

# DoD Interpretations of the High Level Architecture Interface Specification, Version 1.3: Release 3

## Introduction

Some areas of the High Level Architecture Interface Specification, Version 1.3, are not as well-specified as they could be and as a result may have questionable interpretations. Consequently, as a guide to RTI developers and users, we offer the following U.S. Department of Defense interpretations of ambiguous portions of the HLA Interface Specification, Version 1.3.

## How to use this document

This document sets forth the current release (Release 3) of the U.S. DoD Interpretations of Version 1.3 of the HLA Interface Specification. As the DoD gains experience using the HLA, this document is expected to grow and change. In particular, additional interpretations may be added or existing interpretations may be deleted as deemed necessary. Any time interpretations are added or deleted, a revised DoD Interpretations document, which has a unique number (e.g. Release N), will be released.

RTI implementations that are verified as compliant to the HLA Interface Specification, Version 1.3 are verified as compliant not only to the HLA Interface Specification Version 1.3, but also to a particular release of the DoD Interpretations of the HLA Interface Specification Version 1.3. All RTIs will be verified as compliant to the latest release of the DoD Interpretations for 1.3. The only exception to this rule would be certification of an RTI that began verification testing before the latest release of the DoD Interpretations was made available. In this case, an RTI can be verified to the previous release of the DoD Interpretations if the RTI developer so wishes.

All previous releases of the DoD Interpretations are well-defined and can be constructed from the current release of the DoD Interpretations using the pedigree feature that is associated with each individual interpretation in the DoD Interpretations document. The pedigree feature is a label that appears in square brackets [ ] at the end of each individual interpretation below. It has the form [New in Release x] or [New in Release x; Deleted in Release x+2]. The pedigree indicates in which release the interpretation was introduced ("New in") and, if appropriate, in which release the interpretation was deleted ("Deleted in"). An interpretation that has a pedigree of the form [New in Release x], for example, is present in not only release x of the DoD Interpretations document, but in all subsequent releases of the DoD Interpretations document up to and including the current release. This interpretation is not present, however, in any release of the DoD Interpretations Document preceding release x. An interpretation that has a pedigree of the form [New in Release x; Deleted in Release x+2] is present in only releases x and x+1 of the DoD Interpretations document.

## **Service 4.3 : Destroy Federation Execution**

### ***Interpretation 1***

The Destroy Federation Execution service may be invoked if there are unowned instance attributes (instance attributes that are not owned by any federate in the federation execution or by the RTI) in the federation execution. [New in Release 2]

## **Service 4.6: Register Federation Synchronization Point**

### ***Interpretation 1***

The text of the Register Synchronization Point service states in the preconditions that "If an optional set of federate designators is supplied, those federates must be joined to the federation execution". The introductory text to this service says that the synchronization label used must be unique. However, if this service is invoked with a set of joined federates that includes a federate that is not joined to the federation execution, or if this service is invoked with a synchronization point label that is already in use, an exception will not be invoked.

When a federate invokes the Register Federation Synchronization Point service and supplies a label and a set of federate designators as argument, the federate has no way of knowing if the label is in use or if those federates are actually joined to the federation execution at that time. So, if the label is in use or if a federate that is not joined is included in the set, the service will execute successfully. However, the invoking federate shall receive a Confirm Synchronization Point Registration callback (service 4.7) with a registration success indicator of failure. [New in Release 2]

### ***Interpretation 2***

Once a synchronization has completed, a federate may use the same label for another synchronization. [New in Release 2]

## **Service 4.11: Request Federation Save**

### ***Interpretation 1***

The 1.3 Interface Specification is ambiguous with regard to when the RTI is expected to instruct each time-constrained federate to save state. Service 4.11, Request Federation Save, says:

*...If the optional federation time argument is present, the RTI shall instruct each time-constrained federate to save state when its value of logical time advances to the value provided; and shall instruct non-time-constrained federates to save state when the last time-constrained federate's value of logical time advances to the value of the optional federation save time provided....*

It is unclear what is meant by "when its value of logical time advances to the value provided".

This service is intended to work as follows:

*When a save has been scheduled, each time-constrained federate shall be instructed to save state (via the Initiate Federate Save service) as soon as (1) the federate has received all messages that it will receive as TSO messages that have time stamps less than or equal to the scheduled save time, and either (2) the federate is in the time advancing state as a result of invocation of either the Time Advance Request Available (TARA), Next Event Request Available (NERA), or Flush Queue Request (FQR) services and the logical time of the next grant is greater than the time stamp of the scheduled save or (3) the joined federate is in the time advancing state as a result of invocation of either the Time Advance Request (TAR) or Next Event Request (NER) services and the logical time of the next grant is greater than or equal to the time stamp of the scheduled save. [New in Release 1]*

### ***Interpretation 2***

Once a federation save has completed, a federate may use the same label for another federation save, but this is not recommended.

*Rationale: Although the Interface Specification does not say that save labels shall not be reused, it is recommended that federates not re-use federation save labels. If federation save labels were to be reused, this could result in unexpected results. [New in Release 2]*

## **Service 6.2: Register Object Instance**

### ***Interpretation 1***

If a federate has registered an object instance with a name and sometime later deletes it so that the object instance no longer exists, a federate may register a new object instance with this same name, however, this is not recommended.

*Rationale: Although the Interface Specification does not say that object instance names shall not be re-used, it is recommended object instance names not be re-used. If object instance names were to be reused, this could result in unexpected results. [New in Release 2]*

## **Services 6.8 and 6.9: Delete Object instance and Remove Object Instance**

### ***Interpretation 1***

The text for the Delete object Instance service states that "The RTI shall use the Remove Object Instance service to inform the reflecting federates that the object instance has been deleted." This sentence should say instead, "The RTI shall use the Remove Object Instance service to inform all other joined federates that know the object instance that the object instance has been deleted."

*Rationale: It is not necessary that a federate be currently subscribed to any of the corresponding class attributes of an object instance in order to receive a Remove Object Instance callback for that object instance. The pre-condition for receipt of the Remove Object Instance callback stated in the Interface Specification is correct: the federate must know about the object instance. [New in Release 2]*

## **Service 6.13: Attributes In Scope**

### ***Interpretation 1***

The fourth pre-condition of the Attributes In Scope service states simply that "the federate is subscribed to the class attributes". This pre-condition is not complete. It should say, "the federate is subscribed to the class attributes at the known class of the object instance at this federate". [New in Release 2]

## **Service 6.14: Attributes Out Of Scope**

The third pre-condition of the Attributes Out Of Scope service states that:

*At least one of the following is not true:*

*The federate knows about the object instance with the specified designator.*

*The federate is subscribed to the class attributes.*

*The federate does not own the instance attributes.*

*If there are regions involved, they overlap, see 9.1, Overview.*

This pre-condition is not correct in several respects. Instead, the following three interpretations apply:

### ***Interpretation 1***

A federate must know about an object instance in order to receive an Attributes Out Of Scope callback for any instance attributes of that object instance. If a federate does not know about an object instance, it shall not receive an Attributes Out Of Scope callback for any instance attributes of that object instance. [New in Release 2]

### ***Interpretation 2***

A federate must be subscribed to an instance attribute's corresponding class attribute at the known class of the object instance in order to receive an Attributes Out Of Scope callback for that instance attribute. If a federate is not subscribed to an instance attribute's corresponding class attribute at the known class of the object instance, it shall not receive an Attributes Out Of Scope callback for that instance attribute. If an instance attribute is in scope for a federate and that federate unsubscribes that instance attribute's corresponding class attribute at the known class of the object instance, then the instance attribute will no longer be in scope for the federate. However, the federate shall not receive an Attributes Out Of Scope callback for such an instance attribute that goes out of scope as a result of this unsubscribe. This interpretation is consistent with Figure 10, the "Implications of Ownership of Instance Attribute (i)" statechart. [New in Release 2]

### ***Interpretation 3***

A federate must not own an instance attribute in order to receive an Attributes Out Of Scope callback for that instance attribute. If a federate owns an instance attribute, it shall not receive an Attributes Out Of Scope callback for that instance attribute. If an instance attribute is in scope for a federate and that federate becomes the owner of the instance attribute, then the instance attribute will no longer be in scope for the federate. However, the federate shall not receive an Attributes Out Of Scope callback for such an instance attribute that goes out of scope as a result of the federate becoming its owner. This

interpretation is consistent with Figure 10, the "Implications of Ownership of Instance Attribute (i)" statechart. [New in Release 2]

## **Service 6.15: Request Attribute Value Update**

### ***Interpretation 1***

The 1.3 spec says that if the Request Attribute Value Update service is invoked with an object class as parameter, "the RTI shall solicit the values of the specified instance attributes for all the object instances of that class." It is not at all clear what is meant by the phrase *an object instance of that class*: An object instance registered at that class? Known at that class? Discovered at that class? The DOD interpretation is the following: 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 or at some subclass of that class. [New in Release 1]

## **Service 6.17: Turn updates On For Object Instance service**

### ***Interpretation 1***

The last pre-condition of the Turn Updates On For Object Instance service states that, "Some other federate in the execution is actively subscribed to the attributes of the object class". This pre-condition is not completely accurate. This pre-condition should instead say, "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." [New in Release 2; Deleted in Release 3]

### ***Interpretation 2***

The last pre-condition of the Turn Updates On For Object Instance service states that, "Some other federate in the execution is actively subscribed to the attributes of the object class". This pre-condition is not completely accurate. This pre-condition should instead say, "For each attribute designator specified, there is at least one joined federate in the federation execution that is actively subscribed to the attribute at either the registered class of the object instance or at a superclass of the registered class." [New in Release 3]

## **Service 7.2: Unconditional Attribute Ownership Divestiture and other services that may result in instance attributes becoming unowned**

### ***Interpretation 1***

When an attribute becomes unowned, either by a federate invoking the Unconditional Attribute Ownership Divestiture service, the Publish Object Class service, the Unpublish Object Class service, or the Resign Federation Execution service, if no federates are in either the "Acquiring" or "Willing to Acquire" state with respect to that instance attribute, the RTI is expected to offer ownership of the attribute to all eligible federates (excepting the federate that just released ownership of the attribute). The RTI shall offer ownership of this attribute by invoking the Request Attribute Ownership Assumption service at all federates that meet the preconditions for invocation of the Request Attribute Ownership Assumption service, except for the federate that just released ownership of the attributes. [New in Release 2]

## **Service 7.7: Attribute Ownership Acquisition**

### ***Interpretation 1***

If a federate invokes the Attribute Ownership Acquisition service for an instance attribute that is in the Acquisition Pending state, that instance attribute's state will remain unchanged. In other words, if a federate invokes the Attribute Ownership Acquisition service for an instance attribute that is in the Acquiring state, that instance attribute shall continue to be in the Acquiring state. Likewise, if a federate invokes the Attribute Ownership Acquisition service for an instance attribute that is in the Trying to Cancel Acquisition state, that instance attribute shall continue to be in the Trying to Cancel Acquisition state. [New in Release 2]

## **Service 7.16: Inform Attribute Ownership**

### ***Interpretation 1***

There are three possible ownership designator values in the Inform Attribute Ownership callback: attribute owned by a particular federate, attribute owned by the RTI, and attribute unowned. Unowned instance attributes are those attributes that are not owned by either a federate or the RTI. Examples of unowned instance attributes are attributes that have been unconditionally divested and have not subsequently become owned by any federate. Instance attributes that are owned by the RTI are instance attributes of Management Object Model (MOM) object instances. For example, there is a MOM object class `Manager.Federate` and one of its available attributes is `FederateHandle`. The RTI publishes object class `Manager.Federate` and class attribute `FederateHandle` (as well as other class attributes) at this class, and the RTI registers one object instance of this class for each federate in a federation execution. The RTI owns the `FederateHandle` instance attribute of each of these `Manager.Federate` object instances. (See section 11.1.2) [New in Release 2]

## **Service 8.2: Enable Time Regulation**

### ***Interpretation 1***

In response to invocation of the Enable Time Regulation service, the RTI shall indicate the logical time that is assigned to the federate through the Time Regulation Enabled service. According to the service description for the Enable Time Regulation service in the 1.3 Interface Specification, the logical time that is assigned shall be greater than or equal to that requested by the federate. Actually, the logical time that is provided when time regulation is enabled shall be the smallest possible logical time that is greater than or equal to that requested by the federate. This will enable the creation of federation executions that join all federates and make them time-regulating and time-constrained at time zero before beginning to advance time at all. [New in Release 1]

## **Service 8.3: Time Regulation Enabled**

### ***Interpretation 1***

The second post-condition of this service says, "If the federate is time-constrained, no additional TSO messages shall be delivered with time stamps less than or equal to the provided time." This post-condition should instead say, "If the federate is time-

constrained, no additional TSO messages shall be delivered with time stamps less than the provided time." [New in Release 2]

## **Service 8.10: Next Event Request**

### ***Interpretation 1***

The Application Programmer's Interfaces (APIs) for the Next Event Request service should have a Federation Time Is Invalid exception defined. (It is defined in the Interface Specification, but not in all APIs.) [New in Release 1]

## **Service 8.21: Retract**

### ***Interpretation 1***

The result of retracting a delete is undefined and will not be tested. [New in Release 1]

### ***Interpretation 2***

This service description states that, "Only messages sent in TSO may be retracted. A federate may not retract messages in its past. A message shall be in a federate's past if its time is earlier than the federate's current logical time."

This statement is not complete because it does not take the federate's lookahead into account, nor does it distinguish between federates that are in the Time Granted state and those that are in the Time Advancing state. As such, this statement implies that it would be permissible for a federate to retract a TSO message that has a timestamp with a value between the federate's logical time and the federate's logical time plus the federate's lookahead. However, it is not permissible for a federate to retract a message with such a time stamp.

A message may only be retracted if:

- the retracting joined federate sent the message,
- the message had a sent order type of TSO,
- the joined federate is time-regulating, and
- either:
  - 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
  - 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. [New in Release 2]

## **Service 9.6: Associate Region For Updates**

### ***Interpretation 1***

A federate may associate a region for update with an instance attribute before becoming an owner of that instance attribute. [New in Release 2]

### ***Interpretation 2***

A given instance attribute of a given object instance may be associated with multiple regions. [New in Release 2]

## **Service 10.16: Get Attribute Routing Space Handle**

### ***Interpretation 1***

The Get Attribute Routing Space Handle service should have a precondition and an associated exception that prevents it from being invoked with an attribute that does not have an associated routing space in the FED file. [New in Release 1]

## **Management Object Model (MOM): General**

### ***Interpretation 1***

The Management Object Model uses a number of types (Boolean, string, enumerated), without indicating how these types are represented. When a federate reflects values of MOM instance attributes, the values shall be interpreted as ASCII text, suffixed with a trailing NUL. Similarly, when a federate receives parameter values as part of MOM interactions, the values shall be interpreted as ASCII text, suffixed with a trailing NUL; when a federate supplies parameter values as part of a MOM interaction, the values shall be interpreted as ASCII text, suffixed with a trailing NUL. Specifically:

- Handles shall be represented as small integers (in the range 0 to  $2^{32} - 1$ ) encoded as base 10, unsigned. In lists of handles, the handles shall be separated by commas.
- Names shall be encoded as ASCII strings.
- Booleans shall be encoded as “true” or “false”.
- Enumerations shall be encoded as the literal name of the enumeration value as defined in the attribute tables or interaction tables as text.
- There are two cases for how time and interval values shall be encoded:
  - Time and interval values appearing as arguments in service calls that are callbacks from the RTI to the federate shall be represented in some printable form that is produced by the appropriate method in the time class implementation. For example, in Java, the text representation is generated using the toString method; in C++, it is generated using the getPrintableString method.
  - Time and interval values sent as parameters in Manager.Federate.Service.SOMESERVICE interactions are created using the encode and decode methods of the time class implementation.
- Attributes defined as type long shall be encoded as base 10, unsigned.

[New in Release 1]

### ***Interpretation 2***

In the case in which a MOM interaction is sent that is missing a parameter, the federate that sent the offending MOM interaction should receive a Manager.Federate.Report.Alert interaction with an appropriate alert description, and the MOM interaction should be rejected (that is, not acted upon). It should have no affect. [New in Release 1]

### ***Interpretation 3***

In the case in which a MOM Manager.Federate.Service.SOMESERVICE interaction is sent on behalf of a federate when the preconditions for invoking SOMESERVICE at that

federate are not met, the federate that sent the offending MOM interaction should receive a Manager.Federate.Report.Alert interaction with an appropriate alert description, and the MOM interaction should be rejected (that is, not acted upon). It should have no affect.

In other words, the Manager.Federate.Report.Alert should be sent by the RTI to a federate in response to that federate sending an erroneous MOM Manager.Federate.Service interaction. A Manager.Federate.Report.Alert interaction should always be sent when there is such an “exceptional” condition. There is no way to turn the generation of such Manager.Federate.Report.Alert interactions on and off. [New in Release 1]

#### ***Interpretation 4***

There are many unexpected things that can happen if the MOM services are used carelessly. For example, a federate can lose ownership of instance attributes (see the 11.2.1.2: Manager.Federate.Adjust.ModifyAttributeState interpretation), and a federate can be resigned (see the 11.2.4.1:Manager.Federate.Service.ResignFederationExecution interpretation). MOM must be used with caution. [New in Release 2]

### **11.2.1: Manager.Federate.Adjust**

#### ***Interpretation 1***

A Manager.Federate.Adjust interaction that is sent by a federate shall be received by the RTI. If other federates are subscribed to the sent interaction class or to a superclass of the sent interaction class, the interaction shall also be received by those subscribed federates. [New in Release 2]

#### **11.2.1.2: Manager.Federate.Adjust.ModifyAttributeState**

##### ***Interpretation 1***

If the Manager.Federate.Adjust.ModifyAttributeState interaction is used to try to cause an instance attribute that is already owned by some federate to become owned by another federate, a Manager.Federate.Report.Alert interaction shall be sent to the federate that sent the Manager.Federate.Adjust.ModifyAttributeState interaction, and no change of ownership shall occur. If the instance attribute is unowned, its ownership shall be changed.

As a result of a federate sending the MOM Manager.Federate.Adjust.ModifyAttributeState interaction, the RTI is not supposed to generate either an Attribute Ownership Acquisition Notification or an Attribute Ownership Divestiture Notification. However the RTI should generate discoveries and scope advisories if they are warranted. [New in Release 1; Deleted in Release 2]

##### ***Interpretation 2***

If the Manager.Federate.Adjust.ModifyAttributeState interaction is used to try to cause an instance attribute that is already owned by some federate to become owned by another federate, a change of ownership shall occur.

As a result of a federate sending the MOM `Manager.Federate.Adjust.ModifyAttributeState` interaction, the RTI is not supposed to generate either an Attribute Ownership Acquisition Notification or an Attribute Ownership Divestiture Notification. However, the RTI should generate discoveries and scope advisories if they are warranted.

Depending on the implementation, use of the `Manager.Federate.Adjust.ModifyAttributeState` interaction could result in more than one federate being granted ownership of an instance attribute, a federate that owns an instance attribute losing ownership of that instance attribute without its knowledge or consent, a federate being granted ownership of an instance attribute without its knowledge or consent, and other unexpected behavior in a federation execution. Hence, MOM services should be used with caution. [New in Release 2]

### **11.2.2.10: Manager.Federate.Request.RequestObjectInformation**

#### *Interpretation 1*

When a `Manager.Federate.Request.RequestObjectInformation` interaction is sent for a federate that does not know the object instance. The corresponding report interaction should contain the federate handle, the object instance, and null strings for the remaining parameters. [New in Release 1]

### **11.2.3: Manager.Federate.Report**

#### *Interpretation 1*

As with all interactions, an interaction of class `MOM Manager.Federate.Report` or of a subclass of class `Manager.Federate.Report` that is sent by the RTI shall be received only by federates that are subscribed to the sent interaction class or to a superclass of the sent interaction class. [New in Release 2]

### **11.2.3.5: Manager.Federate.Report.ReportObjectsOwned**

#### *Interpretation 1*

The formatting of the `ObjectCounts` list in the `Manager.Federate.Report.ReportObjectsOwned` interaction should contain handle/class pairs for only those classes that have counts that are greater than zero, not for all classes. So, for example, when the `MOM Manager.Federate.Report.ReportObjectsOwned` interaction is sent for a federate that does not own the `privilegeToDelete` attribute of any object instance, the `ObjectCounts` parameter in the interaction should be a null list (not a list that includes handle/0 count pairs for every object class). The same is true for all applicable other `Manager.Federate.Report` interactions, such as `ReportUpdatesSent`, `ReportInteractionsSent`, `ReportReflectionsReceived`, `ReportInteractionsReceived`, `ReportObjectsUpdated`, and `ReportObjectsReflected`. [New in Release 1]

### **11.2.3.8: MOM interaction subclass ReportUpdatesSent**

#### *Interpretation 1*

According to the specification, the `ReportUpdatesSent` interaction shall report the "number of updates sent (by object class) by the federate...". This wording is ambiguous

regarding the question of whether the number of updates is defined as the number of times the Update Attribute Values service was invoked by the federate for all object instances of a given object class, or the number of instance attributes that were updated by the federate for all object instances of a given object class.

The DoD interpretation is that the number of updates is defined as the number of instance attributes updated. That is, if a federate has invoked the Update Attribute Values service only once, and in this service invocation were arguments for an object instance of class A and  $n$  instance attributes of type reliable and  $m$  instance attributes of type best-effort, then in response to an interaction of class Manager.Federate.Request.RequestUpdatesSent being sent by a federate, two Manager.Federate.Report.ReportUpdatesSent interactions should be sent by the RTI: one for transportation type reliable with an update count of  $n$  and one for transportation type best-effort with an update count of  $m$ .

*Rationale: The update service is a service that acts on instance attributes, not on object instances. The fact that updates to several different instance attributes of an object instance can be bundled together in a single Update Attribute Values service invocation is provided as a convenience to the programmer. The value of an update count should not depend on whether or not a federate chooses to combine certain instance attribute value updates together in a single call or perform these updates as separate Update Attribute Values service invocations.*

*Furthermore, consider that there is a MOM interaction subclass called ReportReflectionsReceived (see 11.2.2.8 and 11.2.3.9), which is analogous to ReportUpdatesSent. It would seem reasonable that the update and reflect counters should be equal in the case in which the receiving federate actually received every update that was sent by the sending federate. According to the DoD interpretation of these counters, which is to count each instance attribute update individually, the update counter will always equal the reflect counter. However, according to an interpretation of these counters that counts the number of Update Attribute Values service invocations that were made, the update counter may not equal the reflect counter. Sometimes a single update service invocation will be broken into several reflect service invocations, making the reflect counter exceed the update counter. Specifically, suppose instance attributes X and Y of an object instance of object class A both have transportation types of reliable, but X has an ordering type of time stamp and Y has an ordering type of receive-order. If instance attributes X and Y are both updated in a single invocation of the Update Attribute Values service by a time-regulating federate that includes a time stamp argument in the service invocation, then according to the DoD interpretation, the reliable updates-sent counter will be incremented by 2. According to the alternate interpretation, it will be incremented by only 1. Now, at a time-constrained federate, this update will be received as two separate reflects, one for instance attribute x with a message ordering type of timestamp, and one for instance attribute y with a message ordering type of receive-order. Hence, according to both interpretations, the reliable reflects-received counter would be incremented by 2. The DoD interpretation is preferable in this case, insofar as it enables the updates-sent counter to equal the reflects-received counter. [New in Release 2]*

### **11.2.3.9: MOM interaction subclass ReportReflectionsReceived**

#### ***Interpretation 1***

According to the specification, the ReportReflectionsReceived interaction shall report the "number of reflections received (by object class) by the federate...". This wording is ambiguous regarding the question of whether the number of reflections is defined as the number of times the Reflect Attribute Values service was invoked at the federate for all object instances of a given object class, or the number of instance attributes that had values reflected by the federate for all object instances of a given object class.

The DoD interpretation is that the number of reflections is defined as the number of instance attributes reflected. That is, if a federate has received the Reflect Attribute Values service invocation only once, and in this service invocation were arguments for an object instance of class A and  $n$  instance attributes of type reliable and  $m$  instance attributes of type best-effort, then in response to an interaction of class Manager.Federate.Request.RequestReflectionsReceived, two Manager.Federate.Report.ReportReflectionsReceived interactions should be sent: one for transportation type reliable with a reflect count of  $n$  and one for transportation type best-effort with a reflect count of  $m$ .

*Rationale: As with the Update Attribute Values service, the Reflect Attribute Values service is a service that acts on instance attributes, not on object instances. The rationale for this interpretation is analogous to the rationale for interpretation 11.2.3.8. [New in Release 2]*