Lightweight Services Specification

This OMG document replaces the submission (ptc/2003-10-03). It is an OMG Final Adopted Specification, which has been approved by the OMG board and technical plenaries, and is currently in the finalization phase. Comments on the content of this document are welcomed, and should be directed to *issues@omg.org* by May 3, 2004.

You may view the pending issues for this specification from the OMG revision issues web page *http://www.omg.org/issues/*; however, at the time of this writing there were no pending issues.

The FTF Recommendation and Report for this specification will be published on November 21, 2004.

OMG Final Adopted Specification
ptc/04-02-03
February 2004

Date: February 2004

Lightweight Services

OMG Adopted Specification

ptc/04-02-03

USE OF SPECIFICATION - TERMS, CONDITIONS & NOTICES

The material in this document details an Object Management Group specification in accordance with the terms, conditions and notices set forth below. This document does not represent a commitment to implement any portion of this specification in any company's products. The information contained in this document is subject to change without notice.

LICENSES

The companies listed above have granted to the Object Management Group, Inc. (OMG) a nonexclusive, royalty-free, paid up, worldwide license to copy and distribute this document and to modify this document and distribute copies of the modified version. Each of the copyright holders listed above has agreed that no person shall be deemed to have infringed the copyright in the included material of any such copyright holder by reason of having used the specification set forth herein or having conformed any computer software to the specification.

Subject to all of the terms and conditions below, the owners of the copyright in this specification hereby grant you a fully-paid up, non-exclusive, nontransferable, perpetual, worldwide license (without the right to sublicense), to use this specification to create and distribute software and special purpose specifications that are based upon this specification, and to use, copy, and distribute this specification as provided under the Copyright Act; provided that: (1) both the copyright notice identified above and this permission notice appear on any copies of this specification; (2) the use of the specifications is for informational purposes and will not be copied or posted on any network computer or broadcast in any media and will not be otherwise resold or transferred for commercial purposes; and (3) no modifications are made to this specification. This limited permission automatically terminates without notice if you breach any of these terms or conditions. Upon termination, you will destroy immediately any copies of the specifications in your possession or control.

PATENTS

The attention of adopters is directed to the possibility that compliance with or adoption of OMG specifications may require use of an invention covered by patent rights. OMG shall not be responsible for identifying patents for which a license may be required by any OMG specification, or for conducting legal inquiries into the legal validity or scope of those patents that are brought to its attention. OMG specifications are prospective and advisory only. Prospective users are responsible for protecting themselves against liability for infringement of patents.

GENERAL USE RESTRICTIONS

Any unauthorized use of this specification may violate copyright laws, trademark laws, and communications regulations and statutes. This document contains information which is protected by copyright. All Rights Reserved. No part of this work covered by copyright herein may be reproduced or used in any form or by any means--graphic, electronic, or mechanical, including photocopying, recording, taping, or information storage and retrieval systems--without permission of the copyright owner.

DISCLAIMER OF WARRANTY

WHILE THIS PUBLICATION IS BELIEVED TO BE ACCURATE, IT IS PROVIDED "AS IS" AND MAY CONTAIN ERRORS OR MISPRINTS. THE OBJECT MANAGEMENT GROUP AND THE COMPANIES LISTED ABOVE MAKE NO WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, WITH REGARD TO THIS PUBLICATION, INCLUDING BUT NOT LIMITED TO ANY WARRANTY OF TITLE OR OWNERSHIP, IMPLIED WARRANTY OF MERCHANTABILITY OR WARRANTY OF FITNESS FOR A PARTICULAR PURPOSE OR USE. IN NO EVENT SHALL THE OBJECT MANAGEMENT GROUP OR ANY OF THE COMPANIES LISTED ABOVE BE LIABLE FOR ERRORS CONTAINED

HEREIN OR FOR DIRECT, INDIRECT, INCIDENTAL, SPECIAL, CONSEQUENTIAL, RELIANCE OR COVER DAMAGES, INCLUDING LOSS OF PROFITS, REVENUE, DATA OR USE, INCURRED BY ANY USER OR ANY THIRD PARTY IN CONNECTION WITH THE FURNISHING, PERFORMANCE, OR USE OF THIS MATERIAL, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

The entire risk as to the quality and performance of software developed using this specification is borne by you. This disclaimer of warranty constitutes an essential part of the license granted to you to use this specification.

RESTRICTED RIGHTS LEGEND

Use, duplication or disclosure by the U.S. Government is subject to the restrictions set forth in subparagraph (c) (1) (ii) of The Rights in Technical Data and Computer Software Clause at DFARS 252.227-7013 or in subparagraph (c)(1) and (2) of the Commercial Computer Software - Restricted Rights clauses at 48 C.F.R. 52.227-19 or as specified in 48 C.F.R. 227-7202-2 of the DoD F.A.R. Supplement and its successors, or as specified in 48 C.F.R. 12.212 of the Federal Acquisition Regulations and its successors, as applicable. The specification copyright owners are as indicated above and may be contacted through the Object Management Group, 250 First Avenue, Needham, MA 02494, U.S.A.

TRADEMARKS

The OMG Object Management Group Logo®, CORBA®, CORBA Academy®, The Information Brokerage®, XMI® and IIOP® are registered trademarks of the Object Management Group. OMG™, Object Management Group™, CORBA logos™, OMG Interface Definition Language (IDL)™, The Architecture of Choice for a Changing World™, CORBAservices™, CORBAfacilities™, CORBAmed™, CORBAnet™, Integrate 2002™, Middleware That's Everywhere™, UML™, Unified Modeling Language™, The UML Cube logo™, MOF™, CWM™, The CWM Logo™, Model Driven Architecture™, Model Driven Architecture Logos™, MDA™, OMG Model Driven Architecture™, OMG MDA™ and the XMI Logo™ are trademarks of the Object Management Group. All other products or company names mentioned are used for identification purposes only, and may be trademarks of their respective owners.

COMPLIANCE

The copyright holders listed above acknowledge that the Object Management Group (acting itself or through its designees) is and shall at all times be the sole entity that may authorize developers, suppliers and sellers of computer software to use certification marks, trademarks or other special designations to indicate compliance with these materials.

Software developed under the terms of this license may claim compliance or conformance with this specification if and only if the software compliance is of a nature fully matching the applicable compliance points as stated in the specification. Software developed only partially matching the applicable compliance points may claim only that the software was based on this specification, but may not claim compliance or conformance with this specification. In the event that testing suites are implemented or approved by Object Management Group, Inc., software developed using this specification may claim compliance or conformance with the specification only if the software satisfactorily completes the testing suites.

ISSUE REPORTING

All OMG specifications are subject to continuous review and improvement. As part of this process we encourage readers to report any ambiguities, inconsistencies, or inaccuracies they may find by completing the Issue Reporting Form listed on the main web page http://www.omg.org, under Documents & Specifications, Report a Bug/Issue.

Lightweight Services Table of Contents

1 Scope

This specification defines a compatible subset of three existing CORBA services to make these services suitable for use in resource-constrained systems. These subsets are intended to be inserted as new chapters in the Services documents that they produce the subset of. No other changes to the existing documents are being proposed. This specification defines the Lightweight Naming Service, the Lightweight Event Service, and the Lightweight Time Services.

The services defined by this specificaton are fully upward compatible with the corresponding full-featured services. A better way of looking at it, is to view the “Heavyweight” services as extensions of the lightweight ones. This approach would be much cleaner, but would require edits to these “Heavyweight” specs to make that clarification. Using the extension approach would readily allow specific functions to be removed from interfaces if necessary without any requirement for a **NOT_IMPLEMENTED** exception. Without permission to “merge” versus “insert” the lightweight chapters in to the heavyweight specs, the subset solution presented here must be used.

Semantics

Operations that are termed “disabled” in these conformance points are still part of the associated IDL interface, but implementations may raise either **BAD_OPERATION** or **NO_IMPLEMENT** exceptions when they are invoked. This flexibility allows the lightweight services to avoid extra overhead in the service implementation skeletons and removes any requirement for clients to test explicitly for disabled operations. In cases where the operations is termed optional, **NO_IMPLEMENT** is preferred over **BAD_OPERATION**. However, the Lightweight Service implementer may use **BAD_OPERATION** for the optional interfaces to meet the constraints of their embedded system.

The semantics of “disabled” interfaces with respect to lightweight services is further intended to be consistant with all other OMG Lightweight specifications.

For convenience, in this specification, only the operations that are not disabled are shown in the informative IDL descriptions of these services.

The IDL specifications in the “full” service specifications continue to be the normative definition for each interface.

2 Conformance

2.1 Summary of optional versus mandatory interfaces

All interfaces are mandatory within the compliance points.

2.2 Proposed major conformance points

Each individual service defined in this specification represents an independent item. Each service therefore forms an independent major compliance point:

- Lightweight Naming Service
- Lightweight Event Service
- Lightweight Time Service

2.2.1 Proposed minor conformance points

The Lightweight Time Service defined in this specification supports two optional conformance points:

- Support of multiple clocks
- Support of periodic execution control

3 Normative References

3.1 UML Specifications

3.1.1 UML Language Specification

Unified Modeling Language (UML) Specification Version 1.5
Formal OMG Specification, document number: formal/2003-03-01
The Object Management Group, March 2003
[http://www.omg.org]

Note – The following specifications might become formal before finalization of this Lightweigth Services specification is complete. Unless these documents become formal OMG specifications, their reference is *not normative.*

UML Version 2.0 Infrastructure Specification
final submission (conveniance document), document number: ad/2003-03-01

UML Version 2.0 Superstructure Specification
final adopted specification, document number: ptc/2003-08-02

3.1.2 UML Profile for CORBA Specification

UML Profile for CORBA Specification V1.0
Formal OMG Specification, document number: formal/2002-04-01
The Object Management Group, April 2002
[http://www.omg.org]

3.2 CORBA Core Specifications

3.2.1 CORBA Specification

Common Object Request Broker (CORBA/IIOP), version 3.0.2
Formal OMG Specification, document number: formal/2002-12-06
The Object Management Group, December 2002
[http://www.omg.org]

3.2.2 Minimum CORBA Specification

Minimum CORBA, V1.0
Formal OMG Specification, document number: formal/2002-08-01
The Object Management Group, August 2002
[http://www.omg.org]

3.3 CORBA Services Specifications

3.3.1 Naming Service Specification

Naming Service, version 1.2
Formal OMG Specification, document number: formal/2002-09-02
The Object Management Group, September 2002
[http://www.omg.org]

3.3.2 Event Service Specification

Event Service, version 1.1
Formal OMG Specification, document number: formal/2001-03-01
The Object Management Group, March 2001
[http://www.omg.org]

3.3.3 Enahanched View of Time Specification

Enhanched View of Time Service, version 1.1
Formal OMG Specification, document number: formal/2002-05-07
The Object Management Group, May 2002
[http://www.omg.org]

3.3.4 Property Service Specification

Property Service, version1.0
Formal OMG Specification, document number: formal/2000-06-22
The Object Management Group, June 2000
[http://www.omg.org]

4 Terms and Definitions

NOTE: Needs to be completed by the FTF (or possibly eliminated).

5 Symbols

List of symbols/abbreviations.

NOTE: Needs to be completed by the FTF (or possibly eliminated).

6 Additional Information

6.1 Changes to Adopted OMG Specifications

The specifications contained in this document require no changes to adopted OMG specifications.

Note – The submitters recommend a document merge of the specifications contained in this document with the specifications of the corresponding full-featured services to guarantee consistency even under potential future revisions. In particular the submitters strongly suggest to use the lightweight services as base services and to redefine the full-featured services as specialization of the lightweight services.

6.2 How to Read this Specification

The rest of this document contains the technical specification. We recommend that the reader is familar with the Unified Modeling Language(UML) as defined in the UML Infrastructure and UML Superstructure specifications. It is further required that the reader is familiar with the specifications of the corresponding full-featured versions of the services, since the lightweight service definitions contained in this document will make frequent references to the specifications of the full-featured services.

A knowledge of the particular techincal challences imposed by resource-constraint systems would be of great benefit to understand the design decisions made during the derivation of the lightweight services from their full-featured counterparts.

6.3 Acknowledgements

The following companies are pleased to co-submit the Specification for Lightweight Services in response to the Request For Proposal "Lightweight Services" issued by the OMG Realtime, Embedded and Specialized Systems Platform Task Force as OMG document realtime/2002-06-31.

- Mercury Computer Systems, Inc.
- Objective Interface Systems, Inc.
- Rockwell Collins, Inc.

The following companies are pleased to support the Specification for Lightwight Services as submitted by the submitting companies listed above.

- Raytheon Company
- MITRE Corporation
- BAE Systems
- ITT Industries

7 Lightweight Naming Service

7.1 Platform Independent Model

7.1.1 Overview

This section defines the Platform Independent Model (PIM) for the Lightweight Naming Service. The Lightweight Naming Service is intended to be a subset of the Naming Service Specification. The packages, interfaces, and classes appearing in this chapter are intended to model this subset and should map to the IDL for their counterparts in the Naming Service Specification (Version 1.2, September 2002, formal/02-09-02). The descriptions of the interfaces, operations and their semantics are also intended to be identical to those defined by the Naming Service Specification (Version 1.2, September 2002, formal/02-09-02) over this same subset.

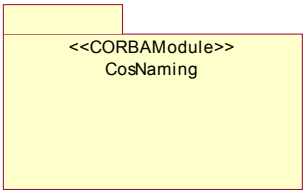


Figure 1 - Lightweight Naming Service Packages

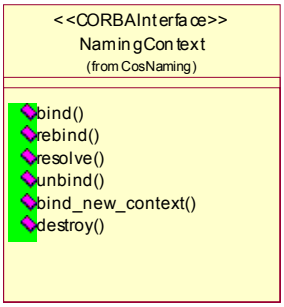


Figure 2 - Lightweight Naming Service Interfaces and Classes

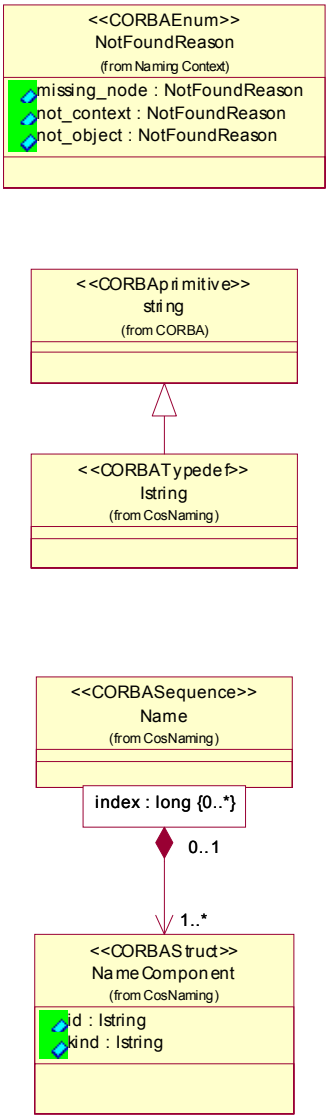


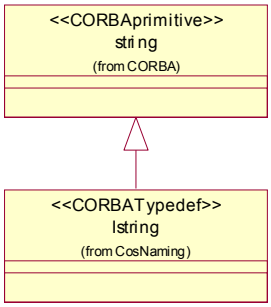
Figure 3 - Lightweight Naming Service Data Types

7.1.2 The CosLightweightNaming Package

The CosLightweightNaming package is a collection of interfaces, datatypes, and exceptions that together define the Lightweight Naming Service. Unlike the full CosNamingService, this package supports only the NamingContext interface.

7.1.2.1 Istring

Description



Istring is a "placeholder for a future IDL internationalized string data type" in the original CosNaming specification. It is maintained solely for compatibility reasons.

Attributes

No additional attributes

Operations

No additional operations

Associations

No associations

Constraints

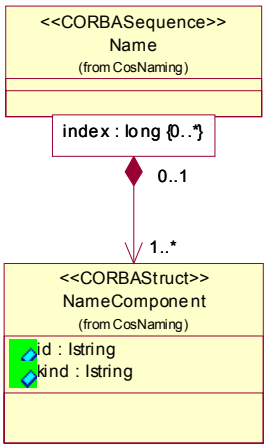
No additional constraints

Semantics

No additional semantics

7.1.2.2 Name

Description



A name is a sequence of NameComponents.

Attributes

No attributes

Operations

No operations

Associations

- `component:` NameComponent[1..*]
  A name consists of an ordered list of NameComponents.

Constraints

No constraints

Semantics

A name is a sequence of NameComponents. The empty sequence is not a legal name. An implementation may limit the length of the sequence to some maximum. When comparing Names for equality, each NameComponent in the first name must match the corresponding NameComponent in the second Name for the names to be considered identical.

7.1.2.3 NameComponent

Description

The NameComponent represents one segment of the name, consisting of two parts represented as attributes.

Attributes

- `id: Istring [1]`

  An arbitrary length string holding the main component of the name.
  (Comment:This is usually the name iteslf.)

- `kind: Istring [1]`

  An arbitrary length string holding the additonal component of the name.
  (Comment: This is usually some characterization of the name.)

Operations

No operations

Associations

No associations

Constraints

No constraints

Semantics

A name component consists of two attributes: the identifier attribute, id, and the kind attribute, kind.

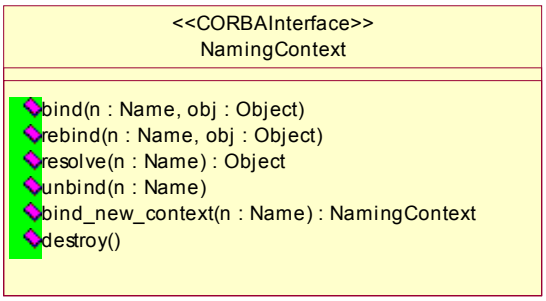
Both of these attributes are arbitrary-length strings of ISO Latin-1 characters, excluding the ASCII NUL character.

When comparing two NameComponents for equality both the id and the kind field must match in order for two NameComponents to be considered identical. This applies for zero-length (empty) fields as well. Name comparisons are case sensitive.

An implementation may place limitations on the characters that may be contained in a name component, as well as the length of a name component. For example, an implementation may disallow certain characters, may not accept the empty string as a legal name component, or may limit name components to some maximum length.

7.1.2.4 NamingContext

Description



A NamingContext is a container hosting a set of name bindings.

Attributes

No attributes.

Operations

- `bind`(in `n`: Name, `in obj`: Object)

  Creates an object binding in the naming context. If a binding with the specified name already exists, bind will raise an AlreadyBound exception. If an implementation places limits on the number of bindings within a context, bind will raise the IMP_LIMIT system exception if the new binding cannot be created. The operation may also raise NotFound, CannotProceed, or InvalidName.

- `rebind`(in n: `Name`, in obj: Object)

  Creates an object binding in the naming context even if the name is already bound in the context. If already bound, the previous binding must be of type object; otherwise, a NotFound exception with a why reason of not_object is raised. If rebind raises a NotFound exception because an already existing binding is of the wrong type, the rest_of_name member of the exception has a sequence length of 1. The operation may also raise CannotProceed or InvalidName.

- `resolve` (in n: Name): Object)

  The resolve operation retrieves an object bound to a name in a given context. The given name must exactly match the bound name. The naming service does not return the type of the object. Clients are responsible for "narrowing" the object to the appropriate type. That is, clients typically cast the returned object from Object to a more specialized interface.

  Names can have multiple components; therefore, name resolution can traverse multiple contexts. These contexts can be federated between different Naming Service instances.
  The operation may raise NotFound, CannotProceed, or InvalidName.

- `unbind`(`in n: Name`)

  The unbind operation removes a name binding from a context. The operation may raise NotFound, CannotProceed, or InvalidName.

- `bind_new_context` (in n: Name): NamingContext

  This operation creates a new context and creates an context binding for it using the name supplied as an argument. If an implementation places limits on the number of naming contexts, bind_new_context can raise the IMP_LIMIT system exception if the context cannot be created. bind_new_context can also raise IMP_LIMIT if the bind would cause an implementation limit on the number of bindings in a context to be exceeded.
  The operation may also raise NotFound, CannotProceed, or InvalidName.

- `destroy`()

  This operation destroys its naming context. If there are bindings denoting the destroyed context, these bindings are not removed. If the naming context contains bindings, the operation raises NotEmpty.

Associations

No association.

Constraints

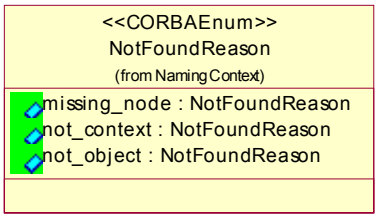
No constraints.

Semantics

A name-to-object association is called a name binding. A name binding is always defined relative to a naming context. A naming context is an object that contains a set of name bindings in which each name is unique. Different names can be bound to an object in the same or different contexts at the same time. There is no requirement, however, that all objects must be named. To resolve a name is to determine the object associated with the name in a given context. To bind a name is to create a name binding in a given context. A name is always resolved relative to a context - there are no absolute names. Because a context is like any other object, it can also be bound to a name in a naming context. Binding contexts in other contexts creates a naming graph - a directed graph with nodes and labeled edges where the nodes are contexts. A naming graph allows more complex names to reference an object. Given a context in a naming graph, a sequence of names can reference an object. This sequence of names (called a compound name) defines a path in the naming graph to navigate the resolution process.

7.1.2.5 NamingContext::NotFoundReason

Description



The enumeration NotFoundReason specifies the reason that a NotFound exception was raised with respect to resolution of a given name (which may be a component of a larger name).

Attributes

- `missing_node`
  The first component of the given name is not bound within its parent context.
- `not_context`
  The first name component of the given name denotes a binding with a type of nobject when the type ncontext was required.
- `not_object`
  The first name component of the given name denotes a binding with a type of ncontext when the type nobject was required.

Operations

No operations

Associations

No associations

Constraints

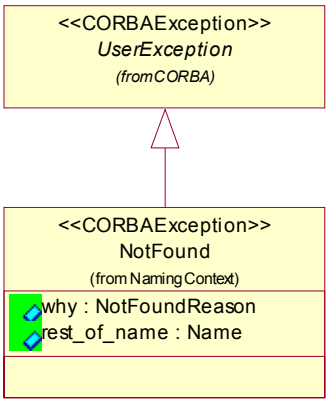
No constraints

Semantics

This is an Enumeration type.

7.1.2.6 NamingContext::NotFound

Description



The NotFound user exception.

Attributes

- `why: NotFoundReason` [1]
  The why attribute explains the reason for the exception.
- `rest_of_name: Name` [1]
  The rest_of_name attribute contains the remainder of the non-working name:

Operations

No operations

Associations

No associations

Constraints

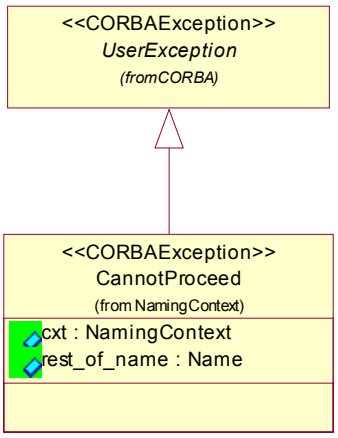
No constraints

Semantics

This exception is raised by operations when a component of a name does not identify a binding, or the type of the binding is incorrect for the operation being performed.

7.1.2.7 NamingContext::CannotProceed

Description



The CannotProceed user exception.

Attributes

- `cxt: NamingContext` [1]
  The cxt attribute contains the context that the operation may be able to retry from.
- `rest_of_name: Name` [1]
  The rest_of_name attribute contains the remainder of the non-working name:

Operations

No operations

Associations

No associations.

Constraints

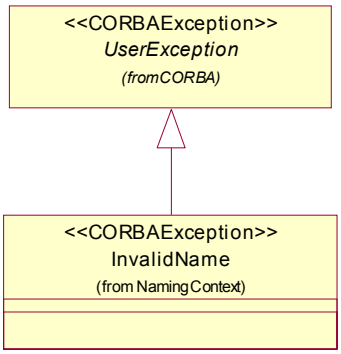
No constraints.

Semantics

This exception is raised when an implementation has given up for some reason. The client, however, may be able to continue the operation at the returned naming context.

7.1.2.8 NamingContext::InvalidName

Description



The InvalidName user exception.

Attributes

No attributes.

Operations

No operation.

Constraints

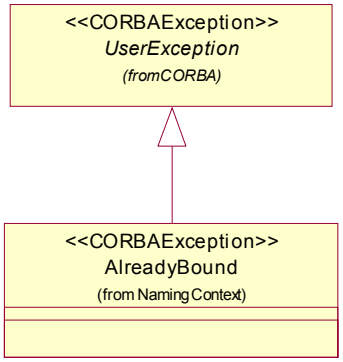
No constraints.

Semantics

This exception is raised if a Name is invalid. A name of length zero is invalid (containing no name components). Implementations may place further limitations on what constitutes a legal name and raise this exception to indicate a violation.

7.1.2.9 NamingContext::AlreadyBound

Description



The AlreadyBound user exception.

Attributes

No attributes.

Operations

No operation.

Constraints

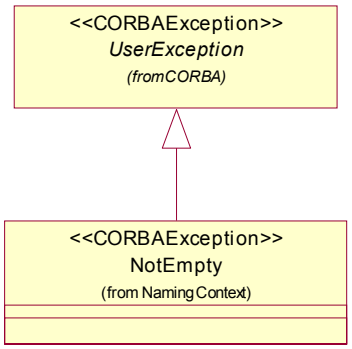
No constraints.

Semantics

Indicates an object is already bound to the specified name. Only one object can be bound to a particular Name in a context. The lightweight naming service user must use the “rebind” interface to explicitly bind a newobject reference to an existing name.

7.1.2.10 NamingContext::NotEmpty

Description



The NotEmpty user exception.

Attributes

No attributes.

Operations

No operation.

Constraints

No constraints.

Semantics

This exception is raised by destroy if the NamingContext contains bindings. A NamingContext must be empty to be destroyed.

7.2 Platform Specific Model: CORBA Service

7.2.1 Overview

The following sections specify a platform specific mapping of the Lightweight Naming Service onto the CORBA platform. The resulting CORBA service is specified in CORBA IDL and represents a fully compatible subset of the CosNamingService.

7.2.2 CosNaming Module

```
#ifndef _COSNAMING_IDL_
#define _COSNAMING_IDL_
```

```
#ifdef _PRE_3_0_COMPILER_
# pragma prefix "omg.org"
#endif

module CosNaming
{
# ifndef _PRE_3_0_COMPILER_
   typeprefix "omg.org";
# endif // _PRE_3_0_COMPILER_
```

7.2.2.1 Istring

```
   typedef string Istring;
```

7.2.2.2 NameComponent

```
   struct NameComponent
   {
      Istring id;
      Istring kind;
   };
   typedef sequence<NameComponent> Name;
```

7.2.2.3 NamingContext

```
   interface NamingContext
   {

      enum NotFoundReason { missing_node, not_context, not_object };

      exception NotFound
         {
         NotFoundReason why;
         Name rest_of_name;
      };

      exception CannotProceed
      {
         NamingContext cxt;
         Name rest_of_name;
      };

      exception InvalidName  {};
      exception AlreadyBound {};
      exception NotEmpty     {};

      void bind(in Name n, in Object obj)
         raises(NotFound, CannotProceed, InvalidName, AlreadyBound);
      void rebind(in Name n, in Object obj)
         raises(NotFound, CannotProceed, InvalidName);
```

```
      Object resolve (in Name n)
         raises(NotFound, CannotProceed, InvalidName);
      void unbind(in Name n)
         raises(NotFound, CannotProceed, InvalidName);
      NamingContext bind_new_context(in Name n)
         raises(NotFound, AlreadyBound, CannotProceed, InvalidName);
      void destroy()
         raises(NotEmpty);
   };

};
#endif // _COSNAMING_IDL_
```

8 Lightweight Event Service

8.1 Platform Independent Model

8.1.1 Overview

This section defines the Platform Independent Model (PIM) for the Lightweight Event Service. The Lightweight Event Service is intended to be a subset of the full CORBA Event Service. The packages, interfaces, and classes appearing in this chapter are intended to model this subset and should map to the IDL for their counterparts in the Event Service Specification (Version 1.1, March 2001). The descriptions of the interfaces, operations and their semantics are also intended to be identical to those defined by the Event Service Specification (Version 1.1, March 2001) over this same subset

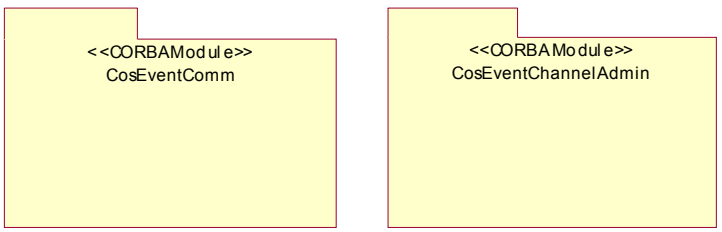


Figure 4 - Lightweight Event Service Packages

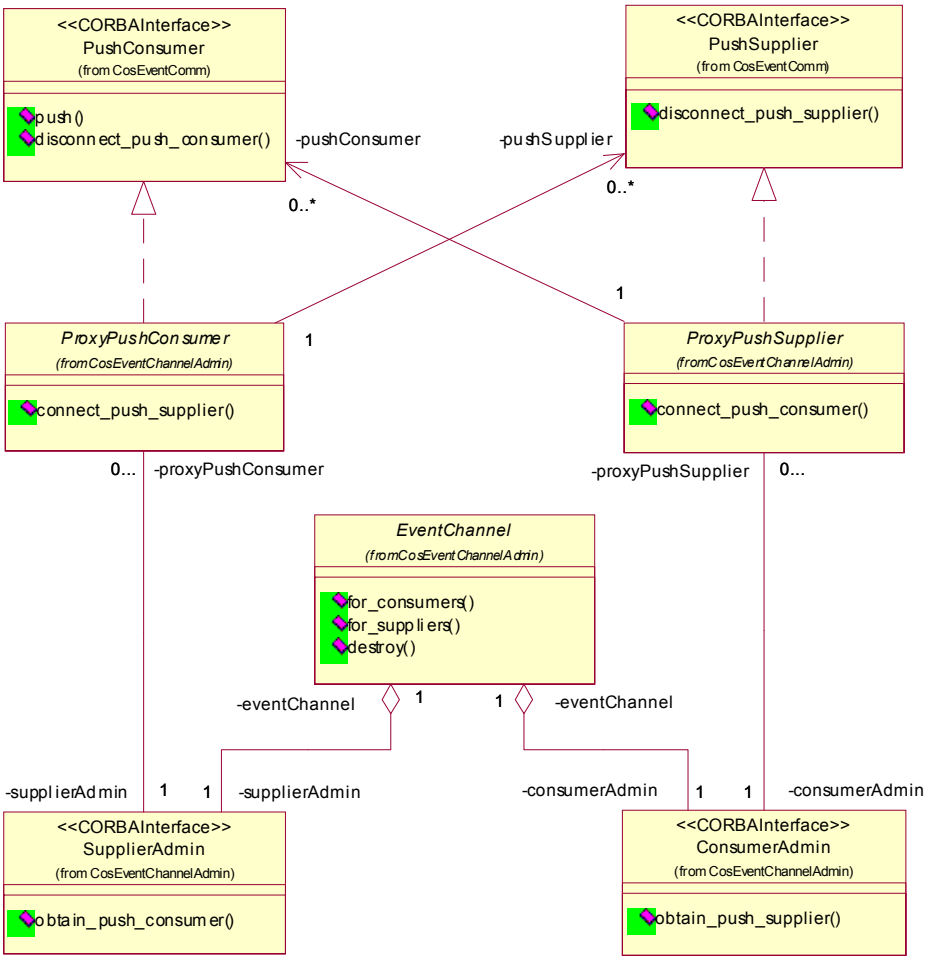


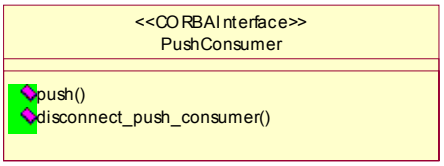
Figure 5 - Lightweight Event Service Interfaces and Classes

8.1.2 The CosLightweightEventComm Package

The CosLightweightEventComm package defines the interfaces for push consumers and push suppliers. Only the push model is supported by the Lightweight Event Service.

8.1.2.1 Push Consumer

Description



A push-style consumer supports the PushConsumer interface to receive event data.

Attributes

No attributes.

Operations

- `push`(in data:Any)

  A supplier communicates event data to the consumer by invoking the push operation and passing the event data as an in parameter. The operation raises the exception Disconnected if the event communication has already been terminated.

- `disconnect_push_consumer` ()

  The disconnect_push_consumer operation terminates the event communication; it releases resources used at the consumer to support the event communication. The PushConsumer object reference is disposed.

Associations

No associations.

Constraints

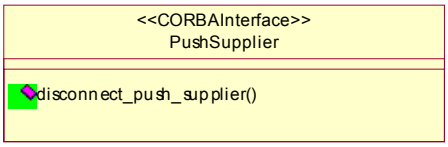
No Constraints.

Semantics

Calling disconnect_push_consumer causes the implementation to call the disconnect_push_supplier operation on the corresponding PushSupplier interface (if that interface is known).

8.1.2.2 Push Supplier

Description



A push-style supplier supports the PushSupplier interface.

Attributes

No attributes.

Operations

- `disconnect_push_supplier` ()

  The disconnect_push_supplier operation terminates the event communication; it releases resources used at the supplier to support the event communication. The PushSupplier object reference is disposed.

Associations

No associations

Constraints

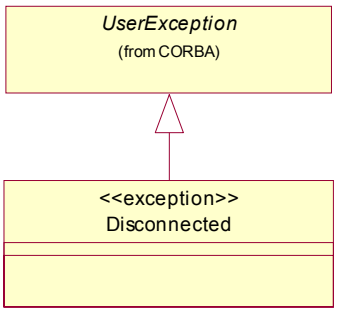
No Constraints.

Semantics

Calling disconnect_push_supplier causes the implementation to call the disconnect_push_consumer operation on the corresponding PushConsumer interface (if that interface is known).

8.1.2.3 Disconnected Exception

Description



Disconnected is the exception raised when an attempt is made to transfer an event after event communication has been terminated. It is a kind of CORBA UserException.

Attributes

No attributes.

Operations

No additional operations.

Associations

No assocication.

Constraints

No constraints.

Semantics

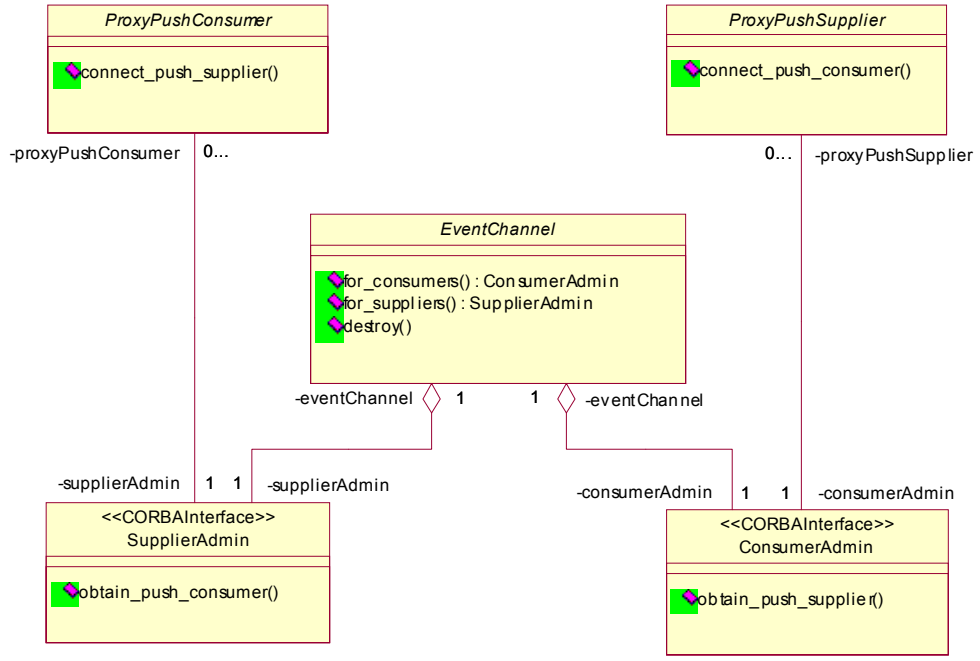
Raised in response to an attempt to push an event after event communication has been terminated. Event communication may be terminated by the operation disconnect_push_consumer.

8.1.3 The CosLightweightEventChannel Package

The CosLightweightEventChannelAdmin package defines the interfaces for making connections between supplier and consumers. Only the push model is supported by the Lightweight Event Service.

8.1.3.1 EventChannel

Description



The EventChannel interface defines three administrative operations: adding consumers, adding suppliers, and destroying the channel.

Any object that possesses an object reference that supports the EventChannel interface can perform the operations listed below.

Consumer administration and supplier administration are defined as separate objects so that the creator of the channel can control the addition of suppliers and consumers. For example, a creator might wish to be the sole supplier of event data but allow many consumers to be connected to the channel. In such a case, the creator would simply export the ConsumerAdmin object.

Attributes

No attributes.

Operations

- `for_consumers`(): ConsumerAdmin

  The ConsumerAdmin interface allows consumers to be connected to the event channel. The for_consumers operation returns an object reference that supports the ConsumerAdmin interface.

- `for_suppliers`(): SupplierAdmin

The SupplierAdmin interface allows suppliers to be connected to the event channel. The for_suppliers operation returns an object reference that supports the SupplierAdmin interface.

- `destroy()`
  The destroy operation destroys the event channel.

Associations

- `supplierAdmnin: SupplierAdmin` [1]
  Each event channel has a single associated SupplierAdmin object.
- `consumerAdmin: ConsumerAdmin` [1]
  Each event channel has a single associated ConsumerAdmin object.

Constraints

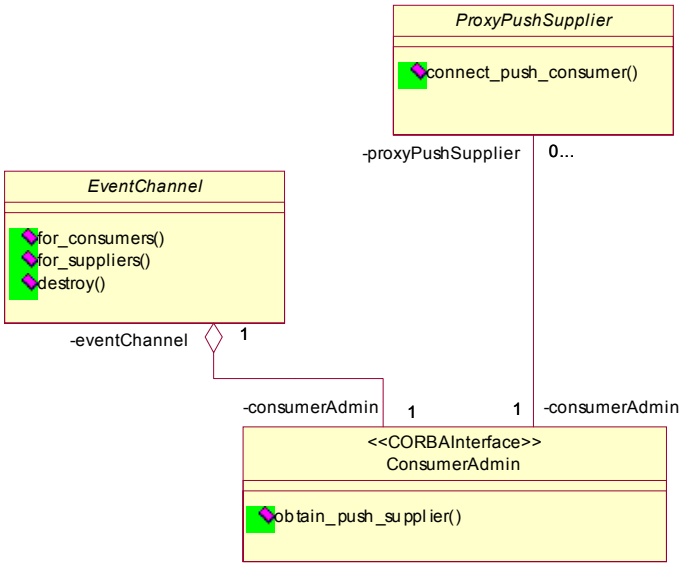
No constraints.

Semantics

Destroying an event channel destroys all ConsumerAdmin and SupplierAdmin objects that were created via that channel. Destruction of a ConsumerAdmin or SupplierAdmin object causes the implementation to invoke the disconnect operation on all proxies that were created via that ConsumerAdmin or SupplierAdmin object.

8.1.3.2 ConsumerAdmin

Description



The ConsumerAdmin interface defines the first step for connecting consumers to the event channel; clients use it to obtain proxy suppliers.

Attributes

No attributes.

Operations

- `obtain_push_supplier(): ProxyPushSupplier`

  The obtain_push_supplier operation returns a ProxyPushSupplier object. The ProxyPushSupplier object is then used to connect a push-style consumer.

Associations

- eventChannel: EventChannel [1]
  The EventChannel object with which the ConsumerAdmin object is associated.
- `proxyPushSupplier`: ProxyPushSupplier [0..*]
  A proxy push supplier returned by the obtain_push_supplier operation.

Constraints

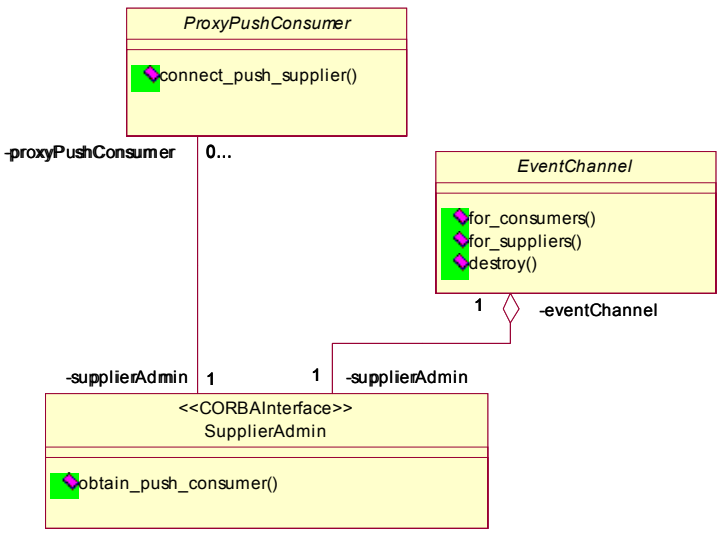
No constraints.

Semantics

The ConsumerAdmin interface for the Lightweight Event Service defines only the full Event Service operations need to support the push model of event communication. It provides a logical link between the EventChannel object with which it is associated and the ProxyPushSupplier object to which consumers connect in order to receive events.

8.1.3.3 SupplierAdmin

Description



The SupplierAdmin interface defines the first step for connecting suppliers to the event channel; clients use it to obtain proxy consumers.

Attributes

No attributes.

Operations

- `obtain_push_consumer(): ProxyPushConsumer`

  The obtain_push_consumer operation returns a ProxyPushConsumer object. The ProxyPushConsumer object is then used to connect a push-style supplier.

Associations

- `eventChannel: EventChannel` [1]
  The EventChannel object with which the SupplierAdmin object is associated.
- `proxyPushConsumer: ProxyPushConsumer` [0..*]
  A proxy push consumer returned by the obtain_push_consumer operation.

Constraints

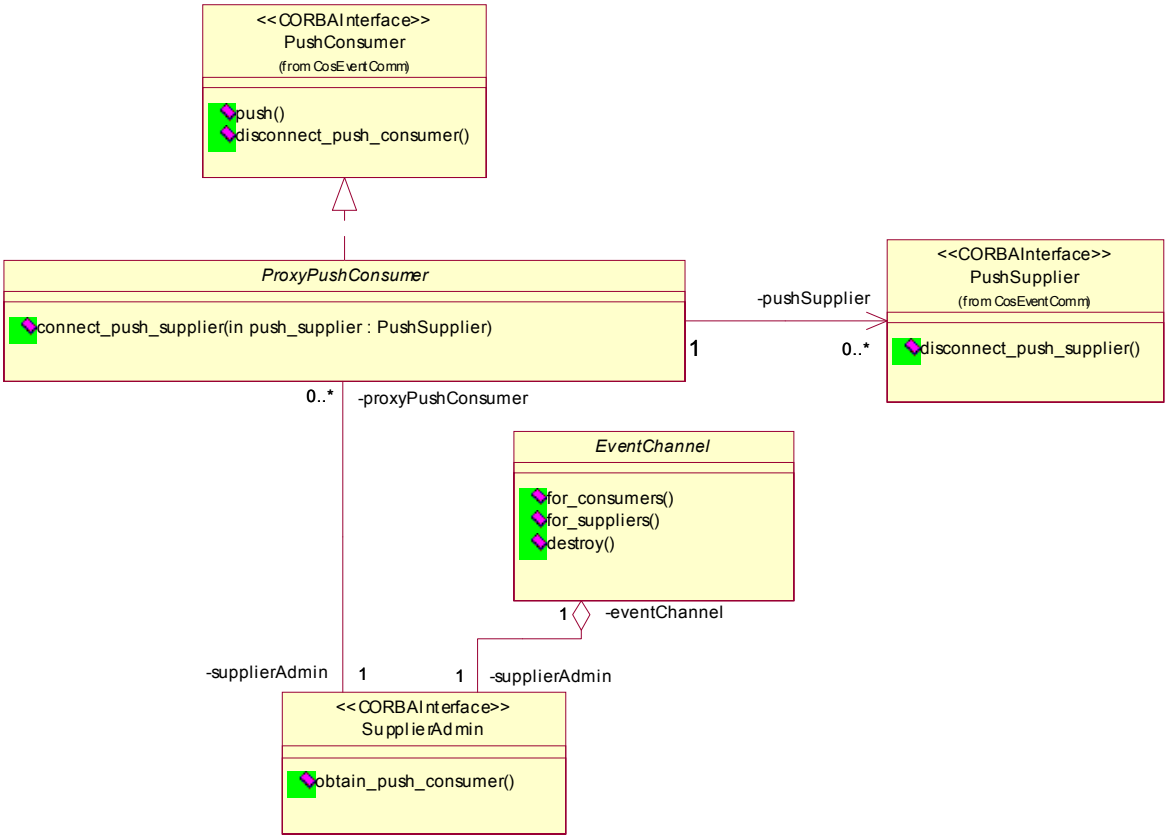
No constraints.

Semantics

The SupplierAdmin interface for the Lightweight Event Service defines only the full Event Service operations needed to support the push model of event communication. It provides a logical link between the EventChannel object with which it is associated and the ProxyPushConsumer object to which suppliers push events.

8.1.3.4 ProxyPushConsumer

Description



The ProxyPushConsumer class defines the second step for connecting push suppliers to the event channel. It realizes the interface defined by PushConsumer and extends it to support the connection of push suppliers.

Attributes

No attributes.

Operations

- `connect_push_supplier(in pushSupplier: PushSupplier)`

  A nil object reference may be passed to the connect_push_supplier operation; if so a channel cannot invoke the disconnect_push_supplier operation on the supplier; the supplier may be disconnected from the channel without being informed.

  If a non-nil reference is passed to connect_push_supplier, the implementation calls disconnect_push_ supplier via that reference when the ProxyPushConsumer is destroyed.

  If the ProxyPushConsumer is already connected to the given PushSupplier, then the AlreadyConnected exception is

raised.

Associations

- `supplierAdmin: SupplierAdmin` [1]
  The SupplierAdmin object with which the ProxyPushConsumer object is associated.
- `pushSupplier: PushSupplier` [0..*]
  The PushSupplier objects (if any) connected to the ProxyPushConsumer object.

Constraints

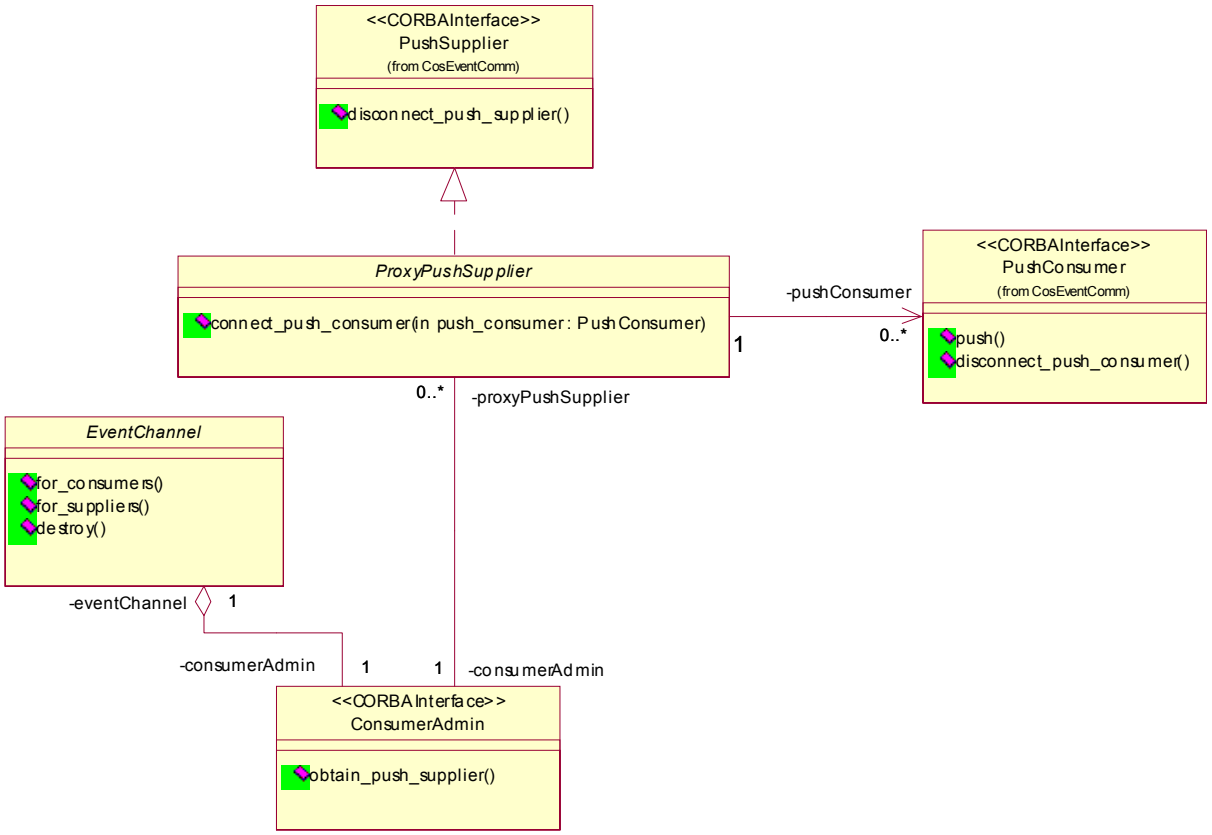
No constraints.

Semantics

The ProxyPushConsumer object acts as a surrogate (proxy) to which suppliers push events.

8.1.3.5 ProxyPushSupplier

Description



The ProxyPushSupplier class defines the second step for connecting push consumers to the event channel. It realizes the interface defined by PushSupplier and extends it to support the connection of push consumers.

Attributes

No attributes.

Operations

- `connect_push_consumer(in pushConsumer: PushConsumer)`

  Implementations shall raise the CORBA standard BAD_PARAM exception if a nil object reference is passed to the connect_push_consumer operation.

  If the ProxyPushSupplier is already connected to the given PushConsumer, then the AlreadyConnected exception is raised.

Associations

- `consumerAdmin: ConsumerAdmin` [1]
  The ConsumerAdmin object with which the ProxyPushSupplier object is associated.
- `pushConsumer: PushConsumer` [0..*]
  The PushConsumer objects (if any) connected to the ProxyPushSupplier object.

Constraints

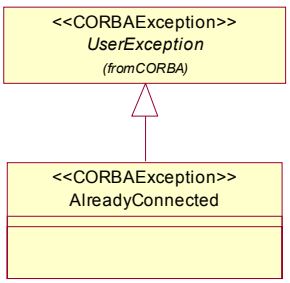
No constraints.

Semantics

The implementation calls disconnect_push_consumer on the reference passed to connect_push_ consumer when the the ProxyPushSupplier is destroyed.

8.1.3.6 AlreadyConnected Exception

Description



AlreadyConnected is the exception raised when an attempt is made to connect a consumer/producer to a proxy that is already has a connection to the same object. It is a kind of CORBA UserException.

Attributes

No attributes.

Operations

No additional operations.

Associations

No associations.

Constraints

No constraints.

Semantics

Raised if an attempt is made to connect a PushConsumer object to a ProxyPushSupplier object when the two are already connected, or when an attempt is made to connect a PushSupplier object to a ProxyPush Consumer object when the two are already connected.

8.2 Platform Specific Model: CORBA Service

8.2.1 Overview

The following sections specify a platform specific mapping of the Lightweight Event Service onto the CORBA platform. The resulting CORBA service is specified in CORBA IDL and represents a fully compatible subset of the CosEventService.

8.2.2 CosEventChannelAdmin Module

```
#include <CosEventComm.idl>
#pragma prefix "omg.org"
module CosEventChannelAdmin {
# ifndef _PRE_3_0_COMPILER_
   typeprefix "omg.org";
# endif // _PRE_3_0_COMPILER_

    exception AlreadyConnected {};
    exception TypeError {};
```

8.2.2.1 ProxyPushConsumer

```
    interface ProxyPushConsumer: CosEventComm::PushConsumer {
        void connect_push_supplier(
                in CosEventComm::PushSupplier push_supplier)
            raises(AlreadyConnected);
    };
```

8.2.2.2 ProxyPushSupplier

```
interface ProxyPushSupplier: CosEventComm::PushSupplier {
    void connect_push_consumer(
            in CosEventComm::PushConsumer push_consumer)
        raises(AlreadyConnected, TypeError);
};
```

8.2.2.3 ConsumerAdmin

```
interface ConsumerAdmin {
    ProxyPushSupplier obtain_push_supplier();
};
```

8.2.2.4 SupplierAdmin

```
interface SupplierAdmin {
    ProxyPushConsumer obtain_push_consumer();
};
```

8.2.2.5 EventChannel

```
interface EventChannel {
    ConsumerAdmin for_consumers();
    SupplierAdmin for_suppliers();
    void destroy();
};

};

#endif /* ifndef _COS_EVENT_CHANNEL_ADMIN_IDL_ */
```

8.2.3 CosEventComm Module

```
//File: CosEventComm.idl
//Part of the Event Service

#ifndef _COS_EVENT_COMM_IDL_
#define _COS_EVENT_COMM_IDL_
#pragma prefix "omg.org"
module CosEventComm
{
    # ifndef _PRE_3_0_COMPILER_
        typeprefix "omg.org";
    # endif // _PRE_3_0_COMPILER_

    exception Disconnected{};
```

8.2.3.1 PushConsumer

```
interface PushConsumer
{
    void push (in any data) raises(Disconnected);
    void disconnect_push_consumer();
};
```

8.2.3.2 PushSupplier

```
  interface PushSupplier
    {
        void disconnect_push_supplier();
    };

};
#endif /* ifndef _COS_EVENT_COMM_IDL_ */
```

9 Lightweight Time Service

9.1 Platform Independent Model

9.1.1 Overview

This section defines the Platform Independent Model (PIM) for the Lightweight Time Service. The Lightweight Time Service is intended to be a subset of the full CORBA Enhanced View of Time Service. The packages, interfaces, and classes appearing in this chapter are intended to model this subset and should map to the IDL for their counterparts in the CORBA Enhanced View of Time Service Specification (Version 1.1, May 2002). The descriptions of the interfaces, operations and their semantics are also intended to be identical to those defined by the CORBA Enhanced View of Time Service Specification (Version 1.1, May 2002) over this same subset.

9.1.2 Minor Conformance Points

The platform independent model of the Lightweight Time Service supports two *optional* minor conformance points: *Support of Multiple Clocks* and *Support of Periodic Execution Control*.

- `Support of Multiple Clocks`

  This conformance point controls the presence or absense of an *optional* model section. If the conformance point evaluates to true, the ClockCatalog interface and the ClockEntry structure.are included in the model, providing support for multiple clocks.

- `Support of Periodic Execution Control`

  This conformance point controls the presence or absense of an optional model section. If the conformance point evaluates to true, the PeriodicExecution package is included in themodel, thus providing support for clock-controlled periodic execution.

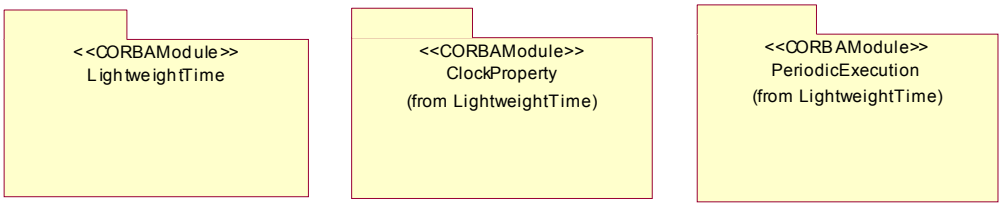


Figure 6 - Lightweight Time Service Package Structure

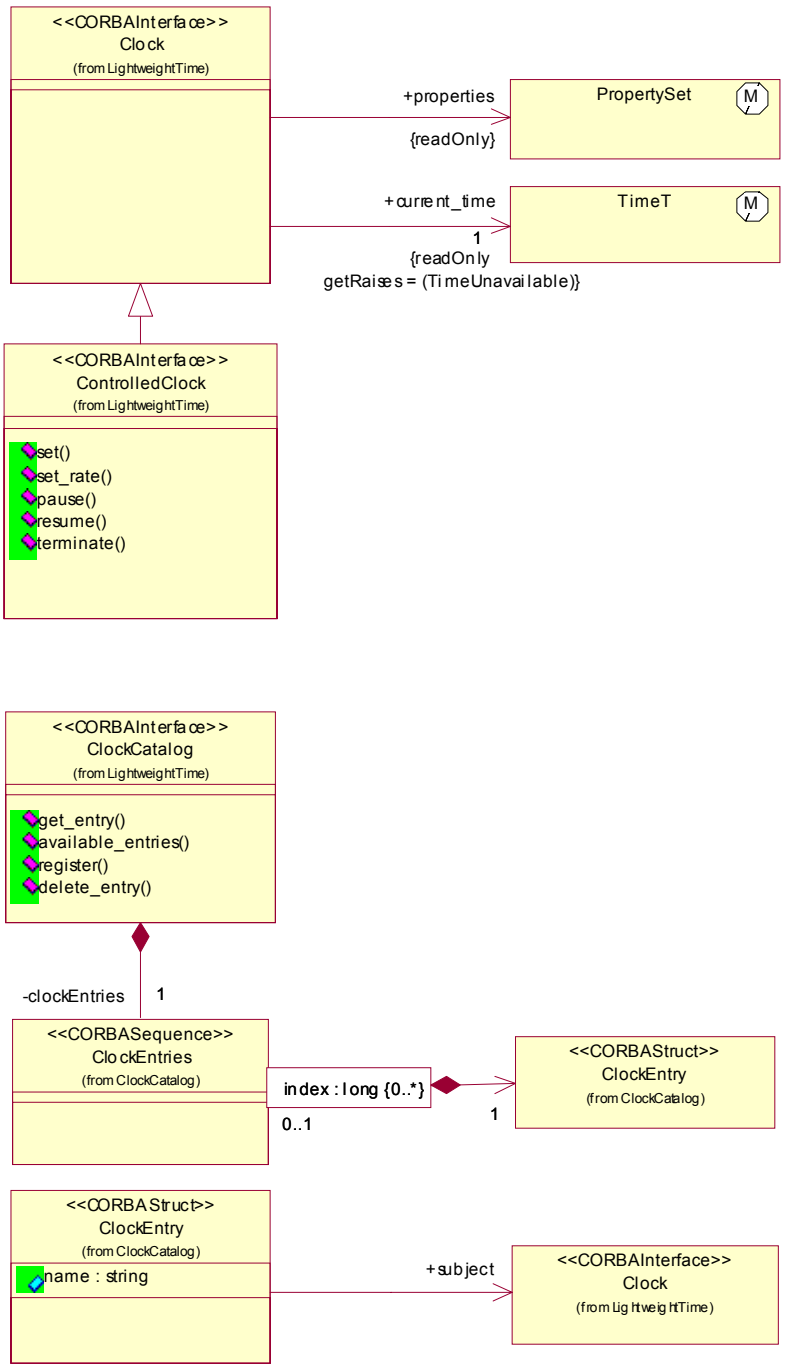
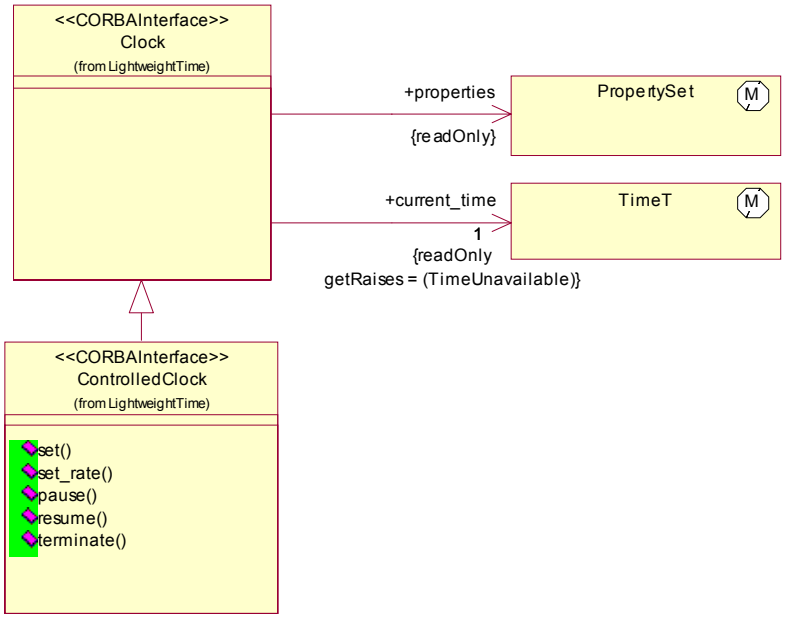


Figure 7 - Lightweight Time Service Interfaces and Classes

9.1.3 The LightweightTime Package

The LightweightTime package defines interfaces for finding a clock reading, a time source, controlling a clock and support for periodic execution. Synchronization of clocks is not supported in the LightweightTime package.



9.1.3.1 Clock

Description

Base interface for all clocks.

Attributes

No attributes.

Operations

No operations.

Associations

- `properties: PropertySet` [1]
  Points to a PropertySet holding the specific properties of the clock.
- `current_time: TimeT` [1]
  Points to a data element holding the current time as a 64-bit value with a resolution of 100 nanoseconds.

Constraints

No constraints.

Semantics

This is the base interface for all clocks defined in the Lightweight Time Service. It provides configurability for the clock via properties (name-value pairs) and access to a time base.

9.1.3.2 ControlledClock

Description

A user-controllable specialization of the Clock interface.

Attributes

No attributes.

Operations

- `set`(in t0: TimeT)
  This operation sets the controllable clock to the specified specific time.
- `set_rate`(in ratio: Float)
  This operation allows a clock to be speeded up or slowed down (or run backwards). The parameter indicates the ratio of the elapse of the clock's readout to the real passage of time.
- `pause`()
  This opperation pauses the apparent elapse of time.
- `resume`()
  This operation resumes the apparent elapse of time.
- `terminate`()
  This operation stops the controlled clock permanently.

Associations

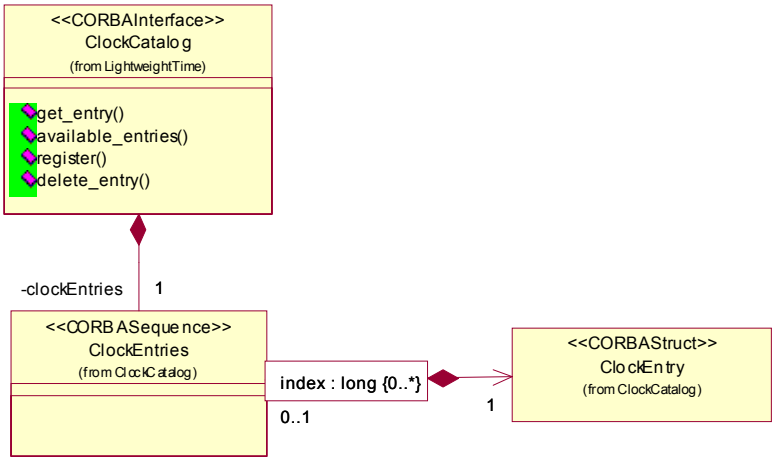
No additonal associations.

Constraints

No Constraints.

Semantics

The ControlledClock is a specialization of the Clock interface. It provides the ability to set the clock to certain value, control the apparent "speed" (time elapse rate), and to pause and resume the clock under user control.



9.1.3.3 ClockCatalog

This interface is part of the optional minor conformance point "Support of Multiple Clocks"

Description

A lightweight catalog of available clocks.

Attributes

No attributes.

Operations

- `get_entry`(in name: String): ClockEntry
  Returns a single clock entry holding the information about a particular clock. The clock entry is selected via the clock entry name.
- `available_entries`(): ClockEntries
  Returns the whole catalog to allow the client the application of a more specific selection mechaism, as for example by a specific property.
- `register`(in entry: ClockEntry)
  Register a new clock entry in the catalog.
- `delete_entry`()
  Permanently removes a clock entry from the clock catalog.

Associations

- `clockEntries: ClockEntries`[1]
  The encapsulation of the clock entry catalog content.

Constraints

No constraints.

Semantics

The ClockCatalog is the user-visible interface to a single-level lightweight trader service equivalent, holding information about available clock definitions.

9.1.3.4 ClockEntries

This set is part of the optional minor conformance point "Support of Multiple Clocks."

Description

The set holding the individual clock entries.

Attributes

No attributes.

Operations

No operations.

Associations

- `clockEntry: ClockEntry`[*]
  The actual set holding the individual entries in the clock catalog.

Constraints

No constraints.

Semantics

Provides an encapsulation for the set of individual clock information entries.

9.1.3.5 ClockEntry

This interface is part of the optional minor conformance point "Support of Multiple Clocks."

Description

An individual entry in the clock catalog.

Attributes

- name: String [1]
  The ClockEntry name.

Operations

No operations.

Associations

- clockl: Clock [1]
  The clock definition represented by this catalog entry.

Constraints

No constraints.

Semantics

A ClockEntry consists of a name (unique within the catalog) and a reference to a particular clock definition.

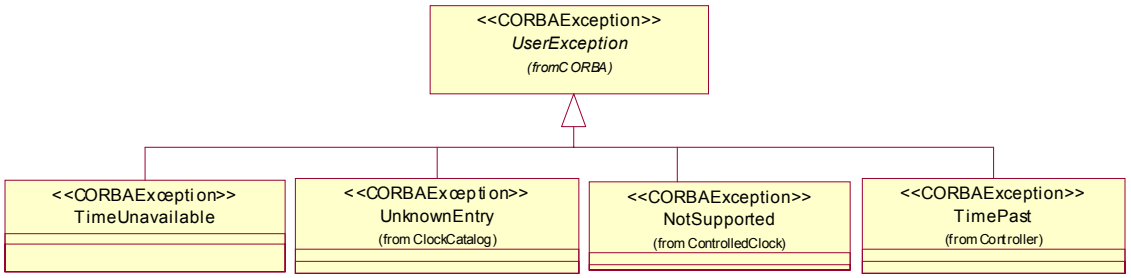


Figure 8 - Lightweight Time Service Exceptions

9.1.3.6 TimeUnavailable

Description

TimeUnavailable exception.

Attributes

No attributes.

Operations

No operations.

Associations

No associations.

Constraints

No constraints.

Semantics

This exception is raised whenever the underlying clock fails, or is unable to providetime that meets the required security assurance.

9.1.3.7 UnknownEntry

Description

UnknownEntry exception.

Attributes

No attributes.

Operations

No operations.

Associations

No associations.

Constraints

No constraints.

Semantics

Indicates that the catalog contains no entry with the given name.

9.1.3.8 NotSupported

Description

NotSupported exception.

Attributes

No attributes.

Operations

No operations.

Associations

No associations.

Constraints

No constraints.

Semantics

The NotSupported exception may be raised if the operation is not supported for the instance of the ControlledClock, or if its characteristics disallow the operation. For example, the rate of a ControlledClock may not be settable. Other clocks may not be allowed to run “backwards.”

9.1.3.9 TimePast

Description

TimePast exception.

Attributes

No attributes.

Operations

No operations.

Associations

No associations.

Constraints

No constraints.

Semantics

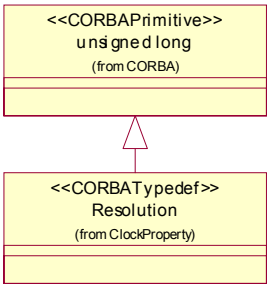
Raised by the start_at or resume_at operations if the requested time is in the past.

9.1.4 The ClockProperty Package

This package contains only data definitions. They constitute the minimum set of properties required for any clock.

9.1.4.1 Resolution

Description



Defines the apparent clock resolution.

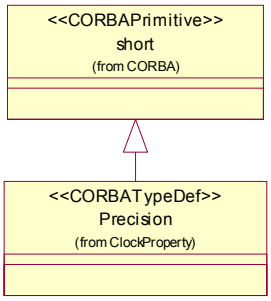
Constraints

Must be specified in units of nanoseconds.

Semantics

No special semantics.

9.1.4.2 Precision



Description

Defines the apparent clock precision.

Constraints

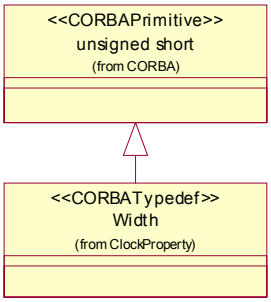
No constraints.

Semantics

Raised by the start_at or resume_at operations if the requested time is in the past.

9.1.4.3 Width

Description



Number of bits in clock readout.

Constraints

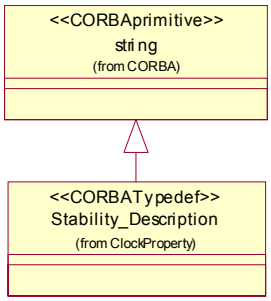
No constraints.

Semantics

Commonly used readout widths are less or equal 64 bits.

9.1.4.4 Stability_Description

Description



Describes the clock stability.

Constraints

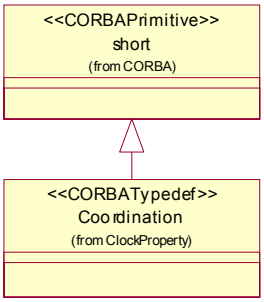
No constraints.

Semantics

No special semantics.

9.1.4.5 Coordination

Description



Defines the clock coordination method

Constraints

Under the Lightweight Time Service, Coordination is restricted to the following set of values:

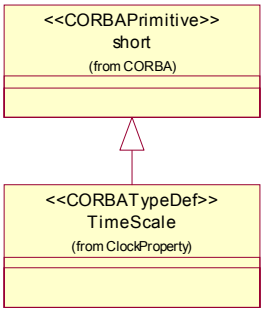
Name	Value	Meaning
`Uncoordinated`	`0`	only static characterization is available

Semantics

No special semantics.

9.1.4.6 TimeScale

Description



Defines the time scale used by the clock.

Constraints

Under the Lightweight Time Service, TimeScale is restricted to the following set of values:

Name	Value	Meaning
`Unknown`	`-1`	
`TAI`	`0`	International Atomic Time
`UT0`	`1`	diurnal day
`UT1`	`2`	+ polar wander
`UTC`	`3`	TAI + leap second
`TT`	`4`	terrestrial time
`TDB`	`5`	Barycentric Dynamical Time
`TCG`	`6`	Geocentric Coordinated Time
`TCB`	`7`	Barycentric Coordinated Time
`Sidereal`	`8`	hour angle of veneral equinox

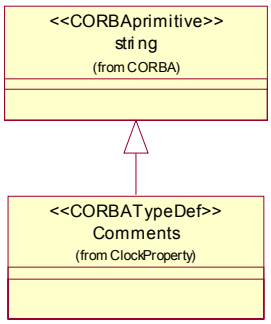
Name	Value	Meaning
`Local`	`9`	UTC + time zone
`GPS`	`10`	Global Positioning System
`Other`	`0x7fff`	e.g., mission

Semantics

No special semantics.

9.1.4.7 Comments

Description



For supplemental comments.

Constraints

No constraints.

Semantics

No special semantics.

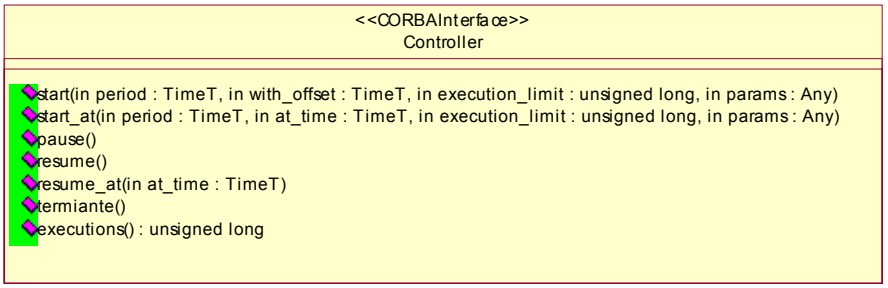
9.1.5 The PeriodicExecution Package

This package is part of the optional minor conformance point "Support of Periodic Execution Control."

9.1.5.1 Controller

This interface is part of the optional minor conformance point "Support of Periodic Execution Control."

Description



Controls periodic execution.

Attributes

No attributes.

Operations

- `start`(in period: TimeT, in with_offset: TimeT, in execution_limit: unsigned long, in params: Any)

  Initiates periodic execution with a specified period for a specified count of executions. Specifying an execution limit of 0 is interpreted as an unbounded number of executions. The with_offset parameter may be used to delay the start of the first execution. The value of the type any parameter params will be passed to each invocation.

- `start_at`(in period: TimeT, in at_time: TimeT, in execution_limit: unsigned long, in params: Any)

  Identical to the start operation except that the at_time parameter specifies an absolute time for the start of the first execution.

- `pause`()
  Pauses periodic execution.

- `resume`()
  Resumes periodic execution immediately.

- `resume_at`(in at_time: TimeT)
  Resumes periodic execution at a particular time.

- `terminate`()
  Terminates periodic execution.

- `executions`(): unsigned long
  Reports the number of periodic executions that have already been initiated.

Associations

No associations.

Constraints

No constraints.

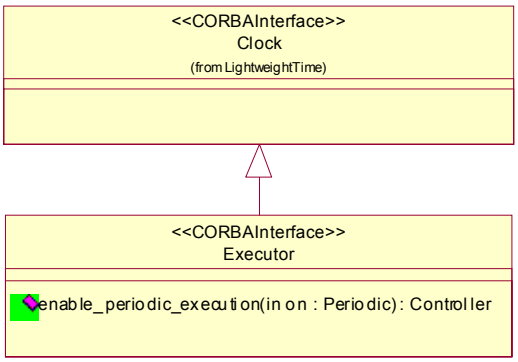
Semantics

This interface provides control over periodic execution. The appropriate object has been registered with the clock and must specialize the Periodic interface.

9.1.5.2 Executor

This interface is part of the optional minor conformance point "Support of Periodic Execution Control."

Description



Register an object for periodic execution.

Attributes

No attributes.

Operations

- `enable_periodic`(in on: Periodic): Controller

  Registers an object that specializes the Periodic interface for periodic execution. The operation returns a reference to the associated Controller interface.

Associations

No associations.

Constraints

No constraints.

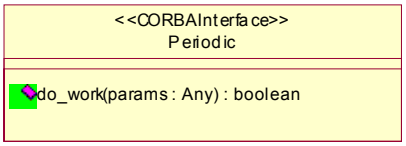
Semantics

The Executor is an interface for a factory that associates the specified object with a clock capable of supporting periodic execution. The registered object must specialize the Periodic interface. The Executor interface returns a reference to the Controller interface associated with this periodic execution.

9.1.5.3 Periodic

This interface is part of the optional minor conformance point "Support of Periodic Execution Control."

Description



Make an object capable for periodic execution.

Attributes

No attributes.

Operations

- `do_work`(in params: Any): boolean

  The do_work operation will be periodically invoked by this service. Each invocation will be passed the type any value registered by the start or start_at operations on the Controller instance. The user implementation of the do_work operation should return a value of TRUE to continue periodic invocation; a value of FALSE will terminate periodic invocation.

Associations

No associations.

Constraints

No constraints.

Semantics

Instances of objects that are to be periodically executed must specialize and implement the Periodic interface. This means they must provide a do_work operation, and a means to enter a "ready to execute" state prior to registration with a clock.

9.2 Platform Specific Model: CORBA Service

9.2.1 Overview

The following sections specify a platform specific mapping of the Lightweight Time Service onto the CORBA platform. The resulting CORBA service is specified in CORBA IDL and represents a fully compatible subset of the Enhanced View of Time service, version 1.1

9.2.2 Minor Conformance Points

The platform specific model of the Lightweight Time Service supports the two minor conformance points of the platform independent model: *Support of Multiple Clocks* and *Support of Periodic Execution Control*. The selection of the corresponding features in the IDL definition is controlled by two preprocessor symbols controlling sets of conditional compilation preprocessor directives.

- `LW_TIME_HAS_SUPPORT_OF_MULTIPLE_CLOCKS`

  If this preprocessor symbol is defined, support for multiple clocks is activated by including the ClockCatalog interface and the ClockEntry structure.

- `LW_TIME_HAS_SUPPORT_OF_PERIODIC_EXECUTION_CONTROL`

  If this preprocessor symbol is defined, the PeriodicExecution module is enabled, which contains support for clock-controlled periodic execution.

9.2.3 LightweightTime Module

```
#include <TimeBase.idl>
#include <CosPropertyService.idl>
#pragma prefix "omg.org"
module LightweightTime
{
# ifndef _PRE_3_0_COMPILER_
   typeprefix "omg.org";
# endif // _PRE_3_0_COMPILER_

    interface Clock;
```

9.2.3.1 ClockProperty Module

```
    module ClockProperty
    {

        // the minimum set of properties to be supported for a clock
        typedef unsigned long Resolution;     // units = nanoseconds
        typedef short Precision;              // ceiling of log_2(seconds
                                              // signified by least significant
                                              // bit of time readout)
        typedef unsigned short Width;         // no. of bits in readout -
                                              // usually <= 64
        typedef string Stability_Description;
        typedef short Coordination;
        const Coordination Uncoordinated = 0; // only static characterization
                                                // is available

        typedef short TimeScale;
        // possible values for TimeScale (“pseudo-enumeration”)
            const TimeScale Unknown      = -1;
```

```
        const TimeScale TAI          = 0;  // International Atomic Time
        const TimeScale UT0          = 1;  // diurnal day
        const TimeScale UT1          = 2;  // + polar wander
        const TimeScale UTC          = 3;  // TAI + leap seconds
        const TimeScale TT           = 4;  // terrestrial time
        const TimeScale TDB          = 5;  // Barycentric Dynamical Time
        const TimeScale TCG          = 6;  // Geocentric Coordinate Time
        const TimeScale TCB          = 7;  // Barycentric Coordinate Time
        const TimeScale Sidereal     = 8;  // hour angle of vernal equinox
        const TimeScale Local        = 9;  // UTC + time zone
        const TimeScale GPS          = 10; // Global Positioning System
        const TimeScale Other        = 0x7fff; // e.g. mission
    // end of pseudo-enumeration

    typedef string Comments;

}; // end of module ClockProperty

exception TimeUnavailable {};
```

9.2.3.2 Clock Interface

```
// the basic clock interface
interface Clock // a source of time readings
{
    readonly attribute CosPropertyService::PropertySet properties;
    readonly attribute TimeBase::TimeT current_time
    getRaises(TimeUnavailable);
};
```

9.2.3.3 ClockCatalog Interface

```
#ifdef LWTIME_HAS_SUPPORT_OF_MULTIPLE_CLOCKS

// alternative to Trader service (e.g., for embedded systems)
//  Optional for system support of multiple clocks.
interface ClockCatalog
{
    struct ClockEntry
    {
        Clock       subject;
        string      name;
    };

    typedef sequence<ClockEntry> ClockEntries;
    exception UnknownEntry {};
    ClockEntry get_entry(in string with_name) raises (UnknownEntry);
    ClockEntries available_entries();
    void register(in ClockEntry entry);
    void delete_entry(in string with_name) raises (UnknownEntry);
};
```

```
#endif  // LWTIME_HAS_SUPPORT_OF_MULTIPLE_CLOCKS
```

9.2.3.4 ControllableClock Interface

```
// a controllable clock
interface ControlledClock: Clock
{
    exception NotSupported {};
    void set(in TimeBase::TimeT to)   raises (NotSupported);
    void set_rate(in float ratio)     raises (NotSupported);
    void pause()                      raises (NotSupported);
    void resume()                     raises (NotSupported);
    void terminate()                  raises (NotSupported);
};
```

9.2.4 PeriodicExecution Module

```
// Optional for Lightweight Time.

#ifdef LWTIME_HAS_SUPPORT_OF_PERIODIC_EXECUTION_CONTROL

module PeriodicExecution
{
```

9.2.4.1 Periodic Interface

```
    // (conceptually abstract) base for objects that can be
    // invoked periodically
    interface Periodic
    {
        boolean do_work(in any params); // return FALSE terminates
                                        // periodic execution
    };
```

9.2.4.2 Controller Interface

```
    // control object for periodic execution
    interface Controller
    {
        exception TimePast {};
        void start(in TimeBase::TimeT period,
                   in TimeBase::TimeT with_offset,
                   in unsigned long   execution_limit, // 0 = no limit
                   in any             params);
        void start_at(in TimeBase::TimeT period,
                   in TimeBase::TimeT at_time,
                   in unsigned long   execution_limit, // 0 = no limit
                   in any             params) raises (TimePast);
        void pause();
        void resume();
        void resume_at(in TimeBase::TimeT at_time) raises(TimePast);
```

```
            void terminate();
            unsigned long executions();
        };
```

9.2.4.3 Executor Interface

```
        // factory clock for periodic execution
        interface Executor : Clock
        {
            Controller enable_periodic_execution(in Periodic on);
        };

    }; // end of module PeriodicExecution

#endif  // LWTIME_HAS_SUPPORT_OF_PERIODIC_EXECUTION_CONTROL

}; //end of module LightweightTime

#endif // _LightweightTime_IDL_
```