @see hla.rti.FederateAmbassador#federationNotResotred
*/

public final class RestoreFailureReason implements java.io.Serializable {
 private int _value; //each instance's value
 private static final int _lowestValue = 1;
 private static int _nextToAssign = _lowestValue; //begins at lowest

 /**
 This is the only public constructor. Each user-defined instance of a
 RestoreFailureReason
 must be initialized with one of the defined static values.
 * @param otherResignActionValue must be a defined static value or another
     instance.
 */
 public RestoreFailureReason(
  RestoreFailureReason otherResignActionValue) {
  _value = otherResignActionValue._value;
 }

 /**
 Private to class
 */
 private RestoreFailureReason() {
  _value = _nextToAssign++;
 }

 RestoreFailureReason(int value)
 throws RTIinternalError
 {
  _value = value;
  if (value < _lowestValue || value >= _nextToAssign) throw new
   RTIinternalError("RestoreFailureReason: illegal value " + value);
 }

 /**
 * @return String with value "RestoreFailureReason(n)"
 * where n is value
 */
 public String toString() {
  return "RestoreFailureReason(" + _value + ")";
 }

 /**
 Allows comparison with other instance of same type.
 * @return true if supplied object is of type RestoreFailureReason
 * and has same value; false otherwise
 */
 public boolean equals(Object otherResignActionValue) {
  if (otherResignActionValue instanceof RestoreFailureReason)
   return _value ==
    ((RestoreFailureReason)otherResignActionValue)._value;
   else return false;
 }

 public int hashCode() {
  return _value;
 }
```

```
 static public final RestoreFailureReason
   RTI_UNABLE_TO_RESTORE = new RestoreFailureReason();
 static public final RestoreFailureReason
   FEDERATE_REPORTED_FAILURE = new RestoreFailureReason();
 static public final RestoreFailureReason
   FEDERATE_RESIGNED = new RestoreFailureReason();
 static public final RestoreFailureReason
   RTI_DETECTED_FAILURE = new RestoreFailureReason();
}



// File: RestoreStatus.java

package hla.rti;

/**
 An enumerated type (not a Java Enumeration!)
*/

public final class RestoreStatus implements java.io.Serializable {
 private int _value; //each instance's value
 private static final int _lowestValue = 1;
 private static int _nextToAssign = _lowestValue; //begins at lowest

 /**
 This is the only public constructor. Each user-defined instance of a RestoreStatus
 must be initialized with one of the defined static values.
 * @param otherRestoreStatusValue must be a defined static value or another
instance.
 */
 public RestoreStatus(RestoreStatus otherRestoreStatusValue) {
   _value = otherRestoreStatusValue._value;
 }

 /**
 Private to class
 */
 private RestoreStatus() {
   _value = _nextToAssign++;
 }

 RestoreStatus(int value)
 throws RTIinternalError
 {
   _value = value;
  if (value < _lowestValue || value >= _nextToAssign) throw new
    RTIinternalError("RestoreStatus: illegal value " + value);
 }

 /**
 * @return String with value "RestoreStatus(n)" where n is value
 */
 public String toString() {
  return "RestoreStatus(" + _value + ")";
 }

 /**
 Allows comparison with other instance of same type.
 * @return true if supplied object is of type RestoreStatus and has same value;
 false otherwise
 */
 public boolean equals(Object otherRestoreStatusValue) {
  if (otherRestoreStatusValue instanceof RestoreStatus)
    return _value == ((RestoreStatus)otherRestoreStatusValue)._value;
  else return false;
 }

 public int hashCode() {
  return _value;
 }
```



```
 static public final RestoreStatus NO_RESTORE_IN_PROGRESS
  = new RestoreStatus();
 static public final RestoreStatus FEDERATE_RESTORE_REQUEST_PENDING
  = new RestoreStatus();
 static public final RestoreStatus FEDERATE_WAITING_FOR_RESTORE_TO_BEGIN
  = new RestoreStatus();
 static public final RestoreStatus FEDERATE_PREPARED_TO_RESTORE
  = new RestoreStatus();
 static public final RestoreStatus FEDERATE_RESTORING
  = new RestoreStatus();
 static public final RestoreStatus FEDERATE_WAITING_FOR_FEDERATION_TO_RESTORE
  = new RestoreStatus();
}
```

```
//File: RTIexception.java
package hla.rti;

/**
 * Superclass of all exceptions thrown by the RTI.
 * All RTI exceptions must be caught or specified.
*/
public class RTIexception extends Exception {
 public RTIexception(String msg) {
  super(msg);
 }
}
//end RTIexception
```

```
// File: SaveFailureReason.java

package hla.rti;

/**
 An enumerated type (not a Java Enumeration!)
 * @see hla.rti.FederateAmbassador#federationNotSaved
*/

public final class SaveFailureReason implements java.io.Serializable {
 private int _value; //each instance's value
 private static final int _lowestValue = 1;
 private static int _nextToAssign = _lowestValue; //begins at lowest

 /**
 This is the only public constructor. Each user-defined instance of a
     SaveFailureReason
 must be initialized with one of the defined static values.
 * @param otherSaveFailureReasonValue must be a defined static value or another
    instance.
 */
 public SaveFailureReason(SaveFailureReason otherSaveFailureReasonValue) {
  _value = otherSaveFailureReasonValue._value;
 }

 /**
 Private to class
 */
 private SaveFailureReason() {
  _value = _nextToAssign++;
 }

 SaveFailureReason(int value)
 throws RTIinternalError
 {
  _value = value;
  if (value < _lowestValue || value >= _nextToAssign) throw new
   RTIinternalError("SaveFailureReason: illegal value " + value);
```

```
  }

  /**
  * @return String with value "SaveFailureReason(n)" where n is value
  */
  public String toString() {
   return "SaveFailureReason(" + _value + ")";
  }

  /**
  Allows comparison with other instance of same type.
  * @return true if supplied object is of type SaveFailureReason and has same
value;
  false otherwise
  */
  public boolean equals(Object otherSaveFailureReasonValue) {
   if (otherSaveFailureReasonValue instanceof SaveFailureReason)
    return _value == ((SaveFailureReason)otherSaveFailureReasonValue)._value;
   else return false;
  }
  public int hashCode() {
   return _value;
}

  static public final SaveFailureReason RTI_UNABLE_TO_SAVE
     = new SaveFailureReason();
  static public final SaveFailureReason FEDERATE_REPORTED_FAILURE
     = new SaveFailureReason();
  static public final SaveFailureReason FEDERATE_RESIGNED
     = new SaveFailureReason();
  static public final SaveFailureReason RTI_DETECTED_FAILURE
     = new SaveFailureReason();
  static public final SaveFailureReason SAVE_TIME_CANNOT_BE_HONORED
     = new SaveFailureReason();
}



// File: SaveStatus.java
package hla.rti;

/**
 An enumerated type (not a Java Enumeration!)
*/

public final class SaveStatus implements java.io.Serializable {
 private int _value; //each instance's value
 private static final int _lowestValue = 1;
 private static int _nextToAssign = _lowestValue; //begins at lowest

 /**
 This is the only public constructor. Each user-defined instance of a SaveStatus
 must be initialized with one of the defined static values.
 * @param otherSaveStatusValue must be a defined static value or another instance.
 */
 public SaveStatus(SaveStatus otherSaveStatusValue) {
   _value = otherSaveStatusValue._value;
 }

 /**
 Private to class
 */
 private SaveStatus() {
   _value = _nextToAssign++;
 }
 SaveStatus(int value)
 throws RTIinternalError
 {
   _value = value;
  if (value < _lowestValue || value >= _nextToAssign) throw new
    RTIinternalError("SaveStatus: illegal value " + value);
```



```
 }

 /**
 * @return String with value "SaveStatus(n)" where n is value
 */
 public String toString() {
  return "SaveStatus(" + _value + ")";
 }

 /**
 Allows comparison with other instance of same type.
 * @return true if supplied object is of type SaveStatus and has same value;
 false otherwise
 */
 public boolean equals(Object otherSaveStatusValue) {
  if (otherSaveStatusValue instanceof SaveStatus)
   return _value == ((SaveStatus)otherSaveStatusValue)._value;
  else return false;
 }

 public int hashCode() {
  return _value;
 }

 static public final SaveStatus NO_SAVE_IN_PROGRESS
  = new SaveStatus();
 static public final SaveStatus FEDERATE_INSTRUCTED_TO_SAVE
  = new SaveStatus();
 static public final SaveStatus FEDERATE_SAVING
  = new SaveStatus();
 static public final SaveStatus FEDERATE_WAITING_FOR_FEDERATION_TO_SAVE
  = new SaveStatus();
}



// File: ServiceGroup.java

package hla.rti;

/**
 An enumerated type (not a Java Enumeration!)
 * @see hla.rti.RTIambassador#normalizeServiceGroup
*/

public final class ServiceGroup implements java.io.Serializable {
 private int _value; //each instance's value
 private static final int _lowestValue = 4; //fedn mgt is chapter 4
 private static int _nextToAssign = _lowestValue; //begins at lowest

 /**
 This is the only public constructor. Each user-defined instance of a ServiceGroup
 must be initialized with one of the defined static values.
 * @param otherServiceGroupValue must be a defined static value or another
    instance.
 */
 public ServiceGroup(ServiceGroup otherServiceGroupValue) {
  _value = otherServiceGroupValue._value;
 }

 /**
 Private to class*/
 private ServiceGroup() {
  _value = _nextToAssign++;
 }

 ServiceGroup(int value)
 throws RTIinternalError
 {
  _value = value;
  if (value < _lowestValue || value >= _nextToAssign) throw new
```

```
    RTIinternalError("ServiceGroup: illegal value " + value);
  }

  /**
  * @return String with value "ServiceGroup(n)" where n is value
  */
  public String toString() {
   return "ServiceGroup(" + _value + ")";
  }

  /**
  Allows comparison with other instance of same type.
  * @return true if supplied object is of type ServiceGroup and has same value;
  false otherwise
  */
  public boolean equals(Object otherServiceGroupValue) {
   if (otherServiceGroupValue instanceof ServiceGroup)
    return _value == ((ServiceGroup)otherServiceGroupValue)._value;
   else return false;
  }

  public int hashCode() {
   return _value;
  }

  static public final ServiceGroup FEDERATION_MANAGEMENT
   = new ServiceGroup();
  static public final ServiceGroup DECLARATION_MANAGEMENT
   = new ServiceGroup();
  static public final ServiceGroup OBJECT_MANAGEMENT
   = new ServiceGroup();
  static public final ServiceGroup OWNERSHIP_MANAGEMENT
   = new ServiceGroup();
  static public final ServiceGroup TIME_MANAGEMENT
   = new ServiceGroup();
  static public final ServiceGroup DATA_DISTRIBUTION_MANAGEMENT
   = new ServiceGroup();
  static public final ServiceGroup SUPPPORT_SERVICES
   = new ServiceGroup();
  }



// File: SynchronizationPointFailureReason.java

package hla.rti;

/**
 An enumerated type (not a Java Enumeration!)
 * @see hla.rti.FederateAmbassador#synchronizationPointRegistrationFailed
*/

public final class SynchronizationPointFailureReason implements
    java.io.Serializable {
  private int _value; //each instance's value
  private static final int _lowestValue = 1;
  private static int _nextToAssign = _lowestValue; //begins at lowest

  /**
  This is the only public constructor. Each user-defined instance of a
  SynchronizationPointFailureReason
  must be initialized with one of the defined static values.
  * @param otherResignActionValue must be a defined static value or another
     instance.
  */
  public SynchronizationPointFailureReason(
   SynchronizationPointFailureReason otherResignActionValue) {
    _value = otherResignActionValue._value;
  }

  /**
```



```
 Private to class
 */
 private SynchronizationPointFailureReason() {
  _value = _nextToAssign++;
 }

 SynchronizationPointFailureReason(int value)
 throws RTIinternalError
 {
  _value = value;
  if (value < _lowestValue || value >= _nextToAssign) throw new
   RTIinternalError("SynchronizationPointFailureReason: illegal value " +
    value);
 }

 /**
 * @return String with value "SynchronizationPointFailureReason(n)"
 * where n is value
 */
 public String toString() {
  return "SynchronizationPointFailureReason(" + _value + ")";
 }

 /**
 Allows comparison with other instance of same type.
 * @return true if supplied object is of type SynchronizationPointFailureReason
 * and has same value; false otherwise
 */
 public boolean equals(Object otherResignActionValue) {
  if (otherResignActionValue instanceof SynchronizationPointFailureReason)
   return _value ==
    ((SynchronizationPointFailureReason)otherResignActionValue)._value;
  else return false;
 }

 public int hashCode() {
  return _value;
 }

 static public final SynchronizationPointFailureReason
  SYNCHRONIZATION_POINT_LABEL_NOT_UNIQUE
  = new SynchronizationPointFailureReason();
 static public final SynchronizationPointFailureReason
  SYNCHRONIZATION_SET_MEMBER_NOT_JOINED
  = new SynchronizationPointFailureReason();
}



//File: TimeQueryReturn.java

/**
 * Record returned by (8.16) queryLBTS and (8.18) queryMinimumNextEventTime
 */

package hla.rti;

public final class TimeQueryReturn {
 public boolean timeIsValid;
 public LogicalTime time;
}
//end TimeQueryReturn



// File: TransportationType.java

package hla.rti;

public class TransportationType implements java.io.Serializable {
```

```
 protected int _value; //each instance's value
 private static final int _lowestValue = 1;
 protected static int _nextToAssign = _lowestValue; //begins at lowest

 /**
 This is the only public constructor. Each user-defined instance of a
   TransportationType
 must be initialized with one of the defined static values.
 * @param otherTransportationTypeValue must be a defined static value or another
    instance.
 */
 public TransportationType(TransportationType otherTransportationTypeValue) {
  _value = otherTransportationTypeValue._value;
 }

 /**
 Private to class and subclasses
 */
 protected TransportationType() {
  _value = _nextToAssign++;
 }

 TransportationType(int value)
 throws RTIinternalError
 {
  _value = value;
  if (value < _lowestValue || value >= _nextToAssign) throw new
   RTIinternalError("TransportationType: illegal value " + value);
 }

 /**
 * @return String with value "TransportationType(n)" where n is value
 */
 public String toString() {
  return "TransportationType(" + _value + ")";
 }

 /**
 Allows comparison with other instance of same type.
 * @return true if supplied object is of type TransportationType and has same
value;
 false otherwise
 */
 public boolean equals(Object otherTransportationTypeValue) {
  if (otherTransportationTypeValue instanceof TransportationType)
   return _value == ((TransportationType)otherTransportationTypeValue)._value;
  else return false;
 }

 public int hashCode() {
  return _value;
 }

 public int encodedLength() {
  return1;
 }

 public void encode(byte[] buffer, int offset) {
  buffer[offset] = (byte)_value;
 }

 public static TransportationType decode(byte[] buffer, int offset)
  throws CouldNotDecode
 {
  int val = buffer[offset];
  TransportationType neo;
  try {
   neo = new TransportationType(val);
  }
  catch (RTIinternalError e) {
   throw new CouldNotDecode(e.getMessage());
  }
```



```
  return neo;
 }
 static public final TransportationType HLA_RELIABLE
  = new TransportationType();
 static public final TransportationType HLA_BEST_EFFORT
  = new TransportationType();
}
```

B.2 Notes

The **`getHLAversion`** method shall return the string "1516.1.5."

There are a great many exceptions mentioned in the ambassador interfaces. All mentioned exceptions extend **`RTIexception`**. For the sake of brevity, only the implementations of **`RTIexception`** and AsynchronousDeliveryAlreadyDisabled.java are included here. The implementations of all other RTI exceptions are analogous. The other exceptions are:

AsynchronousDeliveryAlreadyEnabled.java
AttributeAcquisitionWasNotCanceled.java
AttributeAcquisitionWasNotRequested.java
AttributeAlreadyBeingAcquired.java
AttributeAlreadyBeingDivested.java
AttributeAlreadyOwned.java
AttributeDivestitureWasNotRequested.java
AttributeNotDefined.java
AttributeNotOwned.java
AttributeNotPublished.java
AttributeNotRecognized.java
AttributeNotSubscribed.java
AttributeRelevanceAdvisorySwitchIsOff.java
AttributeRelevanceAdvisorySwitchIsOn.java
AttributeScopeAdvisorySwitchIsOff.java
AttributeScopeAdvisorySwitchIsOn.java
BadInitializationParameter.java
CouldNotDecode.java
CouldNotDiscover.java
CouldNotInitiateRestore.java
CouldNotOpenFDD.java
DeletePrivilegeNotHeld.java
ErrorReadingFDD.java
FederateAlreadyExecutionMember.java
FederateHasNotBegunSave.java
FederateInternalError.java
FederateNotExecutionMember.java
FederateOwnsAttributes.java
FederatesCurrentlyJoined.java
FederateServiceInvocationsAreBeingReportedViaMOM.java
FederateUnableToUseTime.java
FederationExecutionAlreadyExists.java
FederationExecutionDoesNotExist.java
IllegalName.java
IllegalTimeArithmetic.java
InitializeNeverInvoked.java
InitializePreviouslyInvoked.java
InteractionClassNotDefined.java

InteractionClassNotPublished.java
InteractionClassNotRecognized.java
InteractionClassNotSubscribed.java
InteractionParameterNotDefined.java
InteractionParameterNotRecognized.java
InteractionRelevanceAdvisorySwitchIsOff.java
InteractionRelevanceAdvisorySwitchIsOn.java
InTimeAdvancingState.java
InvalidAttributeHandle.java
InvalidDimensionHandle.java
InvalidFederateHandle.java
InvalidInteractionClassHandle.java
InvalidLogicalTime.java
InvalidLookahead.java
InvalidMessageRetractionHandle.java
InvalidObjectClassHandle.java
InvalidOrderName.java
InvalidOrderType.java
InvalidParameterHandle.java
InvalidRangeBound.java
InvalidRegion.java
InvalidRegionContext.java
InvalidTransportationName.java
InvalidTransportationType.java
JoinedFederateIsNotInTimeAdvancingState.java
LogicalTimeAlreadyPassed.java
MessageCanNoLongerBeRetracted.java
NameNotFound.java
NoRequestToEnableTimeConstrainedWasPending.java
NoRequestToEnableTimeRegulationWasPending.java
ObjectClassNotDefined.java
ObjectClassNotPublished.java
ObjectClassNotRecognized.java
ObjectClassRelevanceAdvisorySwitchIsOff.java
ObjectClassRelevanceAdvisorySwitchIsOn.java
ObjectInstanceNameInUse.java
ObjectInstanceNameNotReserved.java
ObjectInstanceNotKnown.java
OwnershipAcquisitionPending.java
RegionDoesNotContainSpecifiedDimension.java
RegionInUseForUpdateOrSubscription.java
RegionNotCreatedByThisFederate.java
RegionNotKnown.java
RequestForTimeConstrainedPending.java
RequestForTimeRegulationPending.java
RestoreInProgress.java
RestoreNotInProgress.java
RestoreNotRequested.java
RTIinternalError.java
SaveInProgress.java
SaveNotInitiated.java
SaveNotInProgress.java
SomeFederateJoinedToAnExecution.java
SpecifiedSaveLabelDoesNotExist.java

SynchronizationPointLabelNotAnnounced.java
SynchronizationSetMemberNotJoined.java
TimeConstrainedAlreadyEnabled.java
TimeConstrainedIsNotEnabled.java
TimeRegulationAlreadyEnabled.java
TimeRegulationIsNotEnabled.java
UnableToPerformSave.java
UnknownName.java

B.3 Examples

This subclause is informative. It provides example files for the following implementations:

— Integer 64-based implementations of the HLA time ADTs

— Sample of an RTI implementation provided transpiration type

```
// File: ExampleLogicalTime.java

package com.some_rti_supplier;

import hla.rti.*;
import java.io.*;
/**
 * ExampleLogicalTime implements the interface LogicalTime.
 */

public class ExampleLogicalTime implements LogicalTime, java.io.Serializable
{
  private final long _value;
  private static final long _initial = 0;
  private static final long _final = Long.MAX_VALUE;
 private transient byte[] _serialization;

 static final ExampleLogicalTime INITIAL = new ExampleLogicalTime(_initial);
 static final ExampleLogicalTime FINAL = new ExampleLogicalTime(_final);

 ExampleLogicalTime(long value)
 {
  _value = value;
 }

 public boolean isInitial()
 {
  return _value == _initial;
 }

 public boolean isFinal()
 {
  return _value == _final;
 }

 /**
  * Returns a LogicalTime whose value is (this + val).
  */
 public LogicalTime add(LogicalTimeInterval val)
 throws IllegalTimeArithmetic
 {
  return
   new ExampleLogicalTime(_value + ((ExampleLogicalTimeInterval)val)._value);
 }
```

```
 /**
  * Returns a LogicalTime whose value is (this - val).
  */
 public LogicalTime subtract(LogicalTimeInterval val)
 throws IllegalTimeArithmetic
 {
  return
   new ExampleLogicalTime(_value - ((ExampleLogicalTimeInterval)val)._value);
 }

 /**
  * Returns a LogicalTimeInterval whose value is the time
  * interval between this and val.
  */
 public LogicalTimeInterval distance(LogicalTime val)
 {
  return new ExampleLogicalTimeInterval(
   Math.abs(_value - ((ExampleLogicalTime)val)._value));
 }

 public int compareTo(Object other)
 {
  long otherValue = ((ExampleLogicalTime)other)._value;
  if (_value == otherValue) {
   return 0;
  } else if (_value < otherValue) {
   return -1;
  } else {
   return 1;
  }
 }

 /**
  * Returns true iff this and other represent the same time interval.
  */
 public boolean equals(Object other)
 {
  if (other instanceof ExampleLogicalTime) {
   return equals((ExampleLogicalTime)other);
  } else {
   return false;
  }
 }

 public boolean equals(ExampleLogicalTime other)
 {
  return _value == other._value;
 }

 /**
  * Two LogicalTimes for which equals() is true should yield
  * same hash code.
  */
 public int hashCode()
 {
  return (new Long(_value)).hashCode();
 }

 public String toString()
 {
  return "ExampleLT<" + _value + ">";
 }

 public int encodedLength()
 {
  updateSerialization();
  return _serialization.length;
 }

 public void encode(byte[] buffer, int offset)
 {
  updateSerialization();
```



```
   System.arraycopy(
    _serialization,
    0,
    buffer,
    offset,
    _serialization.length);
  }

  private void updateSerialization ( ) {
   if (_serialization != null) {
    return;
   }
   try {
    ByteArrayOutputStream bytes = new ByteArrayOutputStream();
    ObjectOutputStream objectStream =
     new ObjectOutputStream(bytes);
    objectStream.writeLong(_value);
    objectStream.close();
    _serialization = bytes.toByteArray();
   }
   catch (IOException e) {
   }
  }
}



// File: ExampleTimeFactory.java

package com.some_rti_supplier;

import hla.rti.*;
import java.io.ByteArrayInputStream;
import java.io.ObjectInputStream;

public final class ExampleTimeFactory
 implements hla.rti.LogicalTimeFactory, java.io.Serializable
{
 public LogicalTime decode(byte[] buffer, int offset)
 throws CouldNotDecode
 {
  try {
   ByteArrayInputStream bytes =
    new ByteArrayInputStream(buffer, offset, buffer.length - offset);
   ObjectInputStream objectStream =
    new ObjectInputStream(bytes);
   long value = objectStream.readLong();
   return new ExampleLogicalTime(value);
  } catch (java.io.IOException e) {
   throw new CouldNotDecode(e.getMessage());
  }
 }

 public LogicalTime makeInitial()
 {
  return ExampleLogicalTime.INITIAL;
 }

 public LogicalTime makeFinal()
 {
  return ExampleLogicalTime.FINAL;
 }
}
```

```
// File: ExampleLogicalTimeInterval.java

package com.some_rti_supplier;

import hla.rti.*;
import java.io.*;
/**
 * ExampleLogicalTimeInterval implements interface LogicalTimeInterval
 */

public class ExampleLogicalTimeInterval
 implements LogicalTimeInterval, java.io.Serializable
{
  final long _value;
 private static final long _epsilon = 1;
  private transient byte[] _serialization;

 static final ExampleLogicalTimeInterval ZERO =
  new ExampleLogicalTimeInterval(0);
 static final ExampleLogicalTimeInterval EPSILON =
  new ExampleLogicalTimeInterval(_epsilon);

 ExampleLogicalTimeInterval(long value)
 {
  _value = value;
 }

 public boolean isZero()
 {
  return _value == 0;
 }

 public boolean isEpsilon()
 {
  return _value == _epsilon;
 }

 /**
  * Returns a LogicalTimeInterval whose value is (this - subtrahend).
  */
 public LogicalTimeInterval subtract(LogicalTimeInterval subtrahend)
 {
  return new ExampleLogicalTimeInterval(
   _value - ((ExampleLogicalTimeInterval)subtrahend)._value);
 }

 public int compareTo(Object other)
 {
  double otherValue = ((ExampleLogicalTimeInterval)other)._value;
  if (_value == otherValue) {
   return 0;
  } else if (_value < otherValue) {
   return -1;
  } else {
   return 1;
  }
 }

 /**
  * Returns true iff this and other represent the same time interval.
  */
 public boolean equals(Object other)
 {
  if (other instanceof ExampleLogicalTimeInterval) {
   return equals((ExampleLogicalTimeInterval)other);
  } else {
   return false;
  }
 }

 public boolean equals(ExampleLogicalTimeInterval other)
 {
```



```
  return _value == other._value;
 }

 /**
  * Two LogicalTimes for which equals() is true should yield
  * same hash code
  */
 public int hashCode()
 {
  return (new Long(_value)).hashCode();
 }

 public String toString()
 {
  return "ExampleLTI<" + _value + ">";
 }

 public int encodedLength()
 {
  updateSerialization();
  return _serialization.length;
 }

 public void encode(byte[] buffer, int offset)
 {
  updateSerialization();
  System.arraycopy(
    _serialization,
    0,
    buffer,
    offset,
    _serialization.length);
 }

 private void updateSerialization ( ) {
  if (_serialization != null) {
   return;
  }
  try {
   ByteArrayOutputStream bytes = new ByteArrayOutputStream();
   ObjectOutputStream objectStream =
    new ObjectOutputStream(bytes);
   objectStream.writeLong(_value);
   objectStream.close();
   _serialization = bytes.toByteArray();
  }
  catch (IOException e) {
  }
 }
}



// File: ExampleIntervalFactory.java

package com.some_rti_supplier;

import hla.rti.*;

import java.io.ByteArrayInputStream;
import java.io.ObjectInputStream;

public final class ExampleIntervalFactory
 implements hla.rti.LogicalTimeIntervalFactory, java.io.Serializable
{
 public LogicalTimeInterval decode(byte[] buffer, int offset)
 throws CouldNotDecode
 {
  try {
   ByteArrayInputStream bytes =
    new ByteArrayInputStream(buffer, offset, buffer.length - offset);
```

```
    ObjectInputStream objectStream =
     new ObjectInputStream(bytes);
    long value = objectStream.readLong();
    return new ExampleLogicalTimeInterval(value);
   } catch (java.io.IOException e) {
      throw new CouldNotDecode(e.getMessage());
   }
  }

  public LogicalTimeInterval makeZero()
  {
   return ExampleLogicalTimeInterval.ZERO;
  }

  public LogicalTimeInterval makeEpsilon()
  {
   return ExampleLogicalTimeInterval.EPSILON;
  }
}



//File: AstoundingNewTransportationType.java
package com.some_rti_supplier;
import hla.rti.TransportationType;

public class AstoundingNewTransportationType extends TransportationType {

  public AstoundingNewTransportationType(
   AstoundingNewTransportationType otherAstoundingNewTransportationTypeValue) {
   _value = otherAstoundingNewTransportationTypeValue._value;
  }
  protected AstoundingNewTransportationType() {
   _value = _nextToAssign++;
  }

  static public final AstoundingNewTransportationType WHIZZBANG
   = new AstoundingNewTransportationType();
}
```


Annex C

(normative)

C++ application programmer's interface

C.1 API

The following files shall comprise the C++ HLA API.

```
/**********************************************************************
IEEE 1516.1 High Level Architecture Interface Specification C++ API
File: RTI_RTIambassador.h
**********************************************************************/

// This interface is used to access the services of the RTI.

#ifndef RTI_RTIambassador_h
#define RTI_RTIambassador_h

//
// These forward declarations significantly decrease compilation time when only
// including this file
//

class RTI_AsynchronousDeliveryAlreadyDisabled;
class RTI_AsynchronousDeliveryAlreadyEnabled;
class RTI_AttributeAcquisitionWasNotRequested;
class RTI_AttributeAlreadyBeingAcquired;
class RTI_AttributeAlreadyBeingDivested;
class RTI_AttributeAlreadyOwned;
class RTI_AttributeDivestitureWasNotRequested;
class RTI_AttributeNotDefined;
class RTI_AttributeNotOwned;
class RTI_AttributeNotPublished;
class RTI_AttributeRelevanceAdvisorySwitchIsOff;
class RTI_AttributeRelevanceAdvisorySwitchIsOn;
class RTI_AttributeScopeAdvisorySwitchIsOff;
class RTI_AttributeScopeAdvisorySwitchIsOn;
class RTI_CouldNotOpenFDD;
class RTI_DeletePrivilegeNotHeld;
class RTI_ErrorReadingFDD;
class RTI_FederateAlreadyExecutionMember;
class RTI_FederateAmbassador;
class RTI_FederateHasNotBegunSave;
class RTI_FederateNotExecutionMember;
class RTI_FederateOwnsAttributes;
class RTI_FederateServiceInvocationsAreBeingReportedViaMOM;
class RTI_FederateUnableToUseTime;
class RTI_FederatesCurrentlyJoined;
class RTI_FederationExecutionAlreadyExists;
class RTI_FederationExecutionDoesNotExist ;
class RTI_FederationExecutionDoesNotExist;
class RTI_IllegalName;
class RTI_InTimeAdvancingState;
class RTI_InteractionClassNotDefined;
class RTI_InteractionClassNotPublished;
class RTI_InteractionParameterNotDefined;
class RTI_InteractionRelevanceAdvisorySwitchIsOff;
class RTI_InteractionRelevanceAdvisorySwitchIsOn;
```

```
class RTI_InvalidAttributeHandle;
class RTI_InvalidDimensionHandle;
class RTI_InvalidInteractionClassHandle;
class RTI_InvalidLogicalTime;
class RTI_InvalidLookahead;
class RTI_InvalidObjectClassHandle;
class RTI_InvalidOrderName;
class RTI_InvalidOrderType;
class RTI_InvalidParameterHandle;
class RTI_InvalidRangeBounds;
class RTI_InvalidRegion;
class RTI_InvalidRegionContext;
class RTI_InvalidRetractionHandle;
class RTI_InvalidTransportationName;
class RTI_InvalidTransportationType;
class RTI_LogicalTime;
class RTI_LogicalTimeAlreadyPassed ;
class RTI_LogicalTimeAlreadyPassed;
class RTI_LogicalTimeFactory;
class RTI_LogicalTimeInterval;
class RTI_LogicalTimeIntervalFactory;
class RTI_MessageCanNoLongerBeRetracted;
class RTI_NameNotFound;
class RTI_ObjectClassNotDefined ;
class RTI_ObjectClassNotDefined;
class RTI_ObjectClassNotPublished;
class RTI_ObjectClassRelevanceAdvisorySwitchIsOff;
class RTI_ObjectClassRelevanceAdvisorySwitchIsOn;
class RTI_ObjectInstanceNameInUse;
class RTI_ObjectInstanceNameNotReserved;
class RTI_ObjectInstanceNotKnown ;
class RTI_ObjectInstanceNotKnown;
class RTI_OrderType;
class RTI_OwnershipAcquisitionPending;
class RTI_RTIinternalError;
class RTI_RangeBounds;
class RTI_RegionDoesNotContainSpecifiedDimension;
class RTI_RegionInUseForUpdateOrSubscription;
class RTI_RegionNotCreatedByThisFederate;
class RTI_RequestForTimeConstrainedPending;
class RTI_RequestForTimeConstrainedPending;
class RTI_RequestForTimeRegulationPending;
class RTI_ResignAction;
class RTI_RestoreInProgress;
class RTI_RestoreNotInProgress;
class RTI_RestoreNotRequested;
class RTI_RestoreStatus;
class RTI_SaveInProgress;
class RTI_SaveNotInProgress;
class RTI_SaveNotInitiated;
class RTI_SaveStatus;
class RTI_ServiceGroupIndicator;
class RTI_SynchronizationPointLabelNotAnnounced;
class RTI_TimeConstrainedAlreadyEnabled;
class RTI_TimeConstrainedIsNotEnabled;
class RTI_TimeRegulationAlreadyEnabled;
class RTI_TimeRegulationIsNotEnabled;
class RTI_TransportationType;

#include <RTI_SpecificConfig.h>
#include <RTI_memory>
#include <RTI_string>
#include <RTI_bool.h>
#include <RTI_Typedefs.h>

class RTI_RTIambassador
{
public:
 // 10.37
 virtual ~RTI_RTIambassador ()
  throw ();
```



```
// 4.2
virtual void createFederationExecution
(RTI_wstring const & federationExecutionName,
 RTI_wstring const & fullPathNameToTheFDDfile)
  throw (RTI_FederationExecutionAlreadyExists,
         RTI_CouldNotOpenFDD,
         RTI_ErrorReadingFDD,
         RTI_RTIinternalError) = 0;

// 4.3
virtual void destroyFederationExecution
(RTI_wstring const & federationExecutionName)
  throw (RTI_FederatesCurrentlyJoined,
         RTI_FederationExecutionDoesNotExist,
         RTI_RTIinternalError) = 0;

// 4.4
virtualRTI_FederateHandle const & joinFederationExecution
(RTI_wstring const & federateType,
 RTI_wstring const & federationExecutionName,
 RTI_auto_ptr< RTI_FederateAmbassador > federateAmbassador,
 RTI_auto_ptr< RTI_LogicalTimeFactory > logicalTimeFactory,
 RTI_auto_ptr< RTI_LogicalTimeIntervalFactory > logicalTimeIntervalFactory)
  throw (RTI_FederateAlreadyExecutionMember,
         RTI_FederationExecutionDoesNotExist,
         RTI_SaveInProgress,
         RTI_RestoreInProgress,
         RTI_RTIinternalError) = 0;

// 4.5
virtual void resignFederationExecution
(RTI_ResignAction resignAction)
  throw (RTI_OwnershipAcquisitionPending,
         RTI_FederateOwnsAttributes,
         RTI_FederateNotExecutionMember,
         RTI_RTIinternalError) = 0;

// 4.6
virtual void registerFederationSynchronizationPoint
(RTI_wstringconst & label,
 RTI_UserSuppliedTag const & theUserSuppliedTag)
  throw (RTI_FederateNotExecutionMember,
         RTI_SaveInProgress,
         RTI_RestoreInProgress,
         RTI_RTIinternalError) = 0;

virtual void registerFederationSynchronizationPoint
(RTI_wstring const & label,
 RTI_UserSuppliedTag const & theUserSuppliedTag,
 RTI_FederateHandleSet const & syncSet)
  throw (RTI_FederateNotExecutionMember,
         RTI_SaveInProgress,
         RTI_RestoreInProgress,
         RTI_RTIinternalError) = 0;

// 4.9
virtual void synchronizationPointAchieved
(RTI_wstring const & label)
  throw (RTI_SynchronizationPointLabelNotAnnounced,
         RTI_FederateNotExecutionMember,
         RTI_SaveInProgress,
         RTI_RestoreInProgress,
         RTI_RTIinternalError) = 0;

// 4.11
virtual void requestFederationSave
(RTI_wstring const & label)
  throw (RTI_FederateNotExecutionMember,
         RTI_SaveInProgress,
         RTI_RestoreInProgress,
         RTI_RTIinternalError) = 0;
```

```
 virtual void requestFederationSave
 (RTI_wstring const & label,
  RTI_LogicalTime const & theTime)
   throw (RTI_LogicalTimeAlreadyPassed,
          RTI_InvalidLogicalTime,
     RTI_FederateUnableToUseTime,
         RTI_FederateNotExecutionMember,
         RTI_SaveInProgress,
         RTI_RestoreInProgress,
         RTI_RTIinternalError) = 0;

 // 4.13
 virtual void federateSaveBegun ()
   throw (RTI_SaveNotInitiated,
          RTI_FederateNotExecutionMember,
          RTI_RestoreInProgress,
          RTI_RTIinternalError) = 0;

 // 4.14
 virtual void federateSaveComplete ()
   throw (RTI_FederateHasNotBegunSave,
          RTI_FederateNotExecutionMember,
          RTI_RestoreInProgress,
          RTI_RTIinternalError) = 0;

 virtual void federateSaveNotComplete()
   throw (RTI_FederateHasNotBegunSave,
          RTI_FederateNotExecutionMember,
          RTI_RestoreInProgress,
          RTsI_RTIinternalError) = 0;

 // 4.16
 virtual void queryFederationSaveStatus ()
   throw (RTI_FederateNotExecutionMember,
          RTI_SaveNotInProgress,
          RTI_RestoreInProgress,
          RTI_RTIinternalError) = 0;

 // 4.18
 virtual void requestFederationRestore
 (RTI_wstring const & label)
   throw (RTI_FederateNotExecutionMember,
          RTI_SaveInProgress,
          RTI_RestoreInProgress,
          RTI_RTIinternalError) = 0;

 // 4.22
 virtual void federateRestoreComplete ()
   throw (RTI_RestoreNotRequested,
          RTI_FederateNotExecutionMember,
          RTI_SaveInProgress,
          RTI_RTIinternalError) = 0;

 virtual void federateRestoreNotComplete ()
   throw (RTI_RestoreNotRequested,
          RTI_FederateNotExecutionMember,
          RTI_SaveInProgress,
          RTI_RTIinternalError) = 0;

 // 4.24
 virtual void queryFederationRestoreStatus ()
   throw (RTI_FederateNotExecutionMember,
          RTI_SaveInProgress,
          RTI_RestoreNotInProgress,
          RTI_RTIinternalError) = 0;

 ////////////////////////////////////
 // Declaration Management Services //
 ////////////////////////////////////

 // 5.2
 virtual void publishObjectClassAttributes
```



```
(RTI_ObjectClassHandleconst & theClass,
 RTI_AttributeHandleSet const & attributeList)
  throw (RTI_ObjectClassNotDefined,
         RTI_AttributeNotDefined,
         RTI_FederateNotExecutionMember,
         RTI_SaveInProgress,
         RTI_RestoreInProgress,
         RTI_RTIinternalError) = 0;

// 5.3
virtual void unpublishObjectClass
(RTI_ObjectClassHandle const & theClass)
  throw (RTI_ObjectClassNotDefined,
         RTI_OwnershipAcquisitionPending,
         RTI_FederateNotExecutionMember,
         RTI_SaveInProgress,
         RTI_RestoreInProgress,
         RTI_RTIinternalError) = 0;

virtual void unpublishObjectClassAttributes
(RTI_ObjectClassHandleconst & theClass,
 RTI_AttributeHandleSet const & attributeList)
  throw (RTI_ObjectClassNotDefined,
         RTI_AttributeNotDefined,
         RTI_OwnershipAcquisitionPending,
         RTI_FederateNotExecutionMember,
         RTI_SaveInProgress,
         RTI_RestoreInProgress,
         RTI_RTIinternalError) = 0;

// 5.4
virtual void publishInteractionClass
(RTI_InteractionClassHandle const & theInteraction)
  throw (RTI_InteractionClassNotDefined,
         RTI_FederateNotExecutionMember,
         RTI_SaveInProgress,
         RTI_RestoreInProgress,
         RTI_RTIinternalError) = 0;

// 5.5
virtual void unpublishInteractionClass
(RTI_InteractionClassHandle const & theInteraction)
         throw (RTI_InteractionClassNotDefined,
         RTI_FederateNotExecutionMember,
         RTI_SaveInProgress,
         RTI_RestoreInProgress,
         RTI_RTIinternalError) = 0;

// 5.6
virtual void subscribeObjectClassAttributes
(RTI_ObjectClassHandleconst & theClass,
 RTI_AttributeHandleSet const & attributeList,
 RTI_bool active = RTI_true)
  throw (RTI_ObjectClassNotDefined,
         RTI_AttributeNotDefined,
         RTI_FederateNotExecutionMember,
         RTI_SaveInProgress,
         RTI_RestoreInProgress,
         RTI_RTIinternalError) = 0;

// 5.7
virtual void unsubscribeObjectClass
(RTI_ObjectClassHandle const & theClass)
  throw (RTI_ObjectClassNotDefined,
         RTI_FederateNotExecutionMember,
         RTI_SaveInProgress,
         RTI_RestoreInProgress,
         RTI_RTIinternalError) = 0;

virtual void unsubscribeObjectClassAttributes
(RTI_ObjectClassHandleconst & theClass,
 RTI_AttributeHandleSet const & attributeList)
```

```
    throw (RTI_ObjectClassNotDefined,
           RTI_AttributeNotDefined,
           RTI_FederateNotExecutionMember,
           RTI_SaveInProgress,
           RTI_RestoreInProgress,
           RTI_RTIinternalError) = 0;

  // 5.8
  virtual void subscribeInteractionClass
  (RTI_InteractionClassHandle const & theClass,
   RTI_bool active = RTI_true)
    throw (RTI_InteractionClassNotDefined,
           RTI_FederateServiceInvocationsAreBeingReportedViaMOM,
           RTI_FederateNotExecutionMember,
           RTI_SaveInProgress,
           RTI_RestoreInProgress,
           RTI_RTIinternalError) = 0;

  // 5.9
  virtual void unsubscribeInteractionClass
  (RTI_InteractionClassHandle const & theClass)
    throw (RTI_InteractionClassNotDefined,
           RTI_FederateNotExecutionMember,
           RTI_SaveInProgress,
           RTI_RestoreInProgress,
           RTI_RTIinternalError) = 0;

  ////////////////////////////////
  // Object Management Services //
  ////////////////////////////////

  // 6.2
  virtual void reserveObjectInstanceName
  (RTI_wstring const & theObjectInstanceName)
    throw (RTI_IllegalName,
           RTI_FederateNotExecutionMember,
           RTI_SaveInProgress,
           RTI_RestoreInProgress,
           RTI_RTIinternalError) = 0;

  // 6.4
  virtual RTI_ObjectInstanceHandle const & registerObjectInstance
  (RTI_ObjectClassHandle const & theClass)
    throw (RTI_ObjectClassNotDefined,
           RTI_ObjectClassNotPublished,
           RTI_FederateNotExecutionMember,
           RTI_SaveInProgress,
           RTI_RestoreInProgress,
           RTI_RTIinternalError) = 0;

  virtual RTI_ObjectInstanceHandle const & registerObjectInstance
  (RTI_ObjectClassHandle const & theClass,
   RTI_wstring const & theObjectInstanceName)
    throw (RTI_ObjectClassNotDefined,
           RTI_ObjectClassNotPublished,
           RTI_ObjectInstanceNameNotReserved,
           RTI_ObjectInstanceNameInUse,
           RTI_FederateNotExecutionMember,
           RTI_SaveInProgress,
           RTI_RestoreInProgress,
           RTI_RTIinternalError) = 0;

  // 6.6
  virtual void updateAttributeValues
  (RTI_ObjectInstanceHandle const & theObject,
   RTI_AttributeHandleValueMap const & theAttributeValues,
   RTI_UserSuppliedTag const & theUserSuppliedTag)
    throw (RTI_ObjectInstanceNotKnown,
           RTI_AttributeNotDefined,
           RTI_AttributeNotOwned,
           RTI_FederateNotExecutionMember,
           RTI_SaveInProgress,
```



```
          RTI_RestoreInProgress,
          RTI_RTIinternalError) = 0;

 virtual RTI_auto_ptr< RTI_MessageRetractionHandle > updateAttributeValues
 (RTI_ObjectInstanceHandle const & theObject,
  RTI_AttributeHandleValueMap const & theAttributeValues,
  RTI_UserSuppliedTag const & theUserSuppliedTag,
  RTI_LogicalTime const & theTime)
   throw (RTI_ObjectInstanceNotKnown,
          RTI_AttributeNotDefined,
          RTI_AttributeNotOwned,
          RTI_InvalidLogicalTime,
          RTI_FederateNotExecutionMember,
          RTI_SaveInProgress,
          RTI_RestoreInProgress,
          RTI_RTIinternalError) = 0;

 // 6.8
 virtual void sendInteraction
 (RTI_InteractionClassHandleconst & theInteraction,
  RTI_ParameterHandleValueMap const & theParameterValues,
  RTI_UserSuppliedTag const & theUserSuppliedTag)
   throw (RTI_InteractionClassNotPublished,
          RTI_InteractionClassNotDefined,
          RTI_InteractionParameterNotDefined,
          RTI_FederateNotExecutionMember,
          RTI_SaveInProgress,
          RTI_RestoreInProgress,
          RTI_RTIinternalError) = 0;

 virtual RTI_auto_ptr< RTI_MessageRetractionHandle > sendInteraction
 (RTI_InteractionClassHandleconst & theInteraction,
  RTI_ParameterHandleValueMap const & theParameterValues,
  RTI_UserSuppliedTag const & theUserSuppliedTag,
  RTI_LogicalTime const & theTime)
   throw (RTI_InteractionClassNotPublished,
          RTI_InteractionClassNotDefined,
          RTI_InteractionParameterNotDefined,
          RTI_InvalidLogicalTime,
          RTI_FederateNotExecutionMember,
          RTI_SaveInProgress,
          RTI_RestoreInProgress,
          RTI_RTIinternalError) = 0;

 // 6.10
 virtual void deleteObjectInstance
 (RTI_ObjectInstanceHandle const & theObject,
  RTI_UserSuppliedTag const & theUserSuppliedTag)
   throw (RTI_DeletePrivilegeNotHeld,
          RTI_ObjectInstanceNotKnown,
          RTI_FederateNotExecutionMember,
          RTI_SaveInProgress,
          RTI_RestoreInProgress,
          RTI_RTIinternalError) = 0;

 virtual RTI_auto_ptr< RTI_MessageRetractionHandle > deleteObjectInstance
 (RTI_ObjectInstanceHandle const & theObject,
  RTI_UserSuppliedTag const & theUserSuppliedTag,
  RTI_LogicalTimeconst & theTime)
   throw (RTI_DeletePrivilegeNotHeld,
          RTI_ObjectInstanceNotKnown,
          RTI_InvalidLogicalTime,
          RTI_FederateNotExecutionMember,
          RTI_SaveInProgress,
          RTI_RestoreInProgress,
          RTI_RTIinternalError) = 0;

 // 6.12
 virtual void localDeleteObjectInstance
 (RTI_ObjectInstanceHandle const & theObject)
   throw (RTI_ObjectInstanceNotKnown,
          RTI_FederateOwnsAttributes,
```

```
          RTI_OwnershipAcquisitionPending,
          RTI_FederateNotExecutionMember,
          RTI_SaveInProgress,
          RTI_RestoreInProgress,
          RTI_RTIinternalError) = 0;

 // 6.13
 virtual void changeAttributeTransportationType
 (RTI_ObjectInstanceHandle const & theObject,
  RTI_AttributeHandleSet const & theAttributes,
  RTI_TransportationType const & theType)
   throw (RTI_ObjectInstanceNotKnown,
          RTI_AttributeNotDefined,
          RTI_AttributeNotOwned,
          RTI_FederateNotExecutionMember,
          RTI_SaveInProgress,
          RTI_RestoreInProgress,
          RTI_RTIinternalError) = 0;

 // 6.14
 virtual void changeInteractionTransportationType
 (RTI_InteractionClassHandle const & theClass,
  RTI_TransportationType const & theType)
   throw (RTI_InteractionClassNotDefined,
          RTI_InteractionClassNotPublished,
          RTI_FederateNotExecutionMember,
          RTI_SaveInProgress,
          RTI_RestoreInProgress,
          RTI_RTIinternalError) = 0;

 // 6.17
 virtual void requestAttributeValueUpdate
 (RTI_ObjectInstanceHandle const & theObject,
  RTI_AttributeHandleSet const & theAttributes,
  RTI_UserSuppliedTag const & theUserSuppliedTag)
   throw (RTI_ObjectInstanceNotKnown,
          RTI_AttributeNotDefined,
          RTI_FederateNotExecutionMember,
          RTI_SaveInProgress,
          RTI_RestoreInProgress,
          RTI_RTIinternalError) = 0;

 virtual void requestAttributeValueUpdate
 (RTI_ObjectClassHandle const & theClass,
  RTI_AttributeHandleSet const & theAttributes,
  RTI_UserSuppliedTag const & theUserSuppliedTag)
   throw (RTI_ObjectClassNotDefined,
          RTI_AttributeNotDefined,
          RTI_FederateNotExecutionMember,
          RTI_SaveInProgress,
          RTI_RestoreInProgress,
          RTI_RTIinternalError) = 0;

 ////////////////////////////////////
 // Ownership Management Services //
 ////////////////////////////////////

 // 7.2
 virtual void unconditionalAttributeOwnershipDivestiture
 (RTI_ObjectInstanceHandle const & theObject,
  RTI_AttributeHandleSet const & theAttributes)
   throw (RTI_ObjectInstanceNotKnown,
          RTI_AttributeNotDefined,
          RTI_AttributeNotOwned,
          RTI_FederateNotExecutionMember,
          RTI_SaveInProgress,
          RTI_RestoreInProgress,
          RTI_RTIinternalError) = 0;

 // 7.3
 virtual void negotiatedAttributeOwnershipDivestiture
 (RTI_ObjectInstanceHandle const & theObject,
```



```
 RTI_AttributeHandleSet const & theAttributes,
 RTI_UserSuppliedTag const & theUserSuppliedTag)
  throw (RTI_ObjectInstanceNotKnown,
         RTI_AttributeNotDefined,
         RTI_AttributeNotOwned,
         RTI_AttributeAlreadyBeingDivested,
         RTI_FederateNotExecutionMember,
         RTI_SaveInProgress,
         RTI_RestoreInProgress,
         RTI_RTIinternalError) = 0;

// 7.6
virtual
void
confirmDivestiture
(RTI_ObjectInstanceHandle const & theObject,
 RTI_auto_ptr< RTI_AttributeHandleSet >securedAttributes,
 RTI_UserSuppliedTagconst & theUserSuppliedTag)
  throw (RTI_ObjectInstanceNotKnown,
         RTI_AttributeNotDefined,
         RTI_AttributeNotOwned,
         RTI_AttributeDivestitureWasNotRequested,
         RTI_FederateNotExecutionMember,
         RTI_SaveInProgress,
         RTI_RestoreInProgress,
         RTI_RTIinternalError) = 0;

// 7.8
virtual void attributeOwnershipAcquisition
(RTI_ObjectInstanceHandle const & theObject,
 RTI_AttributeHandleSet const & desiredAttributes,
 RTI_UserSuppliedTag const & theUserSuppliedTag)
  throw (RTI_ObjectInstanceNotKnown,
         RTI_ObjectClassNotPublished,
         RTI_AttributeNotDefined,
         RTI_AttributeNotPublished,
         RTI_FederateOwnsAttributes,
         RTI_FederateNotExecutionMember,
         RTI_SaveInProgress,
         RTI_RestoreInProgress,
         RTI_RTIinternalError) = 0;

// 7.9
virtual void attributeOwnershipAcquisitionIfAvailable
(RTI_ObjectInstanceHandle const & theObject,
 RTI_AttributeHandleSet const & desiredAttributes)
  throw (RTI_ObjectInstanceNotKnown,
         RTI_ObjectClassNotPublished,
         RTI_AttributeNotDefined,
         RTI_AttributeNotPublished,
         RTI_FederateOwnsAttributes,
         RTI_AttributeAlreadyBeingAcquired,
         RTI_FederateNotExecutionMember,
         RTI_SaveInProgress,
         RTI_RestoreInProgress,
         RTI_RTIinternalError) = 0;

// 7.12
virtual RTI_auto_ptr< RTI_AttributeHandleSet >
 attributeOwnershipDivestitureIfWanted
(RTI_ObjectInstanceHandle const & theObject,
 RTI_AttributeHandleSet const & theAttributes)
  throw (RTI_ObjectInstanceNotKnown,
         RTI_AttributeNotDefined,
         RTI_AttributeNotOwned,
         RTI_FederateNotExecutionMember,
         RTI_SaveInProgress,
         RTI_RestoreInProgress,
         RTI_RTIinternalError) = 0;

// 7.13
virtual void cancelNegotiatedAttributeOwnershipDivestiture
```

```
(RTI_ObjectInstanceHandle const & theObject,
 RTI_AttributeHandleSet const & theAttributes)
  throw (RTI_ObjectInstanceNotKnown,
         RTI_AttributeNotDefined,
         RTI_AttributeNotOwned,
         RTI_AttributeDivestitureWasNotRequested,
         RTI_FederateNotExecutionMember,
         RTI_SaveInProgress,
         RTI_RestoreInProgress,
         RTI_RTIinternalError) = 0;

// 7.14
virtual void cancelAttributeOwnershipAcquisition
(RTI_ObjectInstanceHandle const & theObject,
 RTI_AttributeHandleSet const & theAttributes)
  throw (RTI_ObjectInstanceNotKnown,
         RTI_AttributeNotDefined,
         RTI_AttributeAlreadyOwned,
         RTI_AttributeAcquisitionWasNotRequested,
         RTI_FederateNotExecutionMember,
         RTI_SaveInProgress,
         RTI_RestoreInProgress,
         RTI_RTIinternalError) = 0;

// 7.16
virtual void queryAttributeOwnership
(RTI_ObjectInstanceHandle const & theObject,
 RTI_AttributeHandle const & theAttribute)
  throw (RTI_ObjectInstanceNotKnown,
         RTI_AttributeNotDefined,
         RTI_FederateNotExecutionMember,
         RTI_SaveInProgress,
         RTI_RestoreInProgress,
         RTI_RTIinternalError) = 0;

// 7.18
virtual RTI_bool isAttributeOwnedByFederate
(RTI_ObjectInstanceHandle const & theObject,
 RTI_AttributeHandle const & theAttribute)
  throw (RTI_ObjectInstanceNotKnown,
         RTI_AttributeNotDefined,
         RTI_FederateNotExecutionMember,
         RTI_SaveInProgress,
         RTI_RestoreInProgress,
         RTI_RTIinternalError) = 0;

//////////////////////////////
// Time Management Services //
//////////////////////////////

// 8.2
virtual void enableTimeRegulation
(RTI_LogicalTimeInterval const & theLookahead)
  throw (RTI_TimeRegulationAlreadyEnabled,
         RTI_InvalidLookahead,
         RTI_InTimeAdvancingState,
         RTI_RequestForTimeRegulationPending,
         RTI_FederateNotExecutionMember,
         RTI_SaveInProgress,
         RTI_RestoreInProgress,
         RTI_RTIinternalError) = 0;

// 8.4
virtual void disableTimeRegulation ()
  throw (RTI_TimeRegulationIsNotEnabled,
         RTI_FederateNotExecutionMember,
         RTI_SaveInProgress,
         RTI_RestoreInProgress,
         RTI_RTIinternalError) = 0;

// 8.5
virtual void enableTimeConstrained ()
```



```
  throw (RTI_TimeConstrainedAlreadyEnabled,
         RTI_InTimeAdvancingState,
         RTI_RequestForTimeConstrainedPending,
         RTI_FederateNotExecutionMember,
         RTI_SaveInProgress,
         RTI_RestoreInProgress,
         RTI_RTIinternalError) = 0;

// 8.7
virtual void disableTimeConstrained ()
  throw (RTI_TimeConstrainedIsNotEnabled,
         RTI_FederateNotExecutionMember,
         RTI_SaveInProgress,
         RTI_RestoreInProgress,
         RTI_RTIinternalError) = 0;

// 8.8
virtual void timeAdvanceRequest
(RTI_LogicalTime const & theTime)
  throw (RTI_InvalidLogicalTime,
         RTI_LogicalTimeAlreadyPassed,
         RTI_InTimeAdvancingState,
         RTI_RequestForTimeRegulationPending,
         RTI_RequestForTimeConstrainedPending,
         RTI_FederateNotExecutionMember,
         RTI_SaveInProgress,
         RTI_RestoreInProgress,
         RTI_RTIinternalError) = 0;

// 8.9
virtual void timeAdvanceRequestAvailable
(RTI_LogicalTime const & theTime)
  throw (RTI_InvalidLogicalTime,
         RTI_LogicalTimeAlreadyPassed,
         RTI_InTimeAdvancingState,
         RTI_RequestForTimeRegulationPending,
         RTI_RequestForTimeConstrainedPending,
         RTI_FederateNotExecutionMember,
         RTI_SaveInProgress,
         RTI_RestoreInProgress,
         RTI_RTIinternalError) = 0;

// 8.10
virtual void nextMessageRequest
(RTI_LogicalTime const & theTime)
  throw (RTI_InvalidLogicalTime,
         RTI_LogicalTimeAlreadyPassed,
         RTI_InTimeAdvancingState,
         RTI_RequestForTimeRegulationPending,
         RTI_RequestForTimeConstrainedPending,
         RTI_FederateNotExecutionMember,
         RTI_SaveInProgress,
         RTI_RestoreInProgress,
         RTI_RTIinternalError) = 0;

// 8.11
virtual void nextMessageRequestAvailable
(RTI_LogicalTime const & theTime)
  throw (RTI_InvalidLogicalTime,
         RTI_LogicalTimeAlreadyPassed,
         RTI_InTimeAdvancingState,
         RTI_RequestForTimeRegulationPending,
         RTI_RequestForTimeConstrainedPending,
         RTI_FederateNotExecutionMember,
         RTI_SaveInProgress,
         RTI_RestoreInProgress,
         RTI_RTIinternalError) = 0;

// 8.12
virtual void flushQueueRequest
(RTI_LogicalTime const & theTime)
  throw (RTI_InvalidLogicalTime,
```

```
          RTI_LogicalTimeAlreadyPassed,
          RTI_InTimeAdvancingState,
          RTI_RequestForTimeRegulationPending,
          RTI_RequestForTimeConstrainedPending,
          RTI_FederateNotExecutionMember,
          RTI_SaveInProgress,
          RTI_RestoreInProgress,
          RTI_RTIinternalError) = 0;

 // 8.14
 virtual void enableAsynchronousDelivery ()
   throw (RTI_AsynchronousDeliveryAlreadyEnabled,
          RTI_FederateNotExecutionMember,
          RTI_SaveInProgress,
          RTI_RestoreInProgress,
          RTI_RTIinternalError) = 0;

 // 8.15
   virtual void disableAsynchronousDelivery ()
   throw (RTI_AsynchronousDeliveryAlreadyDisabled,
          RTI_FederateNotExecutionMember,
          RTI_SaveInProgress,
          RTI_RestoreInProgress,
          RTI_RTIinternalError) = 0;

 // 8.16
 virtual RTI_auto_ptr< RTI_LogicalTime > queryGALT ()
   throw (RTI_FederateNotExecutionMember,
          RTI_SaveInProgress,
          RTI_RestoreInProgress,
          RTI_RTIinternalError) = 0;

 // 8.17
virtual RTI_LogicalTime const & queryLogicalTime ()
   throw (RTI_FederateNotExecutionMember,
          RTI_SaveInProgress,
          RTI_RestoreInProgress,
          RTI_RTIinternalError) = 0;

 // 8.18

 virtual RTI_auto_ptr< RTI_LogicalTime > const & queryLITS ()
   throw (RTI_FederateNotExecutionMember,
          RTI_SaveInProgress,
          RTI_RestoreInProgress,
          RTI_RTIinternalError) = 0;

 // 8.19
 virtual void modifyLookahead
 (RTI_LogicalTimeInterval const & theLookahead)
   throw (RTI_TimeRegulationIsNotEnabled,
          RTI_InvalidLookahead,
          RTI_InTimeAdvancingState,
          RTI_FederateNotExecutionMember,
          RTI_SaveInProgress,
          RTI_RestoreInProgress,
          RTI_RTIinternalError) = 0;

 // 8.20
virtual RTI_LogicalTimeInterval const & queryLookahead ()
   throw (RTI_TimeRegulationIsNotEnabled,
          RTI_FederateNotExecutionMember,
          RTI_SaveInProgress,
          RTI_RestoreInProgress,
          RTI_RTIinternalError) = 0;

 // 8.21
 virtual void retract
 (RTI_MessageRetractionHandle const & theHandle)
   throw (RTI_InvalidRetractionHandle,
          RTI_TimeRegulationIsNotEnabled,
          RTI_MessageCanNoLongerBeRetracted,
```



```
          RTI_FederateNotExecutionMember,
          RTI_SaveInProgress,
          RTI_RestoreInProgress,
          RTI_RTIinternalError) = 0;

 // 8.23
 virtual void changeAttributeOrderType
 (RTI_ObjectInstanceHandle const & theObject,
  RTI_AttributeHandleSet const & theAttributes,
  RTI_OrderType const & theType)
   throw (RTI_ObjectInstanceNotKnown,
          RTI_AttributeNotDefined,
          RTI_AttributeNotOwned,
          RTI_FederateNotExecutionMember,
          RTI_SaveInProgress,
          RTI_RestoreInProgress,
          RTI_RTIinternalError) = 0;

 // 8.24
 virtual void changeInteractionOrderType
 (RTI_InteractionClassHandle const & theClass,
  RTI_OrderType const & theType)
   throw (RTI_InteractionClassNotDefined,
          RTI_InteractionClassNotPublished,
          RTI_FederateNotExecutionMember,
          RTI_SaveInProgress,
          RTI_RestoreInProgress,
          RTI_RTIinternalError) = 0;

 ///////////////////////////////////
 // Data Distribution Management //
 ///////////////////////////////////

 // 9.2
 virtual RTI_auto_ptr< RTI_RegionHandle > createRegion
 (RTI_DimensionHandleSet const & theDimensions)
   throw (RTI_InvalidDimensionHandle,
     RTI_FederateNotExecutionMember,
          RTI_SaveInProgress,
          RTI_RestoreInProgress,
          RTI_RTIinternalError) = 0;

 // 9.3
 virtual void commitRegionModifications
 (RTI_RegionHandleSet const & theRegionHandleSet)
   throw (RTI_InvalidRegion,
     RTI_RegionNotCreatedByThisFederate,
          RTI_FederateNotExecutionMember,
          RTI_SaveInProgress,
          RTI_RestoreInProgress,
          RTI_RTIinternalError) = 0;

 // 9.4
 virtual void deleteRegion
 (RTI_auto_ptr< RTI_RegionHandle > theRegion)
   throw (RTI_InvalidRegion,
     RTI_RegionNotCreatedByThisFederate,
          RTI_RegionInUseForUpdateOrSubscription,
          RTI_FederateNotExecutionMember,
          RTI_SaveInProgress,
          RTI_RestoreInProgress,
          RTI_RTIinternalError) = 0;

 // 9.5
 virtual RTI_ObjectInstanceHandle const & registerObjectInstanceWithRegions
 (RTI_ObjectClassHandle const & theClass,
  RTI_AttributeHandleSetRegionHandleSetPairVector const &
theAttributeHandleSetRegionHandleSetPairVector)
   throw (RTI_ObjectClassNotDefined,
          RTI_ObjectClassNotPublished,
          RTI_AttributeNotDefined,
          RTI_AttributeNotPublished,
```

```
      RTI_InvalidRegion,
           RTI_RegionNotCreatedByThisFederate,
           RTI_InvalidRegionContext,
           RTI_FederateNotExecutionMember,
           RTI_SaveInProgress,
           RTI_RestoreInProgress,
           RTI_RTIinternalError) = 0;

 virtual RTI_ObjectInstanceHandle const & registerObjectInstanceWithRegions
 (RTI_ObjectClassHandle const & theClass,
  RTI_AttributeHandleSetRegionHandleSetPairVector const &
theAttributeHandleSetRegionHandleSetPairVector,
  RTI_wstring const & theObjectInstanceName)
   throw (RTI_ObjectClassNotDefined,
           RTI_ObjectClassNotPublished,
           RTI_AttributeNotDefined,
           RTI_AttributeNotPublished,
      RTI_InvalidRegion,
           RTI_RegionNotCreatedByThisFederate,
           RTI_InvalidRegionContext,
           RTI_ObjectInstanceNameNotReserved,
           RTI_ObjectInstanceNameInUse,
           RTI_FederateNotExecutionMember,
           RTI_SaveInProgress,
           RTI_RestoreInProgress,
           RTI_RTIinternalError) = 0;

 // 9.6
 virtual void associateRegionsForUpdates
 (RTI_ObjectInstanceHandle const & theObject,
  RTI_AttributeHandleSetRegionHandleSetPairVector const &
theAttributeHandleSetRegionHandleSetPairVector)
   throw (RTI_ObjectInstanceNotKnown,
           RTI_AttributeNotDefined,
      RTI_InvalidRegion,
           RTI_RegionNotCreatedByThisFederate,
           RTI_InvalidRegionContext,
           RTI_FederateNotExecutionMember,
           RTI_SaveInProgress,
           RTI_RestoreInProgress,
           RTI_RTIinternalError) = 0;

 // 9.7
 virtual void unassociateRegionsForUpdates
 (RTI_ObjectInstanceHandle const & theObject,
  RTI_AttributeHandleSetRegionHandleSetPairVector const &
theAttributeHandleSetRegionHandleSetPairVector)
   throw (RTI_ObjectInstanceNotKnown,
           RTI_AttributeNotDefined,
      RTI_InvalidRegion,
           RTI_RegionNotCreatedByThisFederate,
           RTI_FederateNotExecutionMember,
           RTI_SaveInProgress,
           RTI_RestoreInProgress,
           RTI_RTIinternalError) = 0;

 // 9.8
 virtual void subscribeObjectClassAttributesWithRegions
 (RTI_ObjectClassHandle const & theClass,
  RTI_AttributeHandleSetRegionHandleSetPairVector const &
theAttributeHandleSetRegionHandleSetPairVector,
  RTI_bool active = RTI_true)
   throw (RTI_ObjectClassNotDefined,
           RTI_AttributeNotDefined,
      RTI_InvalidRegion,
           RTI_RegionNotCreatedByThisFederate,
           RTI_InvalidRegionContext,
           RTI_FederateNotExecutionMember,
           RTI_SaveInProgress,
           RTI_RestoreInProgress,
           RTI_RTIinternalError) = 0;
```



```
 // 9.9
 virtual void unsubscribeObjectClassAttributesWithRegions
 (RTI_ObjectClassHandle const & theClass,
  RTI_AttributeHandleSetRegionHandleSetPairVector const
& theAttributeHandleSetRegionHandleSetPairVector)
   throw (RTI_ObjectClassNotDefined,
          RTI_AttributeNotDefined,
     RTI_InvalidRegion,
          RTI_RegionNotCreatedByThisFederate,
          RTI_FederateNotExecutionMember,
          RTI_SaveInProgress,
          RTI_RestoreInProgress,
          RTI_RTIinternalError) = 0;

 // 9.10
 virtual void subscribeInteractionClassWithRegions
 (RTI_InteractionClassHandle const & theClass,
  RTI_RegionHandleSet const & theRegionHandleSet,
  RTI_bool active = RTI_true)
   throw (RTI_InteractionClassNotDefined,RTI_InvalidRegion,
          RTI_RegionNotCreatedByThisFederate,
     RTI_InvalidRegionContext,
          RTI_FederateServiceInvocationsAreBeingReportedViaMOM,
          RTI_FederateNotExecutionMember,
          RTI_SaveInProgress,
          RTI_RestoreInProgress,
          RTI_RTIinternalError) = 0;

// 9.11
 virtual void unsubscribeInteractionClassWithRegions
 (RTI_InteractionClassHandle const & theClass,
  RTI_RegionHandleSet const & theRegionHandleSet)
   throw (RTI_InteractionClassNotDefined,
     RTI_InvalidRegion,
          RTI_RegionNotCreatedByThisFederate,
          RTI_FederateNotExecutionMember,
          RTI_SaveInProgress,
          RTI_RestoreInProgress,
          RTI_RTIinternalError) = 0;

 // 9.12
 virtual void sendInteractionWithRegions
 (RTI_InteractionClassHandleconst & theInteraction,
  RTI_ParameterHandleValueMap const & theParameterValues,
  RTI_RegionHandleSet const & theRegionHandleSet,
  RTI_UserSuppliedTag const & theUserSuppliedTag)
   throw (RTI_InteractionClassNotDefined,
          RTI_InteractionClassNotPublished,
          RTI_InteractionParameterNotDefined,
     RTI_InvalidRegion,
          RTI_RegionNotCreatedByThisFederate,
          RTI_InvalidRegionContext,
          RTI_FederateNotExecutionMember,
          RTI_SaveInProgress,
          RTI_RestoreInProgress,
          RTI_RTIinternalError) = 0;

 virtual RTI_auto_ptr< RTI_MessageRetractionHandle > sendInteractionWithRegions
 (RTI_InteractionClassHandleconst & theInteraction,
  RTI_ParameterHandleValueMap const & theParameterValues,
  RTI_RegionHandleSet const & theRegionHandleSet,
  RTI_UserSuppliedTag const & theUserSuppliedTag,
  RTI_LogicalTime const & theTime)
   throw (RTI_InteractionClassNotDefined,
          RTI_InteractionClassNotPublished,
          RTI_InteractionParameterNotDefined,
     RTI_InvalidRegion,
          RTI_RegionNotCreatedByThisFederate,
          RTI_InvalidRegionContext,
          RTI_InvalidLogicalTime,
          RTI_FederateNotExecutionMember,
          RTI_SaveInProgress,
```

```
          RTI_RestoreInProgress,
          RTI_RTIinternalError) = 0;

 // 9.13
 virtual void requestAttributeValueUpdateWithRegions
 (RTI_ObjectClassHandle const & theClass,
  RTI_AttributeHandleSetRegionHandleSetPairVector const & theSet,
  RTI_UserSuppliedTag const & theUserSuppliedTag)
   throw (RTI_ObjectClassNotDefined,
          RTI_AttributeNotDefined,
     RTI_InvalidRegion,
          RTI_RegionNotCreatedByThisFederate,
          RTI_InvalidRegionContext,
          RTI_FederateNotExecutionMember,
          RTI_SaveInProgress,
          RTI_RestoreInProgress,
          RTI_RTIinternalError) = 0;

 //////////////////////////
 // RTI Support Services //
 //////////////////////////

 // 10.2
 virtual RTI_ObjectClassHandle const & getObjectClassHandle
 (RTI_wstring const & theName)
   throw (RTI_NameNotFound,
          RTI_FederateNotExecutionMember,
          RTI_RTIinternalError) = 0;

 // 10.3
 virtual RTI_wstring const & getObjectClassName
 (RTI_ObjectClassHandle const & theHandle)
   throw (RTI_InvalidObjectClassHandle,
     RTI_FederateNotExecutionMember,
          RTI_RTIinternalError) = 0;

 // 10.4
 virtual RTI_AttributeHandle const & getAttributeHandle
 (RTI_ObjectClassHandle const & whichClass,
  RTI_wstring const & theAttributeName)
   throw (RTI_InvalidObjectClassHandle,
     RTI_NameNotFound,
          RTI_FederateNotExecutionMember,
          RTI_RTIinternalError) = 0;

 // 10.5
virtual RTI_wstring const & getAttributeName
(RTI_ObjectClassHandle const & whichClass,
RTI_AttributeHandle const & theHandle)
 throw (RTI_InvalidObjectClassHandle,
   RTI_InvalidAttributeHandle,
   RTI_AttributeNotDefined,
       RTI_FederateNotExecutionMember,
       RTI_RTIinternalError) = 0;

 // 10.6
 virtual RTI_InteractionClassHandle const & getInteractionClassHandle
 (RTI_wstring const & theName)
   throw (RTI_NameNotFound,
          RTI_FederateNotExecutionMember,
          RTI_RTIinternalError) = 0;

 // 10.7
 virtual RTI_wstring const & getInteractionClassName
 (RTI_InteractionClassHandle const & theHandle)
   throw (RTI_InvalidInteractionClassHandle,
     RTI_FederateNotExecutionMember,
          RTI_RTIinternalError) = 0;
```



```
// 10.8
virtual RTI_ParameterHandle const & getParameterHandle
(RTI_InteractionClassHandle const & whichClass,
 RTI_wstring const & theName)
   throw (RTI_InvalidInteractionClassHandle,
    RTI_NameNotFound,
         RTI_FederateNotExecutionMember,
         RTI_RTIinternalError) = 0;

// 10.9
virtual RTI_wstring const & getParameterName
(RTI_InteractionClassHandle const & whichClass,
 RTI_ParameterHandle const & theHandle)
  throw (RTI_InvalidInteractionClassHandle,
    RTI_InvalidParameterHandle,
    RTI_InteractionParameterNotDefined,
         RTI_FederateNotExecutionMember,
         RTI_RTIinternalError) = 0;

// 10.10
virtual RTI_ObjectInstanceHandle const & getObjectInstanceHandle
(RTI_wstring const & theName)
  throw (RTI_ObjectInstanceNotKnown,
         RTI_FederateNotExecutionMember,
         RTI_RTIinternalError) = 0;

// 10.11
virtual RTI_wstring const & getObjectInstanceName
(RTI_ObjectInstanceHandle const & theHandle)
  throw (RTI_ObjectInstanceNotKnown,
         RTI_FederateNotExecutionMember,
         RTI_RTIinternalError) = 0;

// 10.12
virtual RTI_DimensionHandle const & getDimensionHandle
(RTI_wstring const & theName)
  throw (RTI_NameNotFound,
         RTI_FederateNotExecutionMember,
         RTI_RTIinternalError) = 0;

// 10.13
virtual RTI_wstring const & getDimensionName
(RTI_DimensionHandle const & theHandle)
  throw (RTI_InvalidDimensionHandle,
    RTI_FederateNotExecutionMember,
         RTI_RTIinternalError) = 0;

// 10.14
virtual unsigned long getDimensionUpperBound
(RTI_DimensionHandle const & theHandle)
  throw (RTI_InvalidDimensionHandle,
         RTI_FederateNotExecutionMember,
         RTI_RTIinternalError) = 0;

// 10.15
virtual RTI_DimensionHandleSet const & getAvailableDimensionsForClassAttribute
(RTI_ObjectClassHandle const & theClass,
 RTI_AttributeHandle const & theHandle)
  throw (RTI_InvalidObjectClassHandle,
    RTI_InvalidAttributeHandle,
    RTI_AttributeNotDefined,
         RTI_FederateNotExecutionMember,
         RTI_RTIinternalError) = 0;

// 10.16
virtual RTI_ObjectClassHandle const & getKnownObjectClassHandle
(RTI_ObjectInstanceHandle const & theObject)
  throw (RTI_ObjectInstanceNotKnown,
         RTI_FederateNotExecutionMember,
         RTI_RTIinternalError) = 0;
```

```
// 10.17
virtual RTI_DimensionHandleSet const & getAvailableDimensionsForInteractionClass
(RTI_InteractionClassHandle const & theClass)
  throw (RTI_InvalidInteractionClassHandle,
    RTI_FederateNotExecutionMember,
         RTI_RTIinternalError) = 0;

// 10.18
virtual RTI_TransportationType const & getTransportationType
(RTI_wstring const & transportationName)
  throw (RTI_InvalidTransportationName,
         RTI_FederateNotExecutionMember,
         RTI_RTIinternalError) = 0;

// 10.19
virtual RTI_wstring const & getTransportationName
(RTI_TransportationType const & transportationType)
  throw (RTI_InvalidTransportationType,
         RTI_FederateNotExecutionMember,
         RTI_RTIinternalError) = 0;

// 10.20
virtual RTI_OrderType const & getOrderType
(RTI_wstring const & orderName)
  throw (RTI_InvalidOrderName,
         RTI_FederateNotExecutionMember,
         RTI_RTIinternalError) = 0;

// 10.21
virtual RTI_wstring const & getOrderName
(RTI_OrderType const & orderType)
  throw (RTI_InvalidOrderType,
         RTI_FederateNotExecutionMember,
         RTI_RTIinternalError) = 0;

// 10.22
virtual void enableObjectClassRelevanceAdvisorySwitch ()
  throw (RTI_ObjectClassRelevanceAdvisorySwitchIsOn,
         RTI_FederateNotExecutionMember,
         RTI_SaveInProgress,
         RTI_RestoreInProgress,
         RTI_RTIinternalError) = 0;

// 10.23
virtual void disableObjectClassRelevanceAdvisorySwitch ()
  throw (RTI_ObjectClassRelevanceAdvisorySwitchIsOff,
         RTI_FederateNotExecutionMember,
         RTI_SaveInProgress,
         RTI_RestoreInProgress,
         RTI_RTIinternalError) = 0;

// 10.24
virtual void enableAttributeRelevanceAdvisorySwitch ()
  throw (RTI_AttributeRelevanceAdvisorySwitchIsOn,
         RTI_FederateNotExecutionMember,
         RTI_SaveInProgress,
         RTI_RestoreInProgress,
         RTI_RTIinternalError) = 0;

// 10.25
virtual void disableAttributeRelevanceAdvisorySwitch ()
  throw (RTI_AttributeRelevanceAdvisorySwitchIsOff,
         RTI_FederateNotExecutionMember,
         RTI_SaveInProgress,
         RTI_RestoreInProgress,
         RTI_RTIinternalError) = 0;

// 10.26
virtual void enableAttributeScopeAdvisorySwitch ()
  throw (RTI_AttributeScopeAdvisorySwitchIsOn,
         RTI_FederateNotExecutionMember,
         RTI_SaveInProgress,
```



```
        RTI_RestoreInProgress,
        RTI_RTIinternalError) = 0;

// 10.27
virtual void disableAttributeScopeAdvisorySwitch ()
  throw (RTI_AttributeScopeAdvisorySwitchIsOff,
        RTI_FederateNotExecutionMember,
        RTI_SaveInProgress,
        RTI_RestoreInProgress,
        RTI_RTIinternalError) = 0;

// 10.28
virtual void enableInteractionRelevanceAdvisorySwitch ()
  throw (RTI_InteractionRelevanceAdvisorySwitchIsOn,
        RTI_FederateNotExecutionMember,
        RTI_SaveInProgress,
        RTI_RestoreInProgress,
        RTI_RTIinternalError) = 0;

// 10.29
virtual void disableInteractionRelevanceAdvisorySwitch ()
  throw (RTI_InteractionRelevanceAdvisorySwitchIsOff,
        RTI_FederateNotExecutionMember,
        RTI_SaveInProgress,
        RTI_RestoreInProgress,
        RTI_RTIinternalError) = 0;

// 10.30
virtual
RTI_DimensionHandleSet const & getDimensionHandleSet
(RTI_RegionHandle const & theRegionHandle)
  throw (RTI_InvalidRegion,
    RTI_FederateNotExecutionMember,
        RTI_SaveInProgress,
        RTI_RestoreInProgress,
        RTI_RTIinternalError) = 0;

// 10.31
virtual
RTI_RangeBounds const & getRangeBounds
(RTI_RegionHandle const & theRegionHandle,
 RTI_DimensionHandle const & theDimensionHandle)
  throw (RTI_InvalidRegion,
    RTI_RegionDoesNotContainSpecifiedDimension,
        RTI_FederateNotExecutionMember,
        RTI_SaveInProgress,
        RTI_RestoreInProgress,
        RTI_RTIinternalError) = 0;

// 10.32
virtual void setRangeBounds
(RTI_RegionHandle const & theRegionHandle,
 RTI_DimensionHandle const & theDimensionHandle,
 RTI_RangeBounds const & theRangeBounds)
  throw (RTI_InvalidRegion,
    RTI_RegionNotCreatedByThisFederate,
        RTI_RegionDoesNotContainSpecifiedDimension,
        RTI_InvalidRangeBounds,
        RTI_FederateNotExecutionMember,
        RTI_SaveInProgress,
        RTI_RestoreInProgress,
        RTI_RTIinternalError) = 0;

// 10.33
virtual unsigned long normalizeFederateHandle
(RTI_FederateHandle const & theFederateHandle)
  throw (RTI_FederateHandle,
    RTI_FederateNotExecutionMember,
        RTI_RTIinternalError) = 0;

// 10.34
virtual unsigned long normalizeServiceGroup
```

```
  (RTI_ServiceGroupIndicator const & theServiceGroup)
    throw (RTI_FederateNotExecutionMember,
           RTI_RTIinternalError) = 0;

  // 10.37
  virtual RTI_bool evokeCallback(double approximateMinimumTimeInSeconds)
    throw (RTI_FederateNotExecutionMember,
           RTI_RTIinternalError) = 0;

  // 10.38
  virtual RTI_bool evokeMultipleCallbacks(double approximateMinimumTimeInSeconds,
double approximateMaximumTimeInSeconds)
    throw (RTI_FederateNotExecutionMember,
           RTI_RTIinternalError) = 0;

  // 10.39
   virtual void enableCallbacks ()
    throw (RTI_FederateNotExecutionMember,
           RTI_SaveInProgress,
           RTI_RestoreInProgress,
           RTI_RTIinternalError) = 0;

  // 10.40
  virtual void disableCallbacks ()
    throw (RTI_FederateNotExecutionMember,
           RTI_SaveInProgress,
           RTI_RestoreInProgress,
           RTI_RTIinternalError) = 0;

};

#endif // RTI_RTIambassador_h


/**************************************************************************
 IEEE 1516.1 High Level Architecture Interface Specification C++ API
 File: RTI_FederateAmbassador.h
**************************************************************************/

// This is a pure abstract interface that must be implemented by the
// federate to receive callbacks from the RTI.

#ifndef RTI_FederateAmbassador_h
#define RTI_FederateAmbassador_h

//
// These forward declarations significantly decrease compilation time when only
// including this file
//

class RTI_AttributeAcquisitionWasNotCanceled;
class RTI_AttributeAcquisitionWasNotRequested;
class RTI_AttributeAlreadyOwned;
class RTI_AttributeDivestitureWasNotRequested;
class RTI_AttributeNotOwned;
class RTI_AttributeNotPublished;
class RTI_AttributeNotRecognized;
class RTI_AttributeNotRecognized;
class RTI_AttributeNotSubscribed;
class RTI_CouldNotDiscover;
class RTI_CouldNotInitiateRestore;
class RTI_FederateInternalError;
class RTI_InteractionClassNotPublished;
class RTI_InteractionClassNotRecognized;
class RTI_InteractionClassNotSubscribed;
class RTI_InteractionParameterNotRecognized;
class RTI_InvalidLogicalTime;
class RTI_JoinedFederateIsNotInTimeAdvancingState;
class RTI_LogicalTime;
class RTI_NoRequestToEnableTimeConstrainedWasPending;
class RTI_NoRequestToEnableTimeRegulationWasPending;
class RTI_ObjectClassNotKnown;
```



```
class RTI_ObjectClassNotPublished;
class RTI_ObjectInstanceNotKnown;
class RTI_OrderType;
class RTI_SpecifiedSaveLabelDoesNotExist;
class RTI_TransportationType;
class RTI_UnableToPerformSave;
class RTI_UnknownName;

#include <RTI_SpecificConfig.h>
#include <RTI_memory>
#include <RTI_SynchronizationFailureReason.h>
#include <RTI_RestoreFailureReason.h>
#include <RTI_SaveFailureReason.h>
#include <RTI_SaveStatus.h>
#include <RTI_RestoreStatus.h>
#include <RTI_Typedefs.h>

class RTI_FederateAmbassador
{
public:
 RTI_FederateAmbassador()
   throw (RTI_FederateInternalError);

 virtual
 ~RTI_FederateAmbassador()
   throw ();

 // 4.7
 virtual
 void
 synchronizationPointRegistrationSucceeded(RTI_wstring const & label)
   throw (RTI_FederateInternalError) = 0;

 virtual
 void
 synchronizationPointRegistrationFailed(RTI_wstring const & label,
                RTI_SynchronizationFailureReason theReason)
   throw (RTI_FederateInternalError) = 0;

 // 4.8
 virtual
 void
 announceSynchronizationPoint(RTI_wstringconst & label,
                              RTI_UserSuppliedTag const & theUserSuppliedTag)
   throw (RTI_FederateInternalError) = 0;

 // 4.10
 virtual
 void
 federationSynchronized(RTI_wstring const & label)
   throw (RTI_FederateInternalError) = 0;

 // 4.12
 virtual
 void
 initiateFederateSave(RTI_wstring const & label)
   throw (RTI_UnableToPerformSave,
          RTI_FederateInternalError) = 0;

 virtual
 void
 initiateFederateSave(RTI_wstring const & label,
                      RTI_LogicalTime const & theTime)
   throw (RTI_UnableToPerformSave,
     RTI_InvalidLogicalTime,
          RTI_FederateInternalError) = 0;

 // 4.15
 virtual
 void
 federationSaved()
   throw (RTI_FederateInternalError) = 0;
```

```
  virtual
  void
  federationNotSaved(RTI_SaveFailureReason theSaveFailureReason)
    throw (RTI_FederateInternalError) = 0;

  // 4.17
  virtual
  void
  federationSaveStatusResponse(RTI_auto_ptr<
RTI_FederateHandleSaveStatusPairVector > theFederateStatusVector)
    throw (RTI_FederateInternalError) = 0;

  // 4.19
  virtual
  void
  requestFederationRestoreSucceeded(RTI_wstring const & label)
    throw (RTI_FederateInternalError) = 0;

  virtual
  void
  requestFederationRestoreFailed()
    throw (RTI_FederateInternalError) = 0;

  // 4.20
  virtual
  void
  federationRestoreBegun()
    throw (RTI_FederateInternalError) = 0;

  // 4.21
  virtual
  void
  initiateFederateRestore(RTI_wstring        const & label,
                          RTI_FederateHandle const & handle)
    throw (RTI_SpecifiedSaveLabelDoesNotExist,
           RTI_CouldNotInitiateRestore,
           RTI_FederateInternalError) = 0;

  // 4.23
  virtual
  void
  federationRestored()
    throw (RTI_FederateInternalError) = 0;

  virtual
  void
  federationNotRestored(RTI_RestoreFailureReason theRestoreFailureReason)
    throw (RTI_FederateInternalError) = 0;

  // 4.25
  virtual
  void
  federationRestoreStatusResponse(RTI_auto_ptr<
RTI_FederateHandleRestoreStatusPairVector > theFederateStatusVector)
    throw (RTI_FederateInternalError) = 0;
  ////////////////////////////////////
  // Declaration Management Services //
  ////////////////////////////////////

  // 5.10
  virtual
  void
  startRegistrationForObjectClass(RTI_ObjectClassHandle const & theClass)
    throw (RTI_ObjectClassNotPublished,
           RTI_FederateInternalError) = 0;

  // 5.11
  virtual
  void
  stopRegistrationForObjectClass(RTI_ObjectClassHandle const & theClass)
    throw (RTI_ObjectClassNotPublished,
           RTI_FederateInternalError) = 0;
```



```
 // 5.12
 virtual
 void
 turnInteractionsOn(RTI_InteractionClassHandle const & theHandle)
   throw (RTI_InteractionClassNotPublished,
          RTI_FederateInternalError) = 0;

 // 5.13
 virtual
 void
 turnInteractionsOff(RTI_InteractionClassHandle const & theHandle)
   throw (RTI_InteractionClassNotPublished,
          RTI_FederateInternalError) = 0;
 ////////////////////////////////
 // Object Management Services //
 ////////////////////////////////

 // 6.3
 virtual
 void
 objectInstanceNameReservationSucceeded(RTI_wstring const &
theObjectInstanceName)
   throw (RTI_UnknownName,
          RTI_FederateInternalError) = 0;

 virtual
 void
 objectInstanceNameReservationFailed(RTI_wstring const & theObjectInstanceName)
   throw (RTI_UnknownName,
          RTI_FederateInternalError) = 0;

 // 6.5

 virtual
 void
 discoverObjectInstance(RTI_ObjectInstanceHandleconst & theObject,
                        RTI_ObjectClassHandle     const & theObjectClass,
                        RTI_wstring               const & theObjectInstanceName)
   throw (RTI_CouldNotDiscover,
          RTI_ObjectClassNotKnown,
          RTI_FederateInternalError) = 0;

 // 6.7
 virtual
 void
 reflectAttributeValues
 (RTI_ObjectInstanceHandle                   const & theObject,
  RTI_auto_ptr< RTI_AttributeHandleValueMap > const & theAttributeValues,
  RTI_UserSuppliedTag                        const & theUserSuppliedTag,
  RTI_OrderType                              const & sentOrder,
  RTI_TransportationType                     const & theType)
   throw (RTI_ObjectInstanceNotKnown,
          RTI_AttributeNotRecognized,
          RTI_AttributeNotSubscribed,
          RTI_FederateInternalError) = 0;

 virtual
 void
 reflectAttributeValues
 (RTI_ObjectInstanceHandle                   const & theObject,
  RTI_auto_ptr< RTI_AttributeHandleValueMap > const & theAttributeValues,
  RTI_UserSuppliedTag                        const & theUserSuppliedTag,
  RTI_OrderType                              const & sentOrder,
  RTI_TransportationType                     const & theType,
  RTI_RegionHandleSet                        const & theSentRegionHandleSet)
   throw (RTI_ObjectInstanceNotKnown,
          RTI_AttributeNotRecognized,
          RTI_AttributeNotSubscribed,
          RTI_FederateInternalError) = 0;

 virtual
 void
```

```
reflectAttributeValues
(RTI_ObjectInstanceHandle                   const & theObject,
 RTI_auto_ptr< RTI_AttributeHandleValueMap > theAttributeValues,
 RTI_UserSuppliedTag                        const & theUserSuppliedTag,
 RTI_OrderType                              const & sentOrder,
 RTI_TransportationType                     const & theType,
 RTI_LogicalTime                            const & theTime,
 RTI_OrderType                              const & receivedOrder)
  throw (RTI_ObjectInstanceNotKnown,
         RTI_AttributeNotRecognized,
         RTI_AttributeNotSubscribed,
         RTI_FederateInternalError) = 0;

virtual
void
reflectAttributeValues
(RTI_ObjectInstanceHandle                   const & theObject,
 RTI_auto_ptr< RTI_AttributeHandleValueMap > theAttributeValues,
 RTI_UserSuppliedTag                        const & theUserSuppliedTag,
 RTI_OrderType                              const & sentOrder,
 RTI_TransportationType                     const & theType,
 RTI_LogicalTime                            const & theTime,
 RTI_OrderType                              const & receivedOrder,
 RTI_RegionHandleSet                        const & theSentRegionHandleSet)
  throw (RTI_ObjectInstanceNotKnown,
         RTI_AttributeNotRecognized,
         RTI_AttributeNotSubscribed,
         RTI_FederateInternalError) = 0;

virtual
void
reflectAttributeValues
(RTI_ObjectInstanceHandle                   const & theObject,
 RTI_auto_ptr< RTI_AttributeHandleValueMap > theAttributeValues,
 RTI_UserSuppliedTag                        const & theUserSuppliedTag,
 RTI_OrderType                              const & sentOrder,
 RTI_TransportationType                     const & theType,
 RTI_LogicalTime                            const & theTime,
 RTI_OrderType                              const & receivedOrder,
 RTI_MessageRetractionHandle                const & theHandle)
  throw (RTI_ObjectInstanceNotKnown,
         RTI_AttributeNotRecognized,
         RTI_AttributeNotSubscribed,
    RTI_InvalidLogicalTime,
         RTI_FederateInternalError) = 0;

virtual
void
reflectAttributeValues
(RTI_ObjectInstanceHandle                   const & theObject,
 RTI_auto_ptr< RTI_AttributeHandleValueMap > theAttributeValues,
 RTI_UserSuppliedTag                        const & theUserSuppliedTag,
 RTI_OrderType                              const & sentOrder,
 RTI_TransportationType                     const & theType,
 RTI_LogicalTime                            const & theTime,
 RTI_OrderType                              const & receivedOrder,
 RTI_MessageRetractionHandle                const & theHandle,
 RTI_RegionHandleSet                        const & theSentRegionHandleSet)
  throw (RTI_ObjectInstanceNotKnown,
         RTI_AttributeNotRecognized,
         RTI_AttributeNotSubscribed,
    RTI_InvalidLogicalTime,
         RTI_FederateInternalError) = 0;

// 6.9
virtual
void
receiveInteraction
(RTI_InteractionClassHandle                 const & theInteraction,
 RTI_auto_ptr< RTI_ParameterHandleValueMap > theParameterValues,
 RTI_UserSuppliedTag                        const & theUserSuppliedTag,
 RTI_OrderType                              const & sentOrder,
```



```
  RTI_TransportationType                      const & theType)
    throw (RTI_InteractionClassNotRecognized,
           RTI_InteractionParameterNotRecognized,
           RTI_InteractionClassNotSubscribed,
           RTI_FederateInternalError) = 0;

 virtual
 void
 receiveInteraction
 (RTI_InteractionClassHandle                 const & theInteraction,
  RTI_auto_ptr< RTI_ParameterHandleValueMap >  theParameterValues,
  RTI_UserSuppliedTag                        const & theUserSuppliedTag,
  RTI_OrderType                              const & sentOrder,
  RTI_TransportationType                     const & theType,
  RTI_RegionHandleSet                        const & theSentRegionHandleSet)
    throw (RTI_InteractionClassNotRecognized,
           RTI_InteractionParameterNotRecognized,
           RTI_InteractionClassNotSubscribed,
           RTI_FederateInternalError) = 0;

 virtual
 void
 receiveInteraction
 (RTI_InteractionClassHandle                 const & theInteraction,
  RTI_auto_ptr< RTI_ParameterHandleValueMap >  theParameterValues,
  RTI_UserSuppliedTag                        const & theUserSuppliedTag,
  RTI_OrderType                              const & sentOrder,
  RTI_TransportationType                     const & theType,
  RTI_LogicalTime                            const & theTime,
RTI_OrderType                                   const & receivedOrder)
    throw (RTI_InteractionClassNotRecognized,
           RTI_InteractionParameterNotRecognized,
           RTI_InteractionClassNotSubscribed,
           RTI_FederateInternalError) = 0;

 virtual
 void
 receiveInteraction
 (RTI_InteractionClassHandle                 const & theInteraction,
  RTI_auto_ptr< RTI_ParameterHandleValueMap >  theParameterValues,
  RTI_UserSuppliedTag                        const & theUserSuppliedTag,
  RTI_OrderType                              const & sentOrder,
  RTI_TransportationType                     const & theType,
  RTI_LogicalTime                            const & theTime,
  RTI_OrderType                              const & receivedOrder,
  RTI_RegionHandleSet                        const & theSentRegionHandleSet)
    throw (RTI_InteractionClassNotRecognized,
           RTI_InteractionParameterNotRecognized,
           RTI_InteractionClassNotSubscribed,
           RTI_FederateInternalError) = 0;

 virtual
 void
 receiveInteraction
 (RTI_InteractionClassHandle                 const & theInteraction,
  RTI_auto_ptr< RTI_ParameterHandleValueMap >  theParameterValues,
  RTI_UserSuppliedTag                        const & theUserSuppliedTag,
  RTI_OrderType                              const & sentOrder,
  RTI_TransportationType                     const & theType,
  RTI_LogicalTime                            const & theTime,
  RTI_OrderType                              const & receivedOrder,
  RTI_MessageRetractionHandle                const & theHandle)
    throw (RTI_InteractionClassNotRecognized,
           RTI_InteractionParameterNotRecognized,
           RTI_InteractionClassNotSubscribed,
      RTI_InvalidLogicalTime,
           RTI_FederateInternalError) = 0;

 virtual
 void
 receiveInteraction
 (RTI_InteractionClassHandle                 const & theInteraction,
```

```
 RTI_auto_ptr< RTI_ParameterHandleValueMap >  theParameterValues,
 RTI_UserSuppliedTag                         const & theUserSuppliedTag,
 RTI_OrderType                               const & sentOrder,
 RTI_TransportationType                      const & theType,
 RTI_LogicalTime                             const & theTime,
 RTI_OrderType                               const & receivedOrder,
 RTI_MessageRetractionHandle                 const & theHandle,
 RTI_RegionHandleSet                         const & theSentRegionHandleSet)
  throw (RTI_InteractionClassNotRecognized,
         RTI_InteractionParameterNotRecognized,
         RTI_InteractionClassNotSubscribed,
    RTI_InvalidLogicalTime,
         RTI_FederateInternalError) = 0;

// 6.11
virtual
void
removeObjectInstance(RTI_ObjectInstanceHandleconst & theObject,
                     RTI_UserSuppliedTag       const & theUserSuppliedTag,
                     RTI_OrderType             const & sentOrder)
  throw (RTI_ObjectInstanceNotKnown,
         RTI_FederateInternalError) = 0;

virtual
void
removeObjectInstance(RTI_ObjectInstanceHandleconst & theObject,
                     RTI_UserSuppliedTag       const & theUserSuppliedTag,
                     RTI_OrderType             const & sentOrder,
                     RTI_LogicalTime           const & theTime,
                      RTI_OrderType             const & receivedOrder)
  throw (RTI_ObjectInstanceNotKnown,
         RTI_FederateInternalError) = 0;

virtual
void
removeObjectInstance(RTI_ObjectInstanceHandle const & theObject,
                     RTI_UserSuppliedTag       const & theUserSuppliedTag,
                     RTI_OrderType             const & sentOrder,
                     RTI_LogicalTime           const & theTime,
                     RTI_OrderType             const & receivedOrder,
                     RTI_MessageRetractionHandle const & theHandle)
  throw (RTI_ObjectInstanceNotKnown,
         RTI_InvalidLogicalTime,
         RTI_FederateInternalError) = 0;

// 6.15
virtual
void
attributesInScope
(RTI_ObjectInstanceHandle              const & theObject,
 RTI_auto_ptr< RTI_AttributeHandleSet >  theAttributes)
  throw (RTI_ObjectInstanceNotKnown,
         RTI_AttributeNotRecognized,
         RTI_AttributeNotSubscribed,
         RTI_FederateInternalError) = 0;

// 6.16
virtual
void
attributesOutOfScope
(RTI_ObjectInstanceHandle              const & theObject,
 RTI_auto_ptr< RTI_AttributeHandleSet >  theAttributes)
  throw (RTI_ObjectInstanceNotKnown,
         RTI_AttributeNotRecognized,
         RTI_AttributeNotSubscribed,
         RTI_FederateInternalError) = 0;

// 6.18
virtual
void
provideAttributeValueUpdate
(RTI_ObjectInstanceHandle              const & theObject,
```



```
   RTI_auto_ptr< RTI_AttributeHandleSet >  theAttributes,
   RTI_UserSuppliedTag                     const & theUserSuppliedTag)
    throw (RTI_ObjectInstanceNotKnown,
           RTI_AttributeNotRecognized,
           RTI_AttributeNotOwned,
           RTI_FederateInternalError) = 0;

 // 6.19
 virtual
 void
 turnUpdatesOnForObjectInstance
 (RTI_ObjectInstanceHandle            const & theObject,
  RTI_auto_ptr< RTI_AttributeHandleSet > theAttributes)
    throw (RTI_ObjectInstanceNotKnown,
           RTI_AttributeNotRecognized,
           RTI_AttributeNotOwned,
           RTI_FederateInternalError) = 0;

 // 6.20
 virtual
 void
 turnUpdatesOffForObjectInstance
 (RTI_ObjectInstanceHandle             const & theObject,
  RTI_auto_ptr< RTI_AttributeHandleSet >  theAttributes)
    throw (RTI_ObjectInstanceNotKnown,
           RTI_AttributeNotRecognized,
           RTI_AttributeNotOwned,
           RTI_FederateInternalError) = 0;

 ////////////////////////////////////
 // Ownership Management Services //
 ////////////////////////////////////

 // 7.4
 virtual
 void
 requestAttributeOwnershipAssumption
 (RTI_ObjectInstanceHandle           const & theObject,
RTI_auto_ptr< RTI_AttributeHandleSet >  offeredAttributes,
RTI_UserSuppliedTag                  const & theUserSuppliedTag)
    throw (RTI_ObjectInstanceNotKnown,
           RTI_AttributeNotRecognized,
           RTI_AttributeAlreadyOwned,
           RTI_AttributeNotPublished,
           RTI_FederateInternalError) = 0;

 // 7.5
 virtual
 void
 requestDivestitureConfirmation
 (RTI_ObjectInstanceHandle             const & theObject,
  RTI_auto_ptr< RTI_AttributeHandleSet >  releasedAttributes)
    throw (RTI_ObjectInstanceNotKnown,
           RTI_AttributeNotRecognized,
           RTI_AttributeNotOwned,
           RTI_AttributeDivestitureWasNotRequested,
           RTI_FederateInternalError) = 0;

 // 7.7
 virtual
 void
 attributeOwnershipAcquisitionNotification
 (RTI_ObjectInstanceHandle                  const & theObject,
  RTI_auto_ptr< RTI_AttributeHandleSet >    securedAttributes,
  RTI_UserSuppliedTag                       const & theUserSuppliedTag)
    throw (RTI_ObjectInstanceNotKnown,
           RTI_AttributeNotRecognized,
           RTI_AttributeAcquisitionWasNotRequested,
           RTI_AttributeAlreadyOwned,
           RTI_AttributeNotPublished,
           RTI_FederateInternalError) = 0;
```

```
// 7.10
virtual
void
attributeOwnershipUnavailable
(RTI_ObjectInstanceHandle             const & theObject,
 RTI_auto_ptr< RTI_AttributeHandleSet >  theAttributes)
  throw (RTI_ObjectInstanceNotKnown,
         RTI_AttributeNotRecognized,
         RTI_AttributeAlreadyOwned,
         RTI_AttributeAcquisitionWasNotRequested,
         RTI_FederateInternalError) = 0;

// 7.11
virtual
void
requestAttributeOwnershipRelease
(RTI_ObjectInstanceHandle               const & theObject,
 RTI_auto_ptr< RTI_AttributeHandleSet > candidateAttributes,
 RTI_UserSuppliedTag                    const & theUserSuppliedTag)
  throw (RTI_ObjectInstanceNotKnown,
         RTI_AttributeNotRecognized,
         RTI_AttributeNotOwned,
         RTI_FederateInternalError) = 0;

// 7.15
virtual
void
confirmAttributeOwnershipAcquisitionCancellation
(RTI_ObjectInstanceHandle                  const & theObject,
 RTI_auto_ptr< RTI_AttributeHandleSet >    theAttributes)
  throw (RTI_ObjectInstanceNotKnown,
         RTI_AttributeNotRecognized,
         RTI_AttributeAlreadyOwned,
         RTI_AttributeAcquisitionWasNotCanceled,
         RTI_FederateInternalError) = 0;

// 7.17
virtual
void
informAttributeOwnership(RTI_ObjectInstanceHandle const & theObject,
                         RTI_AttributeHandle        const & theAttribute,
                         RTI_FederateHandle         const & theOwner)
  throw (RTI_ObjectInstanceNotKnown,
         RTI_AttributeNotRecognized,
         RTI_FederateInternalError) = 0;

virtual
void
attributeIsNotOwned(RTI_ObjectInstanceHandle const & theObject,
                    RTI_AttributeHandle        const & theAttribute)
  throw (RTI_ObjectInstanceNotKnown,
         RTI_AttributeNotRecognized,
         RTI_FederateInternalError) = 0;

virtual
 void
 attributeIsOwnedByRTI(RTI_ObjectInstanceHandle const & theObject,
                       RTI_AttributeHandle        const & theAttribute)
  throw (RTI_ObjectInstanceNotKnown,
         RTI_AttributeNotRecognized,
         RTI_FederateInternalError) = 0;

//////////////////////////////
// Time Management Services //
//////////////////////////////

// 8.3
virtual
void
timeRegulationEnabled(RTI_LogicalTime const & theFederateTime)
  throw (RTI_InvalidLogicalTime,
         RTI_NoRequestToEnableTimeRegulationWasPending,
```



```
          RTI_FederateInternalError) = 0;

 // 8.6
 virtual
 void
 timeConstrainedEnabled(RTI_LogicalTime const & theFederateTime)
   throw (RTI_InvalidLogicalTime,
          RTI_NoRequestToEnableTimeConstrainedWasPending,
          RTI_FederateInternalError) = 0;

 // 8.13
 virtual
 void
 timeAdvanceGrant(RTI_LogicalTime const & theTime)
   throw (RTI_InvalidLogicalTime,
          RTI_JoinedFederateIsNotInTimeAdvancingState,
          RTI_FederateInternalError) = 0;

 // 8.22
 virtual
 void
 requestRetraction(RTI_MessageRetractionHandle const & theHandle)
   throw (RTI_FederateInternalError) = 0;

};

#endif // RTI_FederateAmbassador_h


/*************************************************************************
IEEE 1516.1 High Level Architecture Interface Specification C++ API
File: RTI_1516.h
*************************************************************************/

//
// This file is simply a convenience provided for those developers that would
// like to include everything all at once
//

#ifndef RTI_1516_h
#define RTI_1516_h

// These file include declarations/definitions for ISO 14882 standard C++
// classes, renamed for portability.
#include <RTI_string>
#include <RTI_set>
#include <RTI_map>
#include <RTI_vector>
#include <RTI_memory>

// This file contains platform specific configuration info.
#include <RTI_SpecificConfig.h>

#include <RTI_bool.h>

// This file contains standard RTI type declarations/definitions.
#include <RTI_exception.h>
#include <RTI_Handle.h>
#include <RTI_Value.h>
#include <RTI_ResignAction.h>
#include <RTI_TransportationType.h>
#include <RTI_OrderType.h>
#include <RTI_SaveFailureReason.h>
#include <RTI_SaveStatus.h>
#include <RTI_SynchronizationFailureReason.h>
#include <RTI_RestoreStatus.h>
#include <RTI_RestoreFailureReason.h>
#include <RTI_ServiceGroupIndicator.h>
#include <RTI_RangeBounds.h>

// This file has RTI implementation specific declarations/definitions.
#include <RTI_SpecificTypedefs.h>
```

```
// This file contains standard RTI type declarations/definitions which depend on
// RTI implementation specific declarations/definitions.
#include <RTI_Typedefs.h>
#include <RTI_LogicalTime.h>
#include <RTI_LogicalTimeFactory.h>
#include <RTI_LogicalTimeInterval.h>
#include <RTI_LogicalTimeIntervalFactory.h>

static char const * const HLA_VERSION = "1516.1.5";

#include <RTI_FederateAmbassador.h>
#include <RTI_RTIambassador.h>

// This file provides RTI implementation specific decalarations and definitions
// that need to follow the other header files.
#include <RTI_SpecificPostamble.h>

#endif // RTI_1516_h


/**************************************************************************
IEEE 1516.1 High Level Architecture Interface Specification C++ API
File: RTI_EncodedLogicalTime.h
**************************************************************************/

#ifndef RTI_EncodedLogicalTime_h
#define RTI_EncodedLogicalTime_h

#include <RTI_SpecificConfig.h>

class RTI_EncodedLogicalTime
{
public:

  virtual ~RTI_EncodedLogicalTime() throw () {}

  virtual voidconst * data() const = 0;
  virtual size_tsize()     const = 0;
};

#endif // RTI_EncodedLogicalTime_h



/**************************************************************************
IEEE 1516.1 High Level Architecture Interface Specification C++ API
File: RTI_EncodedLogicalTimeInterval.h
**************************************************************************/

#ifndef RTI_EncodedLogicalTimeInterval_h
#define RTI_EncodedLogicalTimeInterval_h

#include <RTI_SpecificConfig.h>

class RTI_EncodedLogicalTimeInterval
{
public:
  virtual ~RTI_EncodedLogicalTimeInterval() throw () {}

  virtual voidconst * data() const = 0;
  virtual size_tsize()     const = 0;
};

#endif // RTI_EncodedLogicalTimeInterval_h
```



```
/*************************************************************************
IEEE 1516.1 High Level Architecture Interface Specification C++ API
File: RTI_Handle.h
*************************************************************************/
#ifndef RTI_Handle_h
#define RTI_Handle_h

class RTI_bool;

#include <RTI_SpecificConfig.h>
#include <RTI_string>

// The RTIhandle class is used to provide the common interface to the different
// RTI handle types used in the API. This interface includes a constructor,
// assignment, equality, inequality, and less than operators. The encode method
// returns a type safe EncodedHandleClass instance that can be used to exchange
// handles between federates as attributes and parameters. The constructor takes
// a EncodedHandleClass which enables the type safe creation of a RTIhandle from
// encoded data passed to a federate. The template parameter class
// ImplementationSpecificHandleClass provides RTI implementations the ability
// to customize a private class member for their particular implementation.The
// int template type parameter provides the ability to support strong typing.

template<class ImplementationSpecificHandleClass,
         class EncodedHandleClass,
         class ImplementationSpecificHandleClassFriend,
         int   i>
class RTI_EXPORT RTI_Handle
{
 public:
  explicit
  RTI_Handle(EncodedHandleClass encodedHandle);

  ~RTI_Handle()
    throw();

  RTI_Handle(RTI_Handle const & rhs);

  RTI_Handle &
  operator=(RTI_Handle const & rhs);

  RTI_bool
  operator==(RTI_Handle const & rhs) const;

  RTI_bool
  operator!=(RTI_Handle const & rhs) const;

  RTI_bool
  operator< (RTI_Handle const & rhs) const;

  EncodedHandleClass
  encode() const;

  RTI_wstring const
  toString() const;

 private:
  ImplementationSpecificHandleClass _impl;

  //
  // This class is the only class which can construct an RTI_Handle
  //
  friend ImplementationSpecificHandleClassFriend;
  RTI_Handle(ImplementationSpecificHandleClass const & impl);
};

#ifdef   RTI_USE_INLINE
#include "RTI_Handle.i"
#endif // RTI_USE_INLINE
#ifdef   RTI_TEMPLATES_REQUIRE_SOURCE
#include "RTI_Handle.cpp"
#endif // RTI_TEMPLATES_REQUIRE_SOURCE

#endif // RTI_Handle_h
```

```
/*************************************************************************
IEEE 1516.1 High Level Architecture Interface Specification C++ API
File: RTI_LogicalTime.h
*************************************************************************/

#ifndef RTI_LogicalTime_h
#define RTI_LogicalTime_h

class RTI_LogicalTimeInterval;

// The classes associated with logical time allow a federation to provide their
// own representation for logical time and logical time interval. The federation
// is responsible to inherit from the abstract classes declared below. The
// encoded time classes are used to hold the arbitrary bit representation of the
// logical time and logical time intervals.

#include <RTI_memory>
class RTI_EncodedLogicalTime;

class RTI_EXPORT RTI_LogicalTime
{
 public:
  virtual
  ~RTI_LogicalTime()
    throw ();

  virtual
  void
  setInitial() = 0;

  virtual
  RTI_bool
  isInitial() = 0;

  virtual
  void
  setFinal() = 0;

  virtual
  RTI_bool
  isFinal() = 0;

  virtual
  void
  setTo(RTI_LogicalTime const & value)
    throw (RTI_InvalidLogicalTime) = 0;

  virtual
  void
  increaseBy(RTI_LogicalTimeInterval const & addend)
    throw (RTI_IllegalTimeArithmetic, RTI_InvalidLogicalTimeInterval) = 0;

  virtual
  void
  decreaseBy(RTI_LogicalTimeInterval const & subtrahend)
    throw (RTI_IllegalTimeArithmetic, RTI_InvalidLogicalTimeInterval) = 0;

  virtual
  RTI_auto_ptr< RTI_LogicalTimeInterval >
  subtract(RTI_LogicalTime const & subtrahend) const
    throw (RTI_InvalidLogicalTime) = 0;

  virtual
  RTI_bool
  isGreaterThan(RTI_LogicalTime const & value) const
    throw (RTI_InvalidLogicalTime) = 0;

  virtual
  RTI_bool
  isLessThan(RTI_LogicalTime const & value) const
    throw (RTI_InvalidLogicalTime) = 0;
```



```
  virtual
  RTI_bool
  isEqualTo(RTI_LogicalTime const & value) const
    throw (RTI_InvalidLogicalTime) = 0;

  virtual
  RTI_bool
  isGreaterThanOrEqualTo(RTI_LogicalTime const & value) const
    throw (RTI_InvalidLogicalTime) = 0;

  virtual
  RTI_bool
  isLessThanOrEqualTo(RTI_LogicalTime const & value) const
    throw (RTI_InvalidLogicalTime) = 0;

  virtual
  RTI_auto_ptr< RTI_EncodedLogicalTime >
  encode() const = 0;

  virtual
  RTI_wstring
  toString() const = 0;
};

#endif // RTI_LogicalTime_h


/**********************************************************************
IEEE 1516.1 High Level Architecture Interface Specification C++ API
File: RTI_LogicalTimeFactory.h
**********************************************************************/

#ifndef RTI_LogicalTimeFactory_h
#define RTI_LogicalTimeFactory_h

// The classes associated with logical time allow a federation to provide
// their own representation for logical time and logical time interval. The
// federation is responsible to inherit from the abstract classes declared
// below. The encoded time classes are used to hold the arbitrary bit
// representation of the logical time and logical time intervals.

class RTI_LogicalTime;
class RTI_CouldNotDecode;
class RTI_InternalError;

class RTI_EXPORT RTI_LogicalTimeFactory
{
 public:
  virtual
  ~RTI_LogicalTimeFactory()
    throw ();

  virtual
  RTI_auto_ptr< RTI_LogicalTime >
  makeInitial()
    throw (RTI_InternalError) = 0;

  virtual
  RTI_auto_ptr< RTI_LogicalTime >
  decode(RTI_EncodedLogicalTime const & encodedLogicalTime)
    throw (RTI_InternalError,
           RTI_CouldNotDecode) = 0;
};

#endif // RTI_LogicalTimeFactory_h
```

```
/*************************************************************************
IEEE 1516.1 High Level Architecture Interface Specification C++ API
File: RTI_LogicalTimeInterval.h
*************************************************************************/

#ifndef RTI_LogicalTimeInterval_h
#define RTI_LogicalTimeInterval_h

// The classes associated with logical time allow a federation to provide
// their own representation for logical time and logical time interval. The
// federation is responsible to inherit from the abstract classes declared
// below. The encoded time classes are used to hold the arbitrary bit
// representation of the logical time and logical time intervals.

#include <RTI_memory>
class RTI_EncodedLogicalTimeInterval;

class RTI_EXPORT RTI_LogicalTimeInterval
{
  public:
  virtual
  ~RTI_LogicalTimeInterval()
    throw ();

  virtual
  void
  setZero() = 0;

  virtual
  RTI_bool
  isZero() = 0;

  virtual
  RTI_bool
  isEpsilon() = 0;

  virtual
  void
  setTo(RTI_LogicalTimeInterval const & value)
    throw (RTI_InvalidLogicalTimeInterval) = 0;

  virtual
  RTI_auto_ptr< RTI_LogicalTimeInterval >
  subtract(RTI_LogicalTimeInterval const & subtrahend) const
    throw (RTI_InvalidLogicalTimeInterval) = 0;

  virtual
  RTI_bool
  isGreaterThan(RTI_LogicalTimeInterval const & value) const
    throw (RTI_InvalidLogicalTimeInterval) = 0;

  virtual
  RTI_bool
  isLessThan(RTI_LogicalTimeInterval const & value) const
    throw (RTI_InvalidLogicalTimeInterval) = 0;

  virtual
  RTI_bool
  isEqualTo(RTI_LogicalTimeInterval const & value) const
    throw (RTI_InvalidLogicalTimeInterval) = 0;

  virtual
  RTI_bool
  isGreaterThanOrEqualTo(RTI_LogicalTimeInterval const & value) const
    throw (RTI_InvalidLogicalTimeInterval) = 0;

  virtual
  RTI_bool
  isLessThanOrEqualTo(RTI_LogicalTimeInterval const & value) const
    throw (RTI_InvalidLogicalTimeInterval) = 0;

  virtual
```



```
  RTI_auto_ptr< RTI_EncodedLogicalTimeInterval >
  encode() const = 0;

  virtual
  RTI_wstring
  toString() const = 0;
};

#endif // RTI_LogicalTimeInterval_h



/*********************************************************************
IEEE 1516.1 High Level Architecture Interface Specification C++ API
File: RTI_LogicalTimeIntervalFactory.h
*********************************************************************/

#ifndef RTI_LogicalTimeIntervalFactory_h
#define RTI_LogicalTimeIntervalFactory_h

// The classes associated with logical time allow a federation to provide
// their own representation for logical time and logical time interval. The
// federation is responsible to inherit from the abstract classes declared
// below. The encoded time classes are used to hold the arbitrary bit
// representation of the logical time and logical time intervals.

class RTI_EXPORT RTI_LogicalTimeIntervalFactory
{
 public:
  virtual
  ~RTI_LogicalTimeIntervalFactory()
    throw ();

  virtual
  RTI_auto_ptr< RTI_LogicalTimeInterval >
  makeZero() = 0;

  virtual
  RTI_auto_ptr< RTI_LogicalTimeInterval >
  epsilon() = 0;

  virtual
  RTI_auto_ptr< RTI_LogicalTimeInterval >
  decode(RTI_EncodedLogicalTimeInterval const & encodedLogicalTimeInterval)
    throw (RTI_CouldNotDecode) = 0;
};

#endif // RTI_LogicalTimeIntervalFactory_h


/*********************************************************************
IEEE 1516.1 High Level Architecture Interface Specification C++ API
File: RTI_OrderType.h
*********************************************************************/

#ifndef RTI_OrderType_h
#define RTI_OrderType_h

class RTI_bool;

#include <RTI_SpecificConfig.h>
#include <RTI_SpecificTypedefs.h> // for RTI_EncodedData
#include <RTI_string>

// Type safe class used to represent type of data order.
class RTI_EXPORT RTI_OrderType
{
 public:
  RTI_OrderType(RTI_EncodedData const & theEncodedOrderType);

  RTI_OrderType(RTI_OrderType const & rhs)
```

```
    throw();

  static
  RTI_OrderType const
  receive()
    throw();

  static
  RTI_OrderType const
  timestamp()
    throw();

  RTI_OrderType &
  operator=(RTI_OrderType const & rhs)
    throw ();

  RTI_bool
  operator==(RTI_OrderType const & rhs) const
    throw();

  RTI_bool
  operator!=(RTI_OrderType const & rhs) const
    throw();

  RTI_wstring const
  toString() const;

  RTI_EncodedData
  encode() const
    throw();
 private:
  RTI_OrderType(unsigned orderType)
    throw();

  unsigned _orderType;
};

// These constants save a little typing for users.
// They can be used much like a enum, but in a type-safe way
RTI_OrderType const
RTI_RECEIVE =
RTI_OrderType::receive();

RTI_OrderType const
RTI_TIMESTAMP =
RTI_OrderType::timestamp();

#ifdef RTI_USE_INLINE
#include "RTI_OrderType.i"
#endif // RTI_USE_INLINE

#endif // RTI_OrderType_h



/*************************************************************************
IEEE 1516.1 High Level Architecture Interface Specification C++ API
File: RTI_RTIambassador.h
*************************************************************************/

#ifndef RTI_RTIambassadorFactory_h
#define RTI_RTIambassadorFactory_h

#include <RTI_memory>
#include <RTI_vector>
#include <RTI_string>

class RTI_RTIinternalError;
class RTI_RTIambassador;

classRTI_RTIambassadorFactory
```



```
{
public:
  RTI_RTIambassadorFactory();

  virtual
  ~RTI_RTIambassadorFactory()
    throw ();

  // 10.35
  RTI_auto_ptr< RTI_RTIambassador >
  createRTIambassador(RTI_vector< RTI_wstring,
                                  RTI___STL_DEFAULT_ALLOCATOR(RTI_wstring) >
                      & args)
    throw (RTI_RTIinternalError);
};

#endif // RTI_RTIambassadorFactory_h



/*************************************************************************
IEEE 1516.1 High Level Architecture Interface Specification C++ API
File: RTI_RangeBounds.h
*************************************************************************/

#ifndef RTI_RangeBounds_h
#define RTI_RangeBounds_h

#include <RTI_SpecificConfig.h>

class RTI_EXPORT RTI_RangeBounds
{
public:
  RTI_RangeBounds();

  RTI_RangeBounds(unsigned long lowerBound,
         unsigned long upperBound);

  ~RTI_RangeBounds()
    throw ();

  RTI_RangeBounds(RTI_RangeBounds const & rhs);

  RTI_RangeBounds &
  operator=(RTI_RangeBounds const & rhs);

  unsigned long
  getLowerBound() const;

  unsigned long
  getUpperBound() const;

  void
  setLowerBound(unsigned long lowerBound);

  void
  setUpperBound(unsigned long upperBound);

  private:
  unsigned long _lowerBound;
  unsigned long _upperBound;
};

#ifdef RTI_USE_INLINE
#include "RTI_RangeBounds.i"
#endif // RTI_USE_INLINE

#endif // RTI_RangeBounds_h
```

```
/**************************************************************************
IEEE 1516.1 High Level Architecture Interface Specification C++ API
File: RTI_ResignAction.h
**************************************************************************/

#ifndef RTI_ResignAction_h
#define RTI_ResignAction_h

class RTI_bool;

#include <RTI_SpecificConfig.h>
#include <RTI_string>

// Type safe class used to represent action taken on resign.
class RTI_EXPORT RTI_ResignAction
{
public:
  RTI_ResignAction(RTI_ResignAction const & rhs);

  static
  RTI_ResignAction const
  unconditionallyDivestAttributes();

  static
  RTI_ResignAction const
  deleteObjects();

  static
  RTI_ResignAction const
  cancelPendingOwnershipAcquisitions();

  static
  RTI_ResignAction const
  deleteObjectsThenDivest();

  static
  RTI_ResignAction const
  cancelThenDeleteThenDivest();

  static
  RTI_ResignAction const
  noAction();

  RTI_ResignAction &
  operator=(RTI_ResignAction const & rhs);

  RTI_bool
  operator==(RTI_ResignAction const & rhs) const;

  RTI_bool
  operator!=(RTI_ResignAction const & rhs) const;

  RTI_wstring const
  toString() const;

 private:
  RTI_ResignAction(unsigned _resignAction);

  unsigned _resignAction;
};

// These constants save a little typing for users.
// They can be used much like a enum, but in a type-safe way
RTI_ResignAction const
RTI_UNCONDITIONALLY_DIVEST_ATTRIBUTES
= RTI_ResignAction::unconditionallyDivestAttributes();

RTI_ResignAction const
RTI_DELETE_OBJECTS
= RTI_ResignAction::deleteObjects();

RTI_ResignAction const
```



```
RTI_CANCEL_PENDING_OWNERSHIP_ACQUISITIONS
= RTI_ResignAction::cancelPendingOwnershipAcquisitions();

RTI_ResignAction const
RTI_DELETE_OBJECTS_THEN_DIVEST
= RTI_ResignAction::deleteObjectsThenDivest();

RTI_ResignAction const
RTI_CANCEL_THEN_DELETE_THEN_DIVEST
= RTI_ResignAction::cancelThenDeleteThenDivest();

RTI_ResignAction const
RTI_NO_ACTION
= RTI_ResignAction::noAction();

#ifdef RTI_USE_INLINE
#include "RTI_ResignAction.i"
#endif // RTI_USE_INLINE

#endif // RTI_ResignAction_h



/*************************************************************************
IEEE 1516.1 High Level Architecture Interface Specification C++ API
File: RTI_RestoreFailureReason.h
*************************************************************************/

#ifndef RTI_RestoreFailureReason_h
#define RTI_RestoreFailureReason_h

class RTI_bool;

#include <RTI_SpecificConfig.h>
#include <RTI_string>

// Type safe class used to represent the reason a restore failed
class RTI_EXPORT RTI_RestoreFailureReason
{
 public:
  RTI_RestoreFailureReason(RTI_RestoreFailureReason const & rhs);

  static
  RTI_RestoreFailureReason const
  rtiUnableToRestore();

  static
  RTI_RestoreFailureReason const
  federateReportedFailureDuringRestore();

  static
  RTI_RestoreFailureReason const
  federateResignedDuringRestore();

  static
  RTI_RestoreFailureReason const
  rtiDetectedFailureDuringRestore();

  RTI_RestoreFailureReason &
  operator=(RTI_RestoreFailureReason const & rhs);

  RTI_bool
  operator==(RTI_RestoreFailureReason const & rhs) const;

  RTI_bool
  operator!=(RTI_RestoreFailureReason const & rhs) const;

  RTI_wstring const
  toString() const;
 private:
  RTI_RestoreFailureReason(unsigned RestoreFailureReason);
```

```
  unsigned _RestoreFailureReason;
};

// These constants Restore a little typing for users.
// They can be used much like a enum, but in a type-safe way
RTI_RestoreFailureReason const
RTI_RTI_UNABLE_TO_RESTORE =
RTI_RestoreFailureReason::rtiUnableToRestore();

RTI_RestoreFailureReason const
RTI_FEDERATE_REPORTED_FAILURE_DURING_RESTORE =
RTI_RestoreFailureReason::federateReportedFailureDuringRestore();

RTI_RestoreFailureReason const
RTI_FEDERATE_RESIGNED_DURING_RESTORE =
RTI_RestoreFailureReason::federateResignedDuringRestore();

RTI_RestoreFailureReason const
RTI_RTI_DETECTED_FAILURE_DURING_RESTORE =
RTI_RestoreFailureReason::rtiDetectedFailureDuringRestore();

#ifdef RTI_USE_INLINE
#include "RTI_RestoreFailureReason.i"
#endif // RTI_USE_INLINE
#endif // RTI_RestoreFailureReason_h
/*************************************************************************
IEEE 1516.1 High Level Architecture Interface Specification C++ API
File: RTI_RestoreStatus.h
*************************************************************************/

#ifndef RTI_RestoreStatus_h
#define RTI_RestoreStatus_h

class RTI_bool;

#include <RTI_SpecificConfig.h>
#include <RTI_string>

// Type safe class used to represent save status of an individual federate.
class RTI_EXPORT RTI_RestoreStatus
{
public:
  RTI_RestoreStatus(RTI_RestoreStatus const & rhs);

  static
  RTI_RestoreStatus const
  noRestoreInProgress();

  static
  RTI_RestoreStatus const
  federateRestoreRequestPending();

  static
  RTI_RestoreStatus const
  federateWaitingForRestoreToBegin();

  static
  RTI_RestoreStatus const
  federatePreparedToRestore();

  static
  RTI_RestoreStatus const
  federateRestoring();

  static
  RTI_RestoreStatus const
  federateWaitingForFederationToRestore();

  RTI_RestoreStatus &
  operator=(RTI_RestoreStatus const & rhs);

  RTI_bool
```



```
  operator==(RTI_RestoreStatus const & rhs) const;

  RTI_bool
  operator!=(RTI_RestoreStatus const & rhs) const;

  RTI_wstring const
  toString() const;

private:
  RTI_RestoreStatus(unsigned _RestoreStatus);
  unsigned _RestoreStatus;
};

// These constants save a little typing for users.
// They can be used much like a enum, but in a type-safe way

RTI_RestoreStatus const
RTI_NO_RESTORE_IN_PROGRESS
= RTI_RestoreStatus::noRestoreInProgress();

RTI_RestoreStatus const
RTI_FEDERATE_RESTORE_REQUEST_PENDING
= RTI_RestoreStatus::federateRestoreRequestPending();

RTI_RestoreStatus const
RTI_FEDERATE_WAITING_FOR_RESTORE_TO_BEGIN
= RTI_RestoreStatus::federateWaitingForRestoreToBegin();

RTI_RestoreStatus const
RTI_FEDERATE_PREPARED_TO_RESTORE
= RTI_RestoreStatus::federatePreparedToRestore();

RTI_RestoreStatus const
RTI_FEDERATE_RESTORING
= RTI_RestoreStatus::federateRestoring();

RTI_RestoreStatus const
RTI_FEDERATE_WAITING_FOR_FEDERATION_TO_RESTORE
= RTI_RestoreStatus::federateWaitingForFederationToRestore();

#ifdef RTI_USE_INLINE
#include "RTI_RestoreStatus.i"
#endif // RTI_USE_INLINE

#endif // RTI_RestoreStatus_h



/*************************************************************************
IEEE 1516.1 High Level Architecture Interface Specification C++ API
File: RTI_SaveFailureReason.h
*************************************************************************/

#ifndef RTI_SaveFailureReason_h
#define RTI_SaveFailureReason_h

class RTI_bool;

#include <RTI_SpecificConfig.h>
#include <RTI_string>

// Type safe class used to represent type of data order.
class RTI_EXPORT RTI_SaveFailureReason
{
 public:
  RTI_SaveFailureReason(RTI_SaveFailureReason const & rhs);

  static
  RTI_SaveFailureReason const
  rtiUnableToSave();
```

```
   static
   RTI_SaveFailureReason const
   federateReportedFailureDuringSave();

   static
   RTI_SaveFailureReason const
   federateResignedDuringSave();

   static
   RTI_SaveFailureReason const
   rtiDetectedFailureDuringSave();

   static
   RTI_SaveFailureReason const
   saveTimeCannotBeHonored();

   RTI_SaveFailureReason &
   operator=(RTI_SaveFailureReason const & rhs);

   RTI_bool
   operator==(RTI_SaveFailureReason const & rhs) const;

   RTI_bool
   operator!=(RTI_SaveFailureReason const & rhs) const;

   RTI_wstring const
   toString() const;
 private:
   RTI_SaveFailureReason(unsigned saveFailureReason);

   unsigned _saveFailureReason;
};

// These constants save a little typing for users.
// They can be used much like a enum, but in a type-safe way
RTI_SaveFailureReason const
RTI_RTI_UNABLE_TO_SAVE =
RTI_SaveFailureReason::rtiUnableToSave();

RTI_SaveFailureReason const
RTI_FEDERATE_REPORTED_FAILURE_DURING_SAVE =
RTI_SaveFailureReason::federateReportedFailureDuringSave();

RTI_SaveFailureReason const
RTI_FEDERATE_RESIGNED_DURING_SAVE =
RTI_SaveFailureReason::federateResignedDuringSave();

RTI_SaveFailureReason const
RTI_RTI_DETECTED_FAILURE_DURING_SAVE =
RTI_SaveFailureReason::rtiDetectedFailureDuringSave();

RTI_SaveFailureReason const
RTI_SAVE_TIME_CANNOT_BE_HONORED =
RTI_SaveFailureReason::saveTimeCannotBeHonored();

#ifdef RTI_USE_INLINE
#include "RTI_SaveFailureReason.i"
#endif // RTI_USE_INLINE

#endif // RTI_SaveFailureReason_h
```



```
/**************************************************************************
IEEE 1516.1 High Level Architecture Interface Specification C++ API
File: RTI_SaveStatus.h
**************************************************************************/
#ifndef RTI_SaveStatus_h
#define RTI_SaveStatus_h

class RTI_bool;

#include <RTI_SpecificConfig.h>
#include <RTI_string>

// Type safe class used to represent save status of an individual federate.
class RTI_EXPORT RTI_SaveStatus
{
public:
  RTI_SaveStatus(RTI_SaveStatus const & rhs);

  static
  RTI_SaveStatus const
  noSaveInProgress();

  static
  RTI_SaveStatus const
  federateInstructedToSave();

  static
  RTI_SaveStatus const
  federateSaving();

  static
  RTI_SaveStatus const
  federateWaitingForFederationToSave();

  RTI_SaveStatus &
  operator=(RTI_SaveStatus const & rhs);

  RTI_bool
  operator==(RTI_SaveStatus const & rhs) const;

  RTI_bool
  operator!=(RTI_SaveStatus const & rhs) const;

  RTI_wstring const
  toString() const;
 private:
  RTI_SaveStatus(unsigned _SaveStatus);

  unsigned _SaveStatus;
};

// These constants save a little typing for users.
// They can be used much like a enum, but in a type-safe way
RTI_SaveStatus const
RTI_NO_SAVE_IN_PROGRESS
= RTI_SaveStatus::noSaveInProgress();

RTI_SaveStatus const
RTI_FEDERATE_INSTRUCTED_TO_SAVE

= RTI_SaveStatus::federateInstructedToSave();
RTI_SaveStatus const
RTI_FEDERATE_SAVING
= RTI_SaveStatus::federateSaving();

RTI_SaveStatus const
RTI_FEDERATE_WAITING_FOR_FEDERATION_TO_SAVE
= RTI_SaveStatus::federateWaitingForFederationToSave();

#ifdef RTI_USE_INLINE
#include "RTI_SaveStatus.i"
#endif // RTI_USE_INLINE
#endif // RTI_SaveStatus_h
```

```
/*************************************************************************
IEEE 1516.1 High Level Architecture Interface Specification C++ API
File: RTI_ServiceGroupIndicator.h
*************************************************************************/

#ifndef RTI_ServiceGroupIndicator_h
#define RTI_ServiceGroupIndicator_h

class RTI_bool;

#include <RTI_SpecificConfig.h>
#include <RTI_string>

// Type safe class used to represent the service group
class RTI_EXPORT RTI_ServiceGroupIndicator
{
public:
  RTI_ServiceGroupIndicator(RTI_ServiceGroupIndicator const & rhs);

  static
  RTI_ServiceGroupIndicator const
  federationManagement();

  static
  RTI_ServiceGroupIndicator const
  declarationManagement();

  static
  RTI_ServiceGroupIndicator const
  objectManagement();

  static
  RTI_ServiceGroupIndicator const
  ownershipManagement();

  static
  RTI_ServiceGroupIndicator const
  timeManagement();

  static
  RTI_ServiceGroupIndicator const
  dataDistributionManagement();

  static
  RTI_ServiceGroupIndicator const
  supportServices();

  RTI_ServiceGroupIndicator &
  operator=(RTI_ServiceGroupIndicator const & rhs);

  RTI_bool
  operator==(RTI_ServiceGroupIndicator const & rhs) const;

  RTI_bool
  operator!=(RTI_ServiceGroupIndicator const & rhs) const;

  RTI_wstring const
  toString() const;
 private:
  RTI_ServiceGroupIndicator(unsigned _ServiceGroupIndicator);

  unsigned _ServiceGroupIndicator;
};

// These constants save a little typing for users.
// They can be used much like a enum, but in a type-safe way
RTI_ServiceGroupIndicator const
RTI_FEDERATION_MANAGEMENT
= RTI_ServiceGroupIndicator::federationManagement();

RTI_ServiceGroupIndicator const
RTI_DECLARATION_MANAGEMENT
```



```
= RTI_ServiceGroupIndicator::declarationManagement();

RTI_ServiceGroupIndicator const
RTI_OBJECT_MANAGEMENT
= RTI_ServiceGroupIndicator::objectManagement();

RTI_ServiceGroupIndicator const
RTI_OWNERSHIP_MANAGEMENT
= RTI_ServiceGroupIndicator::ownershipManagement();

RTI_ServiceGroupIndicator const
RTI_TIME_MANAGEMENT
= RTI_ServiceGroupIndicator::timeManagement();

RTI_ServiceGroupIndicator const
RTI_DATA_DISTRIBUTION_MANAGEMENT
= RTI_ServiceGroupIndicator::dataDistributionManagement();

RTI_ServiceGroupIndicator const
RTI_SUPPORT_SERVICES
= RTI_ServiceGroupIndicator::supportServices();

#ifdef RTI_USE_INLINE
#include "RTI_ServiceGroupIndicator.i"
#endif // RTI_USE_INLINE

#endif // RTI_ServiceGroupIndicator_h



/*************************************************************************
IEEE 1516.1 High Level Architecture Interface Specification C++ API
File: RTI_SynchronizationFailureReason.h
*************************************************************************/

#ifndef RTI_SynchronizationFailureReason_h
#define RTI_SynchronizationFailureReason_h

class RTI_bool;

#include <RTI_SpecificConfig.h>
#include <RTI_string>

// Type safe class used to represent type of data order.
class RTI_EXPORT RTI_SynchronizationFailureReason
{
 public:
  RTI_SynchronizationFailureReason(RTI_SynchronizationFailureReason
const & rhs);

  static
  RTI_SynchronizationFailureReason const
  synchronizationPointLabelNotUnique();

  static
  RTI_SynchronizationFailureReason const
  synchronizationSetMemberNotJoined();

  RTI_SynchronizationFailureReason &
  operator=(RTI_SynchronizationFailureReason const & rhs);

  RTI_bool
  operator==(RTI_SynchronizationFailureReason const & rhs) const;

  RTI_bool
  operator!=(RTI_SynchronizationFailureReason const & rhs) const;

  RTI_wstring const
  toString() const;
 private:
  RTI_SynchronizationFailureReason(unsigned SynchronizationFailureReason);
```

```
  unsigned _SynchronizationFailureReason;
};

// These constants Synchronization a little typing for users.
// They can be used much like a enum, but in a type-safe way
RTI_SynchronizationFailureReason const
RTI_SYNCHRONIZATION_POINT_LABEL_NOT_UNIQUE =
RTI_SynchronizationFailureReason::synchronizationPointLabelNotUnique();

RTI_SynchronizationFailureReason const
RTI_SYNCHRONIZATION_SET_MEMBER_NOT_JOINED =
RTI_SynchronizationFailureReason::synchronizationSetMemberNotJoined();

#ifdef RTI_USE_INLINE
#include "RTI_SynchronizationFailureReason.i"
#endif // RTI_USE_INLINE

#endif // RTI_SynchronizationFailureReason_h



/**************************************************************************
IEEE 1516.1 High Level Architecture Interface Specification C++ API
File: RTI_TransportationType.h
**************************************************************************/

#ifndef RTI_TransportationType_h
#define RTI_TransportationType_h

class RTI_bool;

#include <RTI_SpecificConfig.h>
#include <RTI_SpecificTypedefs.h> // for RTI_EncodedData
#include <RTI_string>

// Type safe class used to represent type of data transportation.
class RTI_EXPORT RTI_TransportationType
{
 public:
  RTI_TransportationType(RTI_EncodedData const & rhs);

  RTI_TransportationType(RTI_TransportationType const & rhs)
    throw ();

  static
  RTI_TransportationType const
  reliable()
    throw();

  static
  RTI_TransportationType const
  bestEffort()
    throw();

  RTI_TransportationType &
  operator =(RTI_TransportationType const & rhs)
    throw();

  RTI_bool
  operator ==(RTI_TransportationType const & rhs) const
    throw();

  RTI_bool
  operator !=(RTI_TransportationType const & rhs) const
    throw();

  RTI_wstring const
  toString() const;

  RTI_EncodedData
  encode() const
    throw();
```



```
 private:
  RTI_TransportationType(unsigned transportationType)
    throw();

  unsigned _transportationType;
};

// These constants save a little typing for users.
// They can be used much like a enum, but in a type-safe way
RTI_TransportationType const
RTI_RELIABLE =
RTI_TransportationType::reliable();

RTI_TransportationType const
RTI_BEST_EFFORT =
RTI_TransportationType::bestEffort();

#ifdef RTI_USE_INLINE
#include "RTI_TransportationType.i"
#endif // RTI_USE_INLINE

#endif // RTI_TransportationType_h



/**************************************************************************
IEEE 1516.1 High Level Architecture Interface Specification C++ API
File: RTI_Typedefs.h
**************************************************************************/

// Purpose: This file contains the standard RTI types that are prefixed
// with the string "RTI". These definitions/declarations are standard
// for all RTI implementations.
//
// The types declared here require the use of some RTI_Specific types.

#ifndef RTI_Typedefs_h
#define RTI_Typedefs_h

// The following type definitions use standard C++ classes for containers
// that are used in the RTI API.

#include <RTI_set>
#include <RTI_map>
#include <RTI_vector>
#include <RTI_SpecificTypedefs.h>

typedef RTI_set< RTI_AttributeHandle, RTI_less< RTI_AttributeHandle > ,
                 RTI___STL_DEFAULT_ALLOCATOR (RTI_AttributeHandle)>
 RTI_AttributeHandleSet;

typedef RTI_set<RTI_FederateHandle, RTI_less<RTI_FederateHandle > ,
                RTI___STL_DEFAULT_ALLOCATOR (RTI_FederateHandle)>
 RTI_FederateHandleSet;

typedef RTI_set< RTI_DimensionHandle, RTI_less< RTI_DimensionHandle >,
                 RTI___STL_DEFAULT_ALLOCATOR (RTI_DimensionHandle) >
 RTI_DimensionHandleSet;

typedef RTI_set< RTI_RegionHandle, RTI_less< RTI_RegionHandle >,
                 RTI___STL_DEFAULT_ALLOCATOR (RTI_RegionHandle) >
 RTI_RegionHandleSet;

// RTI_AttributeHandleValueMap implements a constrained set of
// (attribute handle and value) pairs
typedef RTI_map< RTI_AttributeHandle, RTI_AttributeValue, RTI_less >
                 RTI_AttributeHandle >, RTI___STL_DEFAULT_ALLOCATOR
                (RTI_AttributeHandle) >
 RTI_AttributeHandleValueMap;

// RTI_ParameterHandleValueMap implements a constrained set of
```

```
// (parameter handle and value) pairs
typedef RTI_map< RTI_ParameterHandle, RTI_ParameterValue, RTI_less<
                RTI_ParameterHandle >, RTI___STL_DEFAULT_ALLOCATOR
               (RTI_ParameterHandle) >
 RTI_ParameterHandleValueMap;

// RTI_AttributeHandleSetRegionHandleSetPairVector implements a collection of
// (attribute handle set and region set) pairs
typedef RTI_pair< RTI_AttributeHandleSet, RTI_RegionHandleSet >
  RTI_AttributeHandleSetRegionHandleSetPair;

typedef RTI_vector< RTI_AttributeHandleSetRegionHandleSetPair,
                   RTI___STL_DEFAULT_ALLOCATOR
                  (RTI_AttributeHandleSetRegionHandleSetPair) >
 RTI_AttributeHandleSetRegionHandleSetPairVector;

// RTI_FederateHandleSaveStatusPairVector implements a collection of
// (federate handle and save status) pairs
typedef RTI_pair< RTI_FederateHandle, RTI_SaveStatus >
 RTI_FederateHandleSaveStatusPair;

typedef RTI_vector< RTI_FederateHandleSaveStatusPair,
                   RTI___STL_DEFAULT_ALLOCATOR(RTI_FederateHandleSaveStatusPair >
 RTI_FederateHandleSaveStatusPairVector;

// RTI_FederateHandleRestoreStatusPairVector implements a collection of
// (federate handle and restore status) pairs
typedef RTI_pair< RTI_FederateHandle, RTI_RestoreStatus >
                 RTI_FederateHandleRestoreStatusPair;

typedef RTI_vector< RTI_FederateHandleRestoreStatusPair,
                   RTI___STL_DEFAULT_ALLOCATOR(RTI_FederateHandleRestore
StatusPair) >
 RTI_FederateHandleRestoreStatusPairVector;

#endif // RTI_Typedefs_h



/**************************************************************************
IEEE 1516.1 High Level Architecture Interface Specification C++ API
File: RTI_Value.h
**************************************************************************/

#ifndef RTI_Value_h
#define RTI_Value_h

#include <RTI_SpecificConfig.h>

// The RTI_Value class is used to as a generic value holder that contains a
// pointer to the memory location and the size of the value. This class is
// used for attributes, parameters, and user supplied tags.The constructor
// takes a pointer to the data value and the size of the data. The key methods
// on this class is the data method which returns a constant pointer to the
// value memory location, and the size method which returns the size in bytes
// of the value. The templatized class T provides RTI implementations the
// ability to customize their particular implementation. The int template type
// parameter provides the ability to support strong typing.

template < class T, int i >
RTI_EXPORT class RTI_Value
{
public:
  RTI_Value (void const * data, size_t size);
  ~RTI_Value()
    throw ();

  RTI_Value(RTI_Value const & rhs);

  RTI_Value &
  operator=(RTI_Value const & rhs);
```



```
  void const *
  data() const;

  size_t
  size() const;

private:
  T _impl;
};

#ifdef RTI_USE_INLINE
#include "RTI_Value.i"
#endif // RTI_USE_INLINE

#ifdef RTI_TEMPLATES_REQUIRE_SOURCE
#include "RTI_Value.cpp"
#endif // RTI_TEMPLATES_REQUIRE_SOURCE

#endif // RTI_Value_h



/************************************************************************
IEEE 1516.1 High Level Architecture Interface Specification C++ API
File: RTI_bool.h
************************************************************************/

#ifndef RTI_bool_h
#define RTI_bool_h

#include <RTI_SpecificConfig.h>
#include <RTI_string>

// This is a type-safe boolean class.Not all platforms support the bool
// type. Consequently, one must be provided.
//
class RTI_EXPORT RTI_bool
{
 public:
  RTI_bool();

  RTI_bool(unsigned boolean);
  RTI_bool(RTI_bool const & b);

  ~RTI_bool()
    throw();

  static
  RTI_bool const
  True();

  static
  RTI_bool const
  False();

  operator unsigned() const;

  RTI_bool const
  operator==(RTI_bool const & rhs) const;

  RTI_bool const
  operator!=(RTI_bool const & rhs) const;

  RTI_wstring
  toString() const;
 private:
unsigned _boolean;
};

#ifdef RTI_USE_INLINE
#include "RTI_bool.i"
```

```
#endif // RTI_USE_INLINE

RTI_bool const RTI_true= RTI_bool::True();
RTI_bool const RTI_false = RTI_bool::False();

#endif // RTI_bool_h



/*************************************************************************
IEEE 1516.1 High Level Architecture Interface Specification C++ API
File: RTI_exception.h
*************************************************************************/
#ifndefRTI_exception_h
#defineRTI_exception_h
#include <RTI_SpecificConfig.h>

// The RTI_exception class follows the interface of the C++ standard exception
// class.The key method, what, returns a null terminated character string that
// describes details of the exception that has occured.

class RTI_exception
{
public:
  RTI_exception();

  RTI_exception(RTI_exception const & rhs);

  RTI_exception &
  operator=(RTI_exception const & rhs);

  virtual
  ~RTI_exception()
    throw();

  virtual
  char const *
  what() const
    throw() = 0;
};

#include <RTI_string>

#define RTI_EXCEPTION(A) class A : public RTI_exception { public: A(RTI_string
const & message) throw(); A(A const & rhs) throw(); A & operator=(A const &)
throw(); virtual ~A() throw(); virtual char const * what() const throw(); pri-
vate: RTI_string _msg; };

RTI_EXCEPTION(RTI_ArrayIndexOutOfBounds)
RTI_EXCEPTION(RTI_AsynchronousDeliveryAlreadyDisabled)
RTI_EXCEPTION(RTI_AsynchronousDeliveryAlreadyEnabled)
RTI_EXCEPTION(RTI_AttributeAcquisitionWasNotCanceled)
RTI_EXCEPTION(RTI_AttributeAcquisitionWasNotRequested)
RTI_EXCEPTION(RTI_AttributeAlreadyBeingAcquired)
RTI_EXCEPTION(RTI_AttributeAlreadyBeingDivested)
RTI_EXCEPTION(RTI_AttributeAlreadyOwned)
RTI_EXCEPTION(RTI_AttributeDivestitureWasNotRequested)
RTI_EXCEPTION(RTI_AttributeNotDefined)
RTI_EXCEPTION(RTI_AttributeNotOwned)
RTI_EXCEPTION(RTI_AttributeNotPublished)
RTI_EXCEPTION(RTI_AttributeNotRecognized)
RTI_EXCEPTION(RTI_AttributeNotSubscribed)
RTI_EXCEPTION(RTI_AttributeRelevanceAdvisorySwitchIsOff)
RTI_EXCEPTION(RTI_AttributeRelevanceAdvisorySwitchIsOn)
RTI_EXCEPTION(RTI_AttributeScopeAdvisorySwitchIsOff)
RTI_EXCEPTION(RTI_AttributeScopeAdvisorySwitchIsOn)
RTI_EXCEPTION(RTI_CouldNotDecode)
RTI_EXCEPTION(RTI_CouldNotDiscover)
RTI_EXCEPTION(RTI_CouldNotOpenFDD)
RTI_EXCEPTION(RTI_CouldNotInitiateRestore)
RTI_EXCEPTION(RTI_DeletePrivilegeNotHeld)
```



```
RTI_EXCEPTION(RTI_RequestForTimeConstrainedPending)
RTI_EXCEPTION(RTI_NoRequestToEnableTimeConstrainedWasPending)
RTI_EXCEPTION(RTI_RequestForTimeRegulationPending)
RTI_EXCEPTION(RTI_NoRequestToEnableTimeRegulationWasPending)
RTI_EXCEPTION(RTI_ErrorReadingFDD)
RTI_EXCEPTION(RTI_FederateAlreadyExecutionMember)
RTI_EXCEPTION(RTI_FederateHasNotBegunSave)
RTI_EXCEPTION(RTI_FederateInternalError)
RTI_EXCEPTION(RTI_FederateNotExecutionMember)
RTI_EXCEPTION(RTI_FederateOwnsAttributes)
RTI_EXCEPTION(RTI_FederateServiceInvocationsAreBeingReportedViaMOM)
RTI_EXCEPTION(RTI_FederateUnableToUseTime)
RTI_EXCEPTION(RTI_FederatesCurrentlyJoined)
RTI_EXCEPTION(RTI_FederationExecutionAlreadyExists)
RTI_EXCEPTION(RTI_FederationExecutionDoesNotExist)
RTI_EXCEPTION(RTI_IllegalName)
RTI_EXCEPTION(RTI_IllegalTimeArithmetic)
RTI_EXCEPTION(RTI_InteractionClassNotDefined)
RTI_EXCEPTION(RTI_InteractionClassNotPublished)
RTI_EXCEPTION(RTI_InteractionClassNotRecognized)
RTI_EXCEPTION(RTI_InteractionClassNotSubscribed)
RTI_EXCEPTION(RTI_InteractionParameterNotDefined)
RTI_EXCEPTION(RTI_InteractionParameterNotRecognized)
RTI_EXCEPTION(RTI_InteractionRelevanceAdvisorySwitchIsOff)
RTI_EXCEPTION(RTI_InteractionRelevanceAdvisorySwitchIsOn)
RTI_EXCEPTION(RTI_InTimeAdvancingState)
RTI_EXCEPTION(RTI_InvalidAttributeHandle)
RTI_EXCEPTION(RTI_InvalidDimensionHandle)
RTI_EXCEPTION(RTI_InvalidInteractionClassHandle)
RTI_EXCEPTION(RTI_InvalidLogicalTime)
RTI_EXCEPTION(RTI_InvalidLogicalTimeInterval)
RTI_EXCEPTION(RTI_InvalidLookahead)
RTI_EXCEPTION(RTI_InvalidObjectClassHandle)
RTI_EXCEPTION(RTI_InvalidOrderName)
RTI_EXCEPTION(RTI_InvalidOrderType)
RTI_EXCEPTION(RTI_InvalidParameterHandle)
RTI_EXCEPTION(RTI_InvalidRangeBounds)
RTI_EXCEPTION(RTI_InvalidRegion)
RTI_EXCEPTION(RTI_InvalidRegionContext)
RTI_EXCEPTION(RTI_InvalidRetractionHandle)
RTI_EXCEPTION(RTI_InvalidString)
RTI_EXCEPTION(RTI_InvalidTransportationName)
RTI_EXCEPTION(RTI_InvalidTransportationType)
RTI_EXCEPTION(RTI_JoinedFederateIsNotInTimeAdvancingState)
RTI_EXCEPTION(RTI_LogicalTimeAlreadyPassed)
RTI_EXCEPTION(RTI_LowerBoundOutOfRange)
RTI_EXCEPTION(RTI_MessageCanNoLongerBeRetracted)
RTI_EXCEPTION(RTI_NameNotFound)
RTI_EXCEPTION(RTI_ObjectClassNotDefined)
RTI_EXCEPTION(RTI_ObjectClassNotKnown)
RTI_EXCEPTION(RTI_ObjectClassNotPublished)
RTI_EXCEPTION(RTI_ObjectClassNotSubscribed)
RTI_EXCEPTION(RTI_ObjectClassRelevanceAdvisorySwitchIsOff)
RTI_EXCEPTION(RTI_ObjectClassRelevanceAdvisorySwitchIsOn)
RTI_EXCEPTION(RTI_ObjectInstanceNameInUse)
RTI_EXCEPTION(RTI_ObjectInstanceNameNotReserved)
RTI_EXCEPTION(RTI_ObjectInstanceNotKnown)
RTI_EXCEPTION(RTI_OwnershipAcquisitionPending)
RTI_EXCEPTION(RTI_RTIinternalError)
RTI_EXCEPTION(RTI_RegionDoesNotContainSpecifiedDimension)
RTI_EXCEPTION(RTI_RegionInUseForUpdateOrSubscription)
RTI_EXCEPTION(RTI_RegionNotCreatedByThisFederate)
RTI_EXCEPTION(RTI_RestoreInProgress)
RTI_EXCEPTION(RTI_RestoreNotInProgress)
RTI_EXCEPTION(RTI_RestoreNotRequested)
RTI_EXCEPTION(RTI_SaveInProgress)
RTI_EXCEPTION(RTI_SaveNotInitiated)
RTI_EXCEPTION(RTI_SaveNotInProgress)
RTI_EXCEPTION(RTI_SpecifiedSaveLabelDoesNotExist)
RTI_EXCEPTION(RTI_SynchronizationPointLabelNotAnnounced)
RTI_EXCEPTION(RTI_SynchronizationSetMemberNotJoined)
RTI_EXCEPTION(RTI_TimeConstrainedAlreadyEnabled)
```

```
RTI_EXCEPTION(RTI_TimeConstrainedIsNotEnabled)
RTI_EXCEPTION(RTI_TimeRegulationAlreadyEnabled)
RTI_EXCEPTION(RTI_TimeRegulationIsNotEnabled)
RTI_EXCEPTION(RTI_UnableToPerformSave)
RTI_EXCEPTION(RTI_UnknownName)
RTI_EXCEPTION(RTI_UpperBoundOutOfRange)
RTI_EXCEPTION(RTI_ValueCountExceeded)
RTI_EXCEPTION(RTI_ValueLengthExceeded)

#undef RTI_EXCEPTION

#ifdef RTI_USE_INLINE
#include "RTI_exception.i"
#endif // RTI_USE_INLINE
#endif // RTI_exception_h
```

C.2 Notes

The following files are RTI implementation specific.

```
/*************************************************************************
IEEE 1516.1 High Level Architecture Interface Specification C++ API
File: RTI_SpecificConfig.h
*************************************************************************/

// Purpose: This file contains definitions that are used to isolate
// platform configuration issues.

#ifndef RTI_SpecificConfig_h
#define RTI_SpecificConfig_h

#include <sys/types.h> // for size_t

// macro to support Windows library export mechanism
#define RTI_EXPORT /* nothing */

//
// When and how to use inline methods is left up to the RTI implementor
//
#ifdefRTI_DEBUG_ON
#undefRTI_USE_INLINE
#define RTI_INLINE /* inline */
#else
#define RTI_USE_INLINE
#define RTI_INLINE inline
#endif // RTI_DEBUG_ON

#ifdef __GNUC__
#ifndef RTI_TEMPLATES_REQUIRE_SOURCE
#define RTI_TEMPLATES_REQUIRE_SOURCE
#endif
#endif // __GNUC__

#endif // RTI_SpecificConfig_h



/*************************************************************************
IEEE 1516.1 High Level Architecture Interface Specification C++ API
File: RTI_SpecificPostamble.h
*************************************************************************/

// Purpose: This file contains definitions that need to follow the RTI
// class declaration that are specific to the RTI implementation.

#ifndef RTIspecificPostamble_h
#define RTIspecificPostamble_h
```



```
// RTI vendor specific details could go here

#endif // RTIspecificPostamble_h



/*************************************************************************
IEEE 1516.1 High Level Architecture Interface Specification C++ API
File: RTI_SpecificTypedefs.h
*************************************************************************/

// Purpose: This file contains definitions for RTI types that are specific to
// the RTI implementation. The parameterized classes RTI_Handle and RTI_Value
// enable the RTI implementation to insert a private member. The private
// member declaration must be exposed in this header file. These
// implementation specific declarations and definitions should not be
// utilized since it is non-standard with respect to RTI implementations.

#ifndef RTI_SpecificTypedefs_h
#define RTI_SpecificTypedefs_h

// The following type definitions correspond to standard types used in the API
// (e.g., FederateHandle) that are described through the use of RTI
// implementation specific representations.
//
// The following definitions represent an example of specific representations
// that could be used by an RTI implementation.

#include <RTI_SpecificConfig.h>
//#include <RTI_Value.h>

template < class T, int i > class RTI_Value;

//
// This class is an example of something that is RTI implementor specific.
//
#include <RTI_VariableLengthValueClass.h>

//
// The existence of these typedefs is required by the API Specification.
// However, their particular definition is implementation-specific.RTI-user
// code merely uses the typedefs. What their underlying type is, however, is up
// to the RTI implementor.
//
typedef RTI_Value< RTI_VariableLengthValueClass, 1 > RTI_AttributeValue;
typedef RTI_Value< RTI_VariableLengthValueClass, 2 > RTI_ParameterValue;
typedef RTI_Value< RTI_VariableLengthValueClass, 3 > RTI_UserSuppliedTag;
typedef RTI_Value< RTI_VariableLengthValueClass, 4 > RTI_EncodedData;

//
// The names of these classes are left up to the RTI implementor.
//
class RTI_FederateHandleFactory;
class RTI_ObjectClassHandleFactory;
class RTI_InteractionClassHandleFactory;
class RTI_ObjectInstanceHandleFactory;
class RTI_AttributeHandleFactory;
class RTI_ParameterHandleFactory;
class RTI_DimensionHandleFactory;
class RTI_MessageRetractionHandleFactory;
class RTI_RegionHandleFactory;

//#include <RTI_Handle.h>
template<class ImplementationSpecificHandleClass,
class EncodedHandleClass,
class ImplementationSpecificHandleClassFriend,
inti>
class RTI_Handle;

//
// The existence of these typedefs is required by the API Specification.
```

```
// However, their particular definition is implementation-specific.
//
typedef
RTI_Handle< long, RTI_EncodedData, RTI_FederateHandleFactory, 1 >
RTI_FederateHandle;

typedef
RTI_Handle< long, RTI_EncodedData, RTI_ObjectClassHandleFactory, 2 >
RTI_ObjectClassHandle;

typedef
RTI_Handle< long, RTI_EncodedData, RTI_InteractionClassHandleFactory, 3 >
RTI_InteractionClassHandle;

typedef
RTI_Handle< long, RTI_EncodedData, RTI_ObjectInstanceHandleFactory, 4 >
RTI_ObjectInstanceHandle;

typedef
RTI_Handle< long, RTI_EncodedData, RTI_AttributeHandleFactory, 5 >
RTI_AttributeHandle;

typedef
RTI_Handle< long, RTI_EncodedData, RTI_ParameterHandleFactory, 6 >
RTI_ParameterHandle;

typedef
RTI_Handle< long, RTI_EncodedData, RTI_DimensionHandleFactory, 7 >
RTI_DimensionHandle;

typedef
RTI_Handle< long, RTI_EncodedData, RTI_MessageRetractionHandleFactory, 8 >
RTI_MessageRetractionHandle;

typedef
RTI_Handle< long, RTI_EncodedData, RTI_RegionHandleFactory, 11 >
RTI_RegionHandle;

#endif // RTI_SpecificTypedefs_h
```

C.3 Examples

This subclause is informative. It provides examples of integer 64-based implementations of the HLA time ADTs.

```
/**********************************************************************
IEEE 1516.1 High Level Architecture Interface Specification C++ API
File: RTI_LogicalTimeExample.h
**********************************************************************/

// This file contains an example implementation of federation specified logical
// time classes used to synchronize events. This example implementation uses a
// "64 bit integer" to represent time and time interval. The HLA offers federations
// the ability to construct their own representation and behavior for logical time,
// and a federate application will always need to provide an implementation of
// these classes for linking purposes even if the federation does not utilize the
// time management services of the RTI.

#ifndef RTI_LogicalTimeExample_h
#define RTI_LogicalTimeExample_h

#include <RTI_1516.h>

class RTI_EXPORT RTI_LogicalTimeIntervalExample
: public RTI_LogicalTimeInterval
{
 public:
  RTI_LogicalTimeIntervalExample(unsigned long long time)
```



```
    throw ();

  virtual
  ~RTI_LogicalTimeIntervalExample()
    throw ();

  virtual
  void
  setZero();

  virtual
  RTI_bool
  isZero();

  virtual
  RTI_bool
  isEpsilon();

  virtual
  void
  setTo(RTI_LogicalTimeInterval const & value)
    throw (RTI_InvalidLogicalTimeInterval);

  virtual
  RTI_auto_ptr< RTI_LogicalTimeInterval >
  subtract(RTI_LogicalTimeInterval const & subtrahend) const
    throw (RTI_InvalidLogicalTimeInterval);
virtual
RTI_bool
isGreaterThan(RTI_LogicalTimeInterval const & value) const
 throw (RTI_InvalidLogicalTimeInterval);

  virtual
  RTI_bool
  isLessThan(RTI_LogicalTimeInterval const & value) const
    throw (RTI_InvalidLogicalTimeInterval);

  virtual
  RTI_bool
  isEqualTo(RTI_LogicalTimeInterval const & value) const
    throw (RTI_InvalidLogicalTimeInterval);

  virtual
  RTI_bool
  isGreaterThanOrEqualTo(RTI_LogicalTimeInterval const & value) const
    throw (RTI_InvalidLogicalTimeInterval);

  virtual
  RTI_bool
  isLessThanOrEqualTo(RTI_LogicalTimeInterval const & value) const
    throw (RTI_InvalidLogicalTimeInterval);

  virtual
  RTI_auto_ptr< RTI_EncodedLogicalTimeInterval >
  encode() const;

  virtual
  RTI_wstring
  toString() const;

  unsigned long long getValue() const;

 private:
  unsigned long long _time;
};

class RTI_EXPORT RTI_LogicalTimeExample
: public RTI_LogicalTime
{
 public:
  RTI_LogicalTimeExample(unsigned long long theTime);
```

```
virtual
~RTI_LogicalTimeExample()
  throw ();

virtual
void
setInitial();

virtual
RTI_bool
isInitial();

virtual
void
setFinal();

virtual
RTI_bool
isFinal();

virtual
void
setTo(RTI_LogicalTime const & value)
  throw (RTI_InvalidLogicalTime);

virtual
void
increaseBy(RTI_LogicalTimeInterval const & addend)
  throw (RTI_IllegalTimeArithmetic, RTI_InvalidLogicalTimeInterval);

virtual
void
decreaseBy(RTI_LogicalTimeInterval const & subtrahend)
  throw (RTI_IllegalTimeArithmetic, RTI_InvalidLogicalTimeInterval);

virtual
RTI_auto_ptr< RTI_LogicalTimeInterval >
subtract(RTI_LogicalTime const & subtrahend) const
  throw (RTI_InvalidLogicalTime);

virtual
RTI_bool
isGreaterThan(RTI_LogicalTime const & value) const
  throw (RTI_InvalidLogicalTime);

virtual
RTI_bool
isLessThan(RTI_LogicalTime const & value) const
  throw (RTI_InvalidLogicalTime);

virtual
RTI_bool
isEqualTo(RTI_LogicalTime const & value) const
  throw (RTI_InvalidLogicalTime);

virtual
RTI_bool
isGreaterThanOrEqualTo(RTI_LogicalTime const & value) const
  throw (RTI_InvalidLogicalTime);

virtual
RTI_bool
isLessThanOrEqualTo(RTI_LogicalTime const & value) const
  throw (RTI_InvalidLogicalTime);

virtual
RTI_auto_ptr< RTI_EncodedLogicalTime >
encode() const;

virtual
RTI_wstring
toString() const;
```



```
 private:
  unsigned long long _time;
};

class RTI_EXPORT RTI_LogicalTimeFactoryExample
: public RTI_LogicalTimeFactory
{
 public:

  virtual
  RTI_auto_ptr< RTI_LogicalTime >
  makeInitial();

  virtual
  RTI_auto_ptr< RTI_LogicalTime >
  decode(RTI_EncodedLogicalTime const & encodedLogicalTime)
    throw (RTI_CouldNotDecode);

};

class RTI_EXPORT RTI_LogicalTimeIntervalFactoryExample
: public RTI_LogicalTimeIntervalFactory
{
 public:

  virtual
  RTI_auto_ptr< RTI_LogicalTimeInterval >
  makeZero();

  virtual
  RTI_auto_ptr< RTI_LogicalTimeInterval >
  epsilon();

  virtual
  RTI_auto_ptr< RTI_LogicalTimeInterval >
  decode(RTI_EncodedLogicalTimeInterval const & encodedLogicalTimeInterval)
    throw (RTI_CouldNotDecode);
};

#endif // RTI_LogicalTimeExample_h



/*********************************************************************
IEEE 1516.1 High Level Architecture Interface Specification C++ API
File: RTI_EncodedLogicalTimeExample.h
*********************************************************************/

#ifndef RTI_EncodedLogicalTimeExample_h
#define RTI_EncodedLogicalTimeExample_h

#include <RTI_EncodedLogicalTime.h>

class RTI_EncodedLogicalTimeExample
  : public RTI_EncodedLogicalTime
{
public:
  RTI_EncodedLogicalTimeExample(void const * theData, size_t theSize)
  : _data(theData), _size(theSize) {};
  voidconst * data() const { return _data; }
  size_tsize() const { return _size; }
private:
  void const * _data;
  size_t_size;
};

#endif // RTI_EncodedLogicalTimeExample_h
```

```
/************************************************************************
IEEE 1516.1 High Level Architecture Interface Specification C++ API
File: RTI_EncodedLogicalTimeIntervalExample.h
************************************************************************/

#ifndef RTI_EncodedLogicalTimeIntervalExample_h
#define RTI_EncodedLogicalTimeIntervalExample_h

#include <RTI_EncodedLogicalTimeInterval.h>

class RTI_EncodedLogicalTimeIntervalExample
  : public RTI_EncodedLogicalTimeInterval
{
public:
  RTI_EncodedLogicalTimeIntervalExample(void const * theData, size_t theSize)
  : _data(theData), _size(theSize) {};
  voidconst * data() const { return _data; }
  size_t     size() const { return _size; }
private:
  void const * _data;
  size_t       _size;
};

//typedef EncodedLogicalTimeIntervalExample UsersEncodedLogicalTimeInterval;

#endif // RTI_EncodedLogicalTimeIntervalExample_h
```

Annex D

(informative)

Rationale

D.1 Overview

This Annex contains additional information that might be useful to explain various aspects of HLA and is NOT a complete rationale of the HLA. Nothing in this clause is required in an HLA implementation. The information is provided as a courtesy.

The remainder of this Annex is organized similar to the main 12 clauses of this specification and the clause numbers of this Annex correspond to those of the main body of this specification. Text will be present in this annex if there is something applicable to say with respect to the relevant clause in the main body of this specification.

D.1.1 Scope

Clause intentionally empty.

D.1.2 Purpose

Clause intentionally empty.

D.1.3 Introduction

Clause intentionally empty.

D.1.4 Background

Consideration was given to providing several iterating functions, however, there were found to be serious problems documenting and describing the iterating-based semantics of these services. Additionally, there was a problem with every redaction produced. The candidate definitions of how exceptions were handled and recovered from during the “middle” of an iteration sequence never satisfied a large enough set of users to be acceptable for retention in the specification. Each of the users wanted the behavior to meet their specific application needs, which in most cases were inconsistent and impossible to meet with a common mechanism. As a result, it seemed to make the most sense not to have any iterating services and to let each user produce their own aggregate functionality with the exception response semantics they needed.

D.2 References

Clause intentionally empty.

D.3 Abbreviations, acronyms, and definitions

Clause intentionally empty.

D.4 Federation management

D.4.1 Overview

Clause intentionally empty.

D.4.2 Create Federation Execution

Clause intentionally empty.

D.4.3 Destroy Federation Execution

Since there can be no joined federates for the destroy service invocation to succeed, there might be a problem if the federation execution has “zombie” or otherwise uncooperative joined federates. To aid matters in this situation, a joined federate can be resigned by another joined federate via a MOM interaction.

D.5 Declaration management

D.5.1 Overview

One reason joined federates can subscribe at multiple levels in a class hierarchy is to allow for FOM growth. As a FOM expands, a federate application need not be recoded. It can continue to subscribe at the same levels in the hierarchy while other, “newer,” federate applications in the federation can subscribe at the new classes.

Some services in the interface specification contain postconditions that are less precise than one would expect. The postcondition for the *Subscribe Object Class Attributes* service, for example, states only that “the RTI has been informed of the joined federate’s requested subscription.” It does not say that the joined federate is now subscribed to the specified class attributes, nor is a statement to this effect found in any service postcondition.

Services with such imprecise postconditions include the following:

— Subscribe Object Class Attributes

— Subscribe Interaction Class

— Commit Region Modifications (old Modify Region)

— Subscribe Object Class Attributes with Regions

— Subscribe Interaction Class with Regions

These services have imprecise postconditions, similar to the postcondition quoted above, because the interface specification allows RTI implementations to decide when to determine whether or not a message sent by one federate should be received by another federate. One possibility is to broadcast all sent messages, and perform filtering at each receiving federate. A more efficient possibility is to perform filtering when the message is sent (meaning that the message is sent only to interested federates) and again at each receiving federate (in case the state of the receiver has changed during transmission).

The problem with allowing the latter alternative is that should the receiver add to its subscriptions, that addition cannot be considered to have fully taken effect until the RTI can ensure that messages sent by any

federate will take the added subscriptions into account, thus the imprecise postconditions. Subscription additions take effect locally immediately, but there will be some delay before the additions take effect throughout the federation execution. An RTI implementation must ensure that these changes take effect throughout the federation execution as soon as possible, but it will not delay the return of these service invocations until it can guarantee the subscription additions are complete.

Since federates can come and go and can add subscriptions at any time, the only way the RTI could meet a postcondition asserting that the federate is now subscribed to the specified attributes would be to synchronize the whole federation execution so that it could make that guarantee. This is one of the places in the specification where the desire was to permit more federate asynchronous behavior. This places more responsibility on the federates to ensure that they know what they are doing and make their subscriptions in sufficient time for them to have the desired effect in their federation execution.

D.5.1.1 Static properties of the FED

Clause intentionally empty.

D.5.1.2 Definitions and constraints for object classes and class attributes

Item (o) is a very complex sentence and it has been suggested that it may be difficult for the reader to parse and understand. This text is an attempt to explain what it means by example.

Consider Figure D.1, the object class inheritance tree:

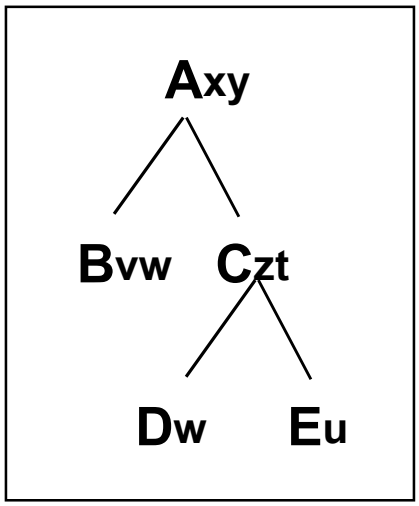

Figure D.1—Example object class inheritance tree

We will use the following notation:

— **Subscribe[C(z)]** denotes the invocation of the Subscribe Object Class Attributes service to subscribe to object class C and class attribute z.

— **Publish[E(x, u)]** denotes the invocation of the Publish Object Class Attributes service to publish object class E, and class attributes x and u.

— **Register (E, e1)** denotes the invocation of the Register Object Instance service to register an object instance of class E that will be denoted e1.

— **e1(x)** denotes instance attribute x of object instance e1.

Suppose federate ***F*** invokes

— Subscribe [C(z)]

— Subscribe [A(x,y)]

Suppose federate ***H*** invokes

— Subscribe [C(x,z)]

— Subscribe [A(y)]

Suppose federate ***G*** invokes

— Publish [C(z,t)]

— Publish [E(x,y,u)]

— Register (E, e1)

Then

— Ownership: Federate ***G*** now owns e1(x), e1(y) and e1(u). Even though federate ***G*** was publishing class attributes z and t when it registered e1, federate ***G*** was not publishing z or t at the registered class of e1 (class E) when it registered e1, so federate ***G*** does not own e1(z) or e1(t). e1(z) and e1(t) are not owned by any federate.

— Discovery at Federate ***F***: Federate ***F*** will not discover object instance e1. The candidate discovery class of e1 at federate ***F*** is class C. But federate ***F*** is not subscribed to any class attribute at class C that has a corresponding instance attribute that is part of e1 and that is owned. Therefore, object instance e1 is not known to federate ***F***, and therefore, it has no known class at federate ***F***. Federate ***F*** is not capable of receiving reflects for updates made to any values of the instance attributes of object instance e1. Even though federate ***F*** is subscribed to class attribute x at class A, an update made to instance attribute e1(x) will not be reflected at ***F*** because e1 is not known to federate ***F***. (For more information on discovery and known classes see also 6.1 of IEEE Std 1516.1-2000)

— Discovery at Federate ***H***: Federate ***H*** will discover object instance e1 as a class C object instance. The candidate discovery class of e1 at federate ***H*** is class C, because federate ***H*** is not subscribed to e1's registered class (class E), and class C is the closest superclass of e1's registered class to which federate ***H*** is subscribed. Federate ***H*** is also subscribed to class attribute x at class C, and class attribute x has a corresponding instance attribute that is part of object instance e1 and that is owned, namely e1(x). Therefore, the known class of e1 at federate ***H*** is class C.

— Reflection: When federate ***G*** updates the value of e1(x), this update will be reflected at Federate ***H***. However, when federate ***G*** updates the value of e1(y) or e1(u), these updates will not be reflected at federate ***H*** because federate ***H*** is not subscribed to class attributes y or u at class C. Notice that even though federate ***H*** is subscribed to class attribute y, it will not receive reflects for updates made to the value of e1(y) because federate ***H*** is not subscribed to class attribute y at the known class (class C) of e1.

Let us look at this example in terms of the wording of item (o) of 5.1.2, that says, "An update to an instance attribute by the federate that owns that instance attribute shall be reflected only by other federates that are subscribed to the instance attribute's corresponding class attribute at the known class of the object instance that contains that instance attribute at the subscribing federate." In terms of our example, this means the following:

An update to e1(x) by federate ***G*** shall be reflected by federate ***H***, because federate ***H*** is subscribed to class attribute x at e1's known class at federate ***H*** (namely, class C). An update to e1(y) or e1(u) by federate ***G*** shall not be reflected by federate ***H*** because federate ***H*** is not subscribed to class attributes y or u at e1's known class at H (class C).

D.6 Object management

D.6.1 Overview

For examples that further elucidate the concepts of discovery and known class, see D.5.1.2 and D.9.1.5 of this Annex.

If an instance attribute is in scope for a given joined federate ***F***, there exists at least one other joined federate in the federation execution that is capable of invoking the *Update Attribute Values* service for that instance attribute such that if that joined federate were to invoke the *Update Attribute Values* service for that instance attribute, all of the preconditions for the invocation of the corresponding *Reflect Attribute Values* † service at ***F*** are satisfied. So, if an instance attribute is in scope for a given joined federate, that joined federate may receive invocations of the *Reflect Attribute Values †* service for that instance attribute. It is also possible, however, for a joined federate to receive invocations of the *Reflect Attribute Values †* service for an instance attribute that is not in scope for that joined federate. All of the conditions that define whether an instance attribute is in scope are also preconditions of the *Reflect Attribute Values †* service, with the exception of the condition that the instance attribute be owned by another joined federate. Because the conditions for determining whether an instance attribute is in scope are more restrictive than the conditions for determining whether an update to that instance attribute should be reflected, it is possible for updates to instance attributes that are not in scope to be reflected.

As an illustrative example, consider the scenario in which a joined federate that owns a particular instance attribute invokes the *Update Attribute Values* service for that instance attribute using a time stamp argument indicating a time far into the future. Then, that joined federate divests ownership of the instance attribute and resigns from the federation execution. No other joined federate acquires ownership of the instance attribute. When the time indicated in the time stamp argument of the update arrives, that instance attribute is, by definition, out of scope for all joined federates in the federation execution, because it is not owned. Nevertheless, assuming all of the other preconditions for invocation of the *Reflect Attribute Values †* service for that instance attribute are satisfied at a given joined federate, the *Reflect Attribute Values †* service will be invoked at that joined federate. Hence, an instance attribute being in or out of scope for a particular joined federate indicates only whether there is some other joined federate in the federation execution that is capable of generating updates to that instance attribute such that if that joined federate were to invoke the *Update Attribute Values* service for that instance attribute, all of the preconditions for the invocation of the corresponding *Reflect Attribute Values †* service at ***F*** are satisfied; an instance attribute being in or out of scope for a particular joined federate makes no assertion regarding whether invocations of the *Reflect Attribute Values †* service for that instance attribute may be invoked at that joined federate.

D.6.2 Reserve object instance name

Clause intentionally empty.

D.6.3 Object instance name reserved †

Clause intentionally empty.

D.6.4 Register object instance

Clause intentionally empty.

D.6.5 Discover object instance †

Clause intentionally empty.

D.6.6 Update attribute values

Clause intentionally empty.

D.6.7 Reflect attribute values †

Clause intentionally empty.

D.6.8 Send interaction

Clause intentionally empty.

D.6.9 Receive interaction †

Clause intentionally empty.

D.6.10 Delete object instance

Clause intentionally empty.

D.6.11 Remove object instance †

Clause intentionally empty.

D.6.12 Local delete object instance

Clause intentionally empty.

D.6.13 Change attribute transportation type

Actual implementation of the transportation types are RTI implementation dependent. The names may be declared in FOM/FDD.

D.6.14 Change interaction transportation type

Clause intentionally empty.

D.6.15 Attributes in scope †

Clause intentionally empty.

D.6.16 Attributes out of scope †

Clause intentionally empty.

D.6.17 Request attribute value update

A single *Request Attribute Value Update* service invocation for an object class may generate many *Provide Attribute Value Update* † service invocations at instance attribute owning joined federates (at least one for each object instance of the class for which the joined federate owns a requested attribute). This will generate a nonpredictable "storm" of "interrupts," possibly causing the timing behavior of real-time federates to become unpredictable. This might be an unwise thing to do in a real time federation.

D.7 Ownership management

Clause intentionally empty.

D.8 Time management

D.8.1 Overview

At the federation level, the HLA interface specification provides a variety of time management services to control the advancement of all joined federates along the HLA time axis. These services allow multiple joined federates with differing internal time advancement strategies to interoperate in a meaningful way. While the strategy chosen for advancing time is an extremely important consideration in the design of a simulation, the format of how time is represented is of equal importance.

It is very important for all federation members to agree upon a common set of key time characteristics that apply across the full federation. In addition to defining certain characteristics of the HLA time axis, it is also important to define the lookahead characteristics of both federates and federations. It is important to recognize that there are semantic differences between the way time is represented for the purpose of advancing time versus calculating lookahead. When advancing time, time can be considered to be an absolute value on a timeline, and thus time comparisons can be drawn to determine if one time value is greater than another. Lookahead, in contrast, represents a duration of time, that can be added to absolute time values but is not generally used for comparison purposes. For this reason, time is defined using two Abstract Datatypes (ADTs), LogicalTime and LogicalTimeInterval.

Each of the specific language APIs contain ADTs for LogicalTime and LogicalTimeInterval. These ADTs each define a set of interfaces for which implementations must be provided before a federation execution that utilizes time management services can take place. It is the responsibility of the federate/federation developer(s) to furnish concrete implementations of LogicalTime and LogicalTimeInterval interfaces.

Additionally, each of the language APIs also provide default concrete implementations of these two ADTs based on the float 64-datatype.

D.8.1.1 Messages

While the specification states no explicit requirements on the order of delivery of RO messages, an RTI implementation may provide a more predictable order of delivery. A number of factors must be considered by an RTI when determining the order in which RO messages are received, including the following:

— Whether or not a series of messages sent from the same federate are delivered in the same order as the order in which they were sent
— Whether or not series of messages sent by different federates are interleaved
— Whether or not messages received as RO that were sent as RO and that were sent as TSO are interleaved

A federate that assumes a particular ordering of RO messages may not be able to freely choose among RTI implementations.

A federate may choose to send TSO messages using different transportation types, but if the delivery mechanism is unreliable, there may be circumstances in which the message must be delivered to a receiving, time-constrained federate as RO. Time synchronization mechanisms typically rely on reliable message delivery to efficiently advance along the HLA time axis and messages transmitted unreliably may be delivered with a time stamp that would violate causality for a receiving federate. Under these circumstances an RTI must deliver the TSO message as RO.

D.9 Data distribution management

D.9.1 Overview

Clause intentionally empty.

D.9.1.1 Definitions for data distribution management

Clause intentionally empty.

D.9.1.2 Relating regions specifications and region realizations

Clause intentionally empty.

D.9.1.3 Using regions

Clause intentionally empty.

D.9.1.4 Calculating region overlaps

Clause intentionally empty.

D.9.1.5 Reinterpretation of selected declaration management services

See D.5.1.2 of this Annex for an explanation of item (o) when regions are not being used for update of an instance attribute or used for subscription of a class attribute. The following example is intended to explain item (o) of section 9.1.1 when regions are being used for update of an instance attribute or used for subscription of a class attribute.

Consider the same object class inheritance tree as was pictured in Figure D.1. We will use the following notation in addition to that which was introduced in D.5.1.2 of this Annex:

— **Subscribe_w_Reg[C,R1,(x,z)]** denotes the invocation of the *Subscribe Object Class Attributes With Regions* service to subscribe to object class C and class attributes x and z, and to use region R1 for subscription of class attributes x and z.

— **Register_w_Reg[E, (R1,x), (R2,y),e1]** denotes the invocation of the *Register Object Instance With Regions* service to register an object instance of class E that will be denoted e1, and to use region R1 for update of instance attribute e1(x) and region R1 for update of instance attribute e1(y).

Suppose that all class attributes in this example have the same set of available dimensions.

Suppose regions R1, R2, and R3 have been created and each of these regions contains all of the available dimensions of the class attributes in this example.

Suppose that

— Regions R1 and R2 overlap

— Regions R1 and R3 do not overlap

Suppose federate ***H*** invokes

— Subscribe_w_Reg [C,R1,(x,z,t)]

— Subscribe_w_Reg [A,R1,(y)]

Suppose federate ***G*** invokes

— Publish [E(x,y,z,t,u)]

— Register_w_Reg [E, (R2,x),(R2,y),(R3,z),e1]

— Then

Ownership: Federate ***G*** now owns e1(x), e1(y), e1(z), e1(t), and e1(u). Region R2 is used for update of e1(x) and e1(y) by federate ***G***; region R3 is used for update of e1(z) by federate ***G***; and the default region is used for update of e1(t) and e1(u) by federate ***G***. By definition, R1, R2, and R3 all overlap with the region used for update of e1(t) and e1(u).

Discovery at federate ***H***: Federate ***H*** will discover object instance e1 as a class C object instance. The candidate discovery class of e1 at federate ***H*** is class C, because federate ***H*** is not subscribed to e1's registered class (class E), and class C is the closest superclass of e1's registered class to which federate ***H*** is subscribed. Federate ***H*** is also subscribed to class attribute x at class C with region R1, and class attribute x has a corresponding instance attribute that is part of object instance e1 and that is owned (namely e1(x)), region R2 is used for update of e1(x) by its owning federate (federate ***G***), and regions R1 and R2 overlap. Therefore, the known class of e1 at federate ***H*** is class C.

Reflection: When federate ***G*** updates the value of e1(x) or e1(t), these updates shall be reflected at federate ***H***. Updates to e1(x) and e1(t) will be reflected at federate ***H*** because federate ***H*** is subscribed to class attributes x and t at e1's known class (class C), and the region that is used for subscription of x and t by federate ***H*** (region R1) overlaps the region that is used for updates of e1(x) and e1(t) by owning federate ***G*** (namely, region R2 in the case of e1(x) and the default region of routing space R in the case of e1(t)). However, when federate ***G*** updates the value of e1(y), e1(z), or e1(u), these updates will not be reflected at federate ***H***. Updates to e1(y) and e1(u) will not be reflected at federate ***H*** because federate ***H*** is not subscribed to class attribute y at object instance

e1's known class (class C) and class attribute u is not even available for subscription at object instance e1's known class (class C). Federate ***H*** is subscribed with region to class attribute z at e1's known class, but updates to e1(z) will not be reflected at federate ***H*** because federate ***H*** is using region R1 for subscription of class attribute z and federate ***G*** is using region R3 for update of instance attribute e1(z), and these two regions (R1 and R3) do not overlap.

D.9.2 Create region

Clause intentionally empty.

D.9.3 Commit region modifications

Clause intentionally empty.

D.9.4 Delete region

Clause intentionally empty.

D.9.5 Register object instance with regions

Clause intentionally empty.

D.9.6 Associate region for updates

Clause intentionally empty.

D.9.7 Unassociate region for updates

Clause intentionally empty.

D.9.8 Subscribe object class attributes with regions

Clause intentionally empty.

D.9.9 Unsubscribe object class attributes with regions

Clause intentionally empty.

D.9.10 Subscribe interaction class with regions

Clause intentionally empty.

D.9.11 Unsubscribe interaction class with regions

Clause intentionally empty.

D.9.12 Send interaction with regions

Clause intentionally empty.

D.9.13 Request attribute value update with regions

Unlike the corresponding object management service (*Request Attribute Value Update*), this service does not allow an object instance to be used as an argument; only an object class can be used. If an update for attributes of a specific object instance are needed, the object management service should be used. As a request that takes an object instance is already quite specific, additional DDM filtering based on regions are not needed for such cases.

A single *Request Attribute Value Update With Regions* service invocation for an object class may generate many *Provide Attribute Value Update* † service invocations at instance attribute owning joined federates (at least one for each object instance of the class for which the joined federate owns a requested attribute). This will generate a nonpredictable "storm" of "interrupts," possibly causing the timing behavior of real-time federates to become unpredictable. This might be an unwise thing to do in a real time federation.

D.10 Support services

Clause intentionally empty.

D.11 Management object model (MOM)

The defined MOM functionality is necessary for successful federation execution operation and can be provided only by the RTI. The MOM definition needs to be standardized by defining it in this specification.

Remotely changing joined federate states (publication, transportation types, time management, etc.) as provided the MOM can be error prone. However, these sorts of actions are needed in federations to recover from catastrophic error conditions. They should be used with extreme caution.

Additionally, there are ways a federate can ascertain that MOM has done something "on its behalf." The "target" federate can subscribe to **ModifyAttributeState** and all **Manager.Federate.Service** interactions and inspect them to determine if another federates are attempting to change its owned attribute state or invoke HLA services on its behalf. The only issue is that the RTI may react to the MOM interaction before the affected federate receives the particular notifying interaction.

D.11.1 Overview

Clause intentionally empty.

D.11.2 MOM object classes

Clause intentionally empty.

D.11.3 MOM interaction classes

Clause intentionally empty.

D.11.4 MOM related characteristics of the RTI

Clause intentionally empty.

D.11.5 Service-reporting

Clause intentionally empty.

D.11.6 MOM OMT tables

The value of the HLAfederateType class attribute of the object class HLAmanager.HLAfederate is actually a joined federate type, since the federate “indicator” argument to the *Join Federation Execution* service is a federate type and not a federate name.

Annex E

(informative)

Bibliography

[B1] *"Ada 95 Quality and Style: Guidelines for Professional Programmers"* SPC-94093-CMC, The United States Department of Defense Ada Joint Program Office.

[B2] *"Ada 95 Rationale—The Language—The Standard Libraries,"* Intermetrics, Inc., 1994, 1995

[B3] *"Annotated Ada 95 Reference Manual"* (AARM).

[B4] Austern, Matthew H., Generic Programming and the STL, Using and Extending the C++ Standard Template Library. Reading, MA: Addison-Wesley, 1999.

[B5] Harel, David, "Statecharts: A Visual Formalism for Complex Systems," *Science of Computer Programming* (Netherlands) *8*, *3*, pp. 231-274, June 1987.

[B6] IEEE 100, The Authoritative Dictionary of IEEE Standards Terms, Seventh Edition.

[B7] ISO/IEC 8652:1995: *"Information Technology—Programming Languages – Ada."* Copyright 1992,1993,1994,1995, Intermetrics, Inc.

[B8] ISO/IEC 14882:1998, *"Programming Languages—C++,"* date of approval: 7/27/1998, American National Standards Institute.

[B9] Meyers, Scott, Effective C++: 50 Specific Ways to Improve Your Programs and Designs, 2nd ed. Reading, MA: Addison-Wesley, 1997.

[B10] Stroustrup, Bjarne, The C++ Programming Language, 3rd ed. Reading, MA: Addison-Wesley, 1997.

Index

