WebSphere Application Server Enterprise Edition
Component Broker

Programming Reference

Version 3.0

SC09-4446-00

WebSphere Application Server Enterprise Edition
Component Broker

Programming Reference

Version 3.0

SC09-4446-00

Note!
Before using this information and the product it supports, be sure to read the general information under “Notices” on page 943.

Fifth Edition (August, 1999)

This edition applies to:

WebSphere Application Server Version 3.0, Enterprise Edition Development Toolkit for AIX, program number 5765–E27

WebSphere Application Server Version 3.0, Enterprise Edition Development Toolkit for Windows NT, program number 5639–I07

WebSphere Application Server Version 3.0, Enterprise Edition for AIX, program number 5765–E28

WebSphere Application Server Version 3.0, Enterprise Edition for Windows NT and Gradient DCE, program number 5639–I08

WebSphere Application Server Version 3.0, Enterprise Edition for Windows NT and IBM DCE, program number 5639–I09

WebSphere Application Server Version 3.0, Enterprise Edition Encina Client for Windows NT/Windows 95, program number 5639–I10

WebSphere Application Server Version 3.0, Enterprise Edition MQSeries Application Adapter, program number 5639–I11

WebSphere Application Server Version 3.0, Enterprise Edition Oracle Application Adapter, program number 5639–I12

WebSphere Application Server Version 3.0, Enterprise Edition CICS/IMS Application Adapter, program number 5639–I13

and to all subsequent versions, releases, and modifications until otherwise indicated in new editions. Consult the latest edition of the applicable system bibliography for current information on these products.

Order publications through your IBM representative or through the IBM branch office serving your locality.

Contents

About this book

The *Component Broker Programming Reference* contains information about programming interfaces. This includes:

- Interfaces that make up the Object Request Broker, CORBA module, and the TypeCode library
- Interfaces for the object services
- Object-Oriented Structured Query Language (OOSQL)
- Date, Decimal, Duration, Time and Timestamp helper classes that emulate DB2 data types and can be used to manipulate Oracle DATE and NUMBER datatypes

Who should read this book

The *Component Broker Programming Reference* is intended for application developers who use the Component Broker environment to build distributed object-oriented applications.

The examples are written in C++; therefore, programming experience in C++ and a background in object-oriented programming is required. A familiarity with Java is also helpful, but not required.

This book is a programming manual for experienced programmers who are going to use this product.

Documentation conventions

The following conventions distinguish different text elements:

plain Window titles, folder names, icon names, and method names.

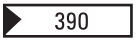
monospace
Programming examples, user input at the command line prompt or into an entry field, user output, and directory paths.

bold Menu choices and menu names, labels for push buttons, check boxes, radio buttons, group-box controls, drop-down list boxes, combo-boxes, notebook tabs, and entry fields.

italics Programming keywords and variables, titles of documents, and initial use of terms that are in the glossary.

The following icons are used to indicate platform-specific sections:

Denotes a section that applies to the AIX platform



Denotes a section that applies to the OS/390 platform

Denotes a section that applies to the Solaris platform

Denotes a section that applies to the Windows NT platform

Denotes a section that does not apply to OS/390 Component Broker

How to read the syntax diagrams

The following rules apply to the syntax diagrams in this section:

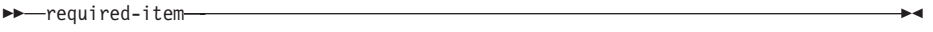
- Read the syntax diagrams from left to right, from top to bottom, following the path of the line.
- The following symbols are used are used in the syntax diagrams.

  ▸▸– Indicates the beginning of a statement.

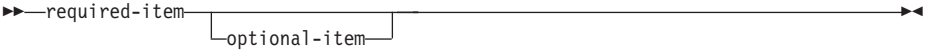
  –▸ Indicates that the statement is continued on the next line.

  ▸– Indicates that the statement is continued from the previous line.

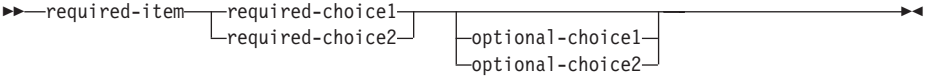
  –▸◂ Indicates the end of the statement.
- Diagrams of syntactical units other than complete statements start with |– and end with –|.
- Required items are on the main path.



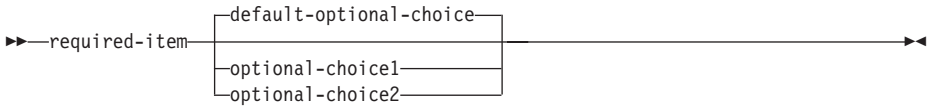
- Optional items are below the main path.



- Choice items are in a stack. If required, on main path; if optional, below main path.



- The default item of a choice is above the main path.



- Repeated items have an arrow returning to the left above the main line.
- Keywords are in uppercase (for example, KEY). Each must be spelled as shown.
- Variables are in lowercase (for example, variable-name). Each represents a user-supplied name or value.
- Inclusive punctuation is part of the syntax statement.

Accessing Java ORB and Object Services API documentation

HTML documentation for the ORB and object services bindings is available by unzipping the file apidocs.zip, located in the subdirectory doc\client\javacl under the directory where the Java Client is installed. Using a browser, open the HTML file *packages.html* to view a listing of available Java packages for the Java client API.

How to send your comments

Your feedback is important in helping to provide the most accurate and highest quality information. If you have any comments about this book or any other WebSphere Application Server Enterprise Edition documentation, send your comments by e-mail to waseedoc@us.ibm.com. Be sure to include the name of the book, the document number, the version of WebSphere Application Server Enterprise Edition, and, if applicable, the specific location of the information on which you are commenting (for example, a page number or a table number).

Cross-reference tables for finding classes and interfaces

This section provides two different tables to help you quickly locate a class or interface.

Interfaces and classes by module and service

Table 1 identifies the

- modules associated with the Object Request Broker, the Managed Object Framework and each Object Service
- classes and interfaces associated with each module
- database data type classes

Table 1. Classes and interfaces by module and service

Service	Module Name	Class/Interface
Concurrency Service	"CosConcurrencyControl Module" on page 287	• "LockCoordinator Interface" on page 287 • "LockSet Interface" on page 289 • "LockSetFactory Interface" on page 296 • "TransactionalLockSet Interface" on page 301
Database Data Type Classes	Not Applicable	• "ICBCDate Class" on page 864 • "ICBCDecimal Class" on page 879 • "ICBCDuration Class" on page 903 • "ICBCTime Class" on page 910 • "ICBCTimestamp Class" on page 918
Event Service	"CosEventComm Module" on page 327	• "PullConsumer Interface" on page 328 • "PullSupplier Interface" on page 329 • "PushConsumer Interface" on page 333 • "PushSupplier Interface" on page 336
	"CosEventChannelAdmin Module" on page 309	• "ConsumerAdmin Interface" on page 310 • "EventChannel Interface" on page 313 • "ProxyPullConsumer Interface" on page 315 • "ProxyPullSupplier Interface" on page 317 • "ProxyPushConsumer Interface" on page 319 • "ProxyPushSupplier Interface" on page 321 • "SupplierAdmin Interface" on page 323
	"IExtendedEventChannelAdmin Module" on page 569	• "IEventChannelHome Interface" on page 569

Table 1. Classes and interfaces by module and service (continued)

Service	Module Name	Class/Interface
Externalization Service	"CosStream Module" on page 467	• "Streamable Interface" on page 467 • "StreamIO Interface" on page 470
	"IExtendedStream Module" on page 647	• "StreamIO Interface" on page 647
Identity Service	"CosObjectIdentity Module" on page 457	• "IdentifiableObject Interface" on page 457
LifeCycle Service	"CosLifeCycle Module" on page 339	• "FactoryFinder Interface" on page 339 • "GenericFactory Interface" on page 341 • "LifeCycleObject Interface" on page 343
	"IExtendedLifeCycle Module" on page 575	• "FactoryFinder Interface" on page 576 • "FactoryFinderHome Interface" on page 581 • "Location Interface" on page 584 • "OrderedLocation Interface" on page 586 • "OrderedLocationHome Interface" on page 587 • "ScopeManipulator Interface" on page 593 • "SingleLocation Interface" on page 596 • "SingleLocationHome Interface" on page 598
	"ILifeCycleLocalObjectImpl Module" on page 649	• "FactoryFinder Interface" on page 649 • "ILocalOnly Interface" on page 659 • "OrderedLocation Interface" on page 660 • "ScopeManipulator Interface" on page 667 • "SingleLocation Interface" on page 669
	"ILifeCycleManagedClient Module" on page 675	• "FactoryFinder Interface" on page 675 • "FactoryFinderHome Interface" on page 676 • "OrderedLocation Interface" on page 676 • "OrderedLocationHome Interface" on page 676 • "SingleLocation Interface" on page 677 • "SingleLocationHome Interface" on page 677

Table 1. Classes and interfaces by module and service (continued)

Service	Module Name	Class/Interface
Managed Object Framework	"CBSeriesGlobal Module" on page 1	• "CBSeriesGlobal Interface" on page 1
	"IBOIMExtLocal Module" on page 525	• "IUUIDCopyHelperBase Interface" on page 526 • "IUUIDPrimaryKey Interface" on page 526
	"IBOIMExtLocalToServer Module" on page 529	• "IDataObject Interface" on page 530 • "IHomeAwareDataObject Interface" on page 534 • "IQueryableDataObject Interface" on page 535
	"IBOIMInstanceManagerFriendQOS Module" on page 537	• "IMForMixinAndMO Interface" on page 538 • "IMLocalFriend Interface" on page 538 • "IMMixin Interface" on page 539 • "IMOnlyForMO Interface" on page 539
	"IBOIMManagedObjectFriendQOS Module" on page 541	• Not applicable
	"IBOIMServiceFriendQOS Module" on page 543	• Not applicable

Table 1. Classes and interfaces by module and service (continued)

Service	Module Name	Class/Interface
Managed Object Framework (continued)	"ICollectionsBase Module" on page 545	• "IIterator Interface" on page 546 • "IMIterable Interface" on page 551 • "IMIterableUpdate Interface" on page 551 • "IMKeyable Interface" on page 553 • "IMKeyedUpdate Interface" on page 555 • "IMNonKeyedUpdate Interface" on page 557 • "IMNonUniqueKeyable Interface" on page 559 • "IMOrderedIterableAdd Interface" on page 559 • "IMOrderedNonKeyedAdd Interface" on page 560 • "IMQueryable Interface" on page 562 • "IMTyped Interface" on page 565 • "ITwoWayIterator Interface" on page 566
	"IManagedAdvancedClient Module" on page 679	• "IIterableHome Interface" on page 679 • "IQueryableIterableHome Interface" on page 680 • "IView Interface" on page 680
	"IManagedAdvancedServer Module" on page 683	• "ISpecializedHome Interface" on page 684 • "ISpecializedHomeDataObject Interface" on page 684 • "ISpecializedHomeManagedObject Interface" on page 685 • "ISpecializedIterableHome Interface" on page 685 • "ISpecializedIterableHomeManagedObject Interface" on page 685 • "ISpecializedQueryableIterableHome Interface" on page 686 • "ISpecializedQueryableIterableHomeManagedObject Interface" on page 686 • "IUUIDCopyHelperBase Interface" on page 686 • "IUUIDPrimaryKey Interface" on page 687

Table 1. Classes and interfaces by module and service (continued)

Service	Module Name	Class/Interface
Managed Object Framework (continued)	"IManagedClient Module" on page 689	• "IHome Interface" on page 690 • "IManageable Interface" on page 695
	"IManagedCollections Module" on page 701	• "ICollectionHome Interface" on page 702 • "ICommonCollection Interface" on page 703 • "IKeyedReferenceCollection Interface" on page 705 • "IReferenceCollection Interface" on page 705
	"IManagedLocal Module" on page 707	• "IHandle Interface" on page 708 • "ILocalOnly Interface" on page 712 • "INonManageable Interface" on page 713 • "IPrimaryKey Interface" on page 715 • "IUniqueKey Interface" on page 715
	"IManagedServer Module" on page 717	• "CBSeriesGlobal Interface" on page 1 • "IDataObject Interface" on page 718 • "IManagedObject Interface" on page 718 • "IManagedObjectWithCachedDataObject Interface" on page 719 • "IManagedObjectWithDataObject Interface" on page 723 • "IManagedObjectWithoutDataObject Interface" on page 726 • "IWrappable Interface" on page 728
	"IRDBIMExtLocalToServer Module" on page 737	• "ICachingServiceDataObject Interface" on page 737 • "IDataObject Interface" on page 738
Naming Service	"CosNaming Module" on page 347	• "BindingIterator Interface" on page 347 • "NamingContext Interface" on page 353
	"IExtendedNaming Module" on page 601	• "BindingStringIterator Interface" on page 602 • "NamingContext Interface" on page 607
	"IExtendedNamingStringSyntax Module" on page 619	• "StandardSyntaxModel Interface" on page 619
	"NamingStringSyntax Module" on page 769	• "StandardSyntaxModel Interface" on page 769 • "StringName Interface" on page 774

Table 1. Classes and interfaces by module and service (continued)

Service	Module Name	Class/Interface
Notification Service	"CosNotification Module" on page 367	• "StructuredEvent Data Structure" on page 370 • "QoSAdmin Interface" on page 373
	"CosNotifyFilter Module" on page 435	• "Filter Interface" on page 438 • "FilterAdmin Interface" on page 449 • "FilterFactory Interface" on page 453
	"CosNotifyComm Module" on page 423	• "StructuredPullConsumer Interface" on page 425 • "StructuredPullSupplier Interface" on page 426 • "StructuredPushConsumer Interface" on page 429 • "StructuredPushSupplier Interface" on page 432
	"CosNotifyChannelAdmin Module" on page 377	• "ConsumerAdmin Interface" on page 382 • "EventChannel Interface" on page 388 • "EventChannelFactory Interface" on page 397 • "ProxyConsumer Interface" on page 401 • "ProxySupplier Interface" on page 403 • "StructuredProxyPullConsumer Interface" on page 405 • "StructuredProxyPullSupplier Interface" on page 407 • "StructuredProxyPushConsumer Interface" on page 409 • "StructuredProxyPushSupplier Interface" on page 411 • "SupplierAdmin Interface" on page 415
	"INotifyChannelAdminManagedClient Module" on page 729	• "EventChannelFactory Interface" on page 729
	"INotifyFilterManagedClient Module" on page 735	• None documented
Query Service	"CosQuery Module" on page 461	• "QueryEvaluator Interface" on page 461
	"CosQueryCollection Module" on page 465	• Not applicable
	"IExtendedQuery Module" on page 621	• "DataArrayDescriptor Interface" on page 621 • "DataArrayIterator Interface" on page 624 • "ParameterListBuilder Interface" on page 628 • "QueryEvaluator Interface" on page 633

Table 1. Classes and interfaces by module and service (continued)

Service	Module Name	Class/Interface
Security Service	"IExtendedSecurity Module" on page 639	• "Credentials Interface" on page 640 • "Current Interface" on page 640 • "LoginHelper Interface" on page 641 • "Principal Interface" on page 643
	"IExtendedSecurityClient Module" on page 645	• "SecurableObject Interface" on page 645
	"Security Module" on page 777	• Not applicable
	"SecurityLevel1 Module" on page 783	• "Current Interface" on page 783
	"SecurityLevel2 Module" on page 787	• "Credentials Interface" on page 788 • "Current Interface" on page 796 • "PrincipalAuthenticator Interface" on page 800
Session Service	"ISessions Module" on page 741	• "Control Interface" on page 743 • "Coordinator Interface" on page 744 • "Current Interface" on page 749 • "Resource Interface" on page 762 • "SessionableObject Interface" on page 766
Transaction Service	"CosTransactions Module" on page 483	• "Control Interface" on page 485 • "Coordinator Interface" on page 488 • "Current Interface" on page 499 • "RecoveryCoordinator Interface" on page 519 • "Resource Interface" on page 519 • "Synchronization Interface" on page 519 • "Terminator Interface" on page 520 • "TransactionalObject Interface" on page 523 • "TransactionFactory Interface" on page 523

Listing of interfaces and classes

Table 2 provides an alphabetical listing of class and interface names and identifies the module each is associated with.

Table 2. Alphabetical listing of interfaces and classes

Interface or Class Name	Module Name
“AliasDef Interface” on page 11	“CORBA Module” on page 7
“Any Class” on page 13	“CORBA Module” on page 7
“ArrayDef Interface” on page 21	“CORBA Module” on page 7
“AttributeDef Interface” on page 26	“CORBA Module” on page 7
“BindingIterator Interface” on page 347	“CosNaming Module” on page 347
“BindingStringIterator Interface” on page 602	“IExtendedNaming Module” on page 601
“BOA Class” on page 30	“CORBA Module” on page 7
“CBSeriesGlobal Interface” on page 1	“CBSeriesGlobal Module” on page 1
“ConstantDef Interface” on page 43	“CORBA Module” on page 7
“ConsumerAdmin Interface” on page 310	“CosEventChannelAdmin Module” on page 309
“ConsumerAdmin Interface” on page 382	“CosNotifyChannelAdmin Module” on page 377
“Contained Interface” on page 47	“CORBA Module” on page 7
“Container Interface” on page 56	“CORBA Module” on page 7
“Context Class” on page 76	“CORBA Module” on page 7
“ContextList Class” on page 83	“CORBA Module” on page 7
“Control Interface” on page 485	“CosTransactions Module” on page 483
“Control Interface” on page 743	“ISessions Module” on page 741
“Coordinator Interface” on page 488	“CosTransactions Module” on page 483
“Coordinator Interface” on page 744	“ISessions Module” on page 741
“CORBA Class” on page 88	“CORBA Module” on page 7
“Credentials Interface” on page 640	“IExtendedSecurity Module” on page 639
“Credentials Interface” on page 788	“SecurityLevel2 Module” on page 787
“Current Class” on page 102	“CORBA Module” on page 7
“Current Interface” on page 499	“CosTransactions Module” on page 483
“Current Interface” on page 640	“IExtendedSecurity Module” on page 639
“Current Interface” on page 749	“ISessions Module” on page 741
“Current Interface” on page 783	“SecurityLevel1 Module” on page 783
“Current Interface” on page 796	“SecurityLevel2 Module” on page 787
“DataArrayDescriptor Interface” on page 621	“IExtendedQuery Module” on page 621
“DataArrayIterator Interface” on page 624	“IExtendedQuery Module” on page 621
“DynamicImplementation Class” on page 104	“CORBA Module” on page 7
“EnumDef Interface” on page 106	“CORBA Module” on page 7
“Environment Class” on page 108	“CORBA Module” on page 7
“EventChannel Interface” on page 313	“CosEventChannelAdmin Module” on page 309
“EventChannel Interface” on page 388	“CosNotifyChannelAdmin Module” on page 377
“EventChannelFactory Interface” on page 397	“CosNotifyChannelAdmin Module” on page 377

Table 2. Alphabetical listing of interfaces and classes (continued)

Interface or Class Name	Module Name
“EventChannelFactory Interface” on page 729	“INotifyChannelAdminManagedClient Module” on page 729
“Exception Class” on page 110	“CORBA Module” on page 7
“ExceptionDef Interface” on page 112	“CORBA Module” on page 7
“ExceptionList Class” on page 116	“CORBA Module” on page 7
“FactoryFinder Interface” on page 339	“CosLifeCycle Module” on page 339
“FactoryFinder Interface” on page 576	“IExtendedLifeCycle Module” on page 575
“FactoryFinder Interface” on page 649	“ILifeCycleLocalObjectImpl Module” on page 649
“FactoryFinder Interface” on page 675	“ILifeCycleManagedClient Module” on page 675
“FactoryFinderHome Interface” on page 581	“IExtendedLifeCycle Module” on page 575
“FactoryFinderHome Interface” on page 676	“ILifeCycleManagedClient Module” on page 675
“FilterAdmin Interface” on page 449	“CosNotifyFilter Module” on page 435
“Filter Interface” on page 438	“CosNotifyFilter Module” on page 435
“FilterFactory Interface” on page 453	“CosNotifyFilter Module” on page 435
“GenericFactory Interface” on page 341	“CosLifeCycle Module” on page 339
“ICachingServiceDataObject Interface” on page 737	“IRDBIMExtLocalToServer Module” on page 737
“ICBCDate Class” on page 864	ICBC Data Type Classes
“ICBCDecimal Class” on page 879	ICBC Data Type Classes
“ICBCDuration Class” on page 903	ICBC Data Type Classes
“ICBCTime Class” on page 910	ICBC Data Type Classes
“ICBCTimestamp Class” on page 918	ICBC Data Type Classes
“ICollectionHome Interface” on page 702	“IManagedCollections Module” on page 701
“ICommonCollection Interface” on page 703	“IManagedCollections Module” on page 701
“IDataObject Interface” on page 530	“IBOIMExtLocalToServer Module” on page 529
“IDataObject Interface” on page 718	“IManagedServer Module” on page 717
“IDataObject Interface” on page 738	“IRDBIMExtLocalToServer Module” on page 737
“IdentifiableObject Interface” on page 457	“CosObjectIdentity Module” on page 457
“IDLType Interface” on page 121	“CORBA Module” on page 7
“IEventChannelHome Interface” on page 569	“IExtendedEventChannelAdmin Module” on page 569
“IHandle Interface” on page 708	“IManagedLocal Module” on page 707
“IHome Interface” on page 690	“IManagedClient Module” on page 689
“IHomeAwareDataObject Interface” on page 534	“IBOIMExtLocalToServer Module” on page 529
“IIterableHome Interface” on page 679	“IManagedAdvancedClient Module” on page 679
“IIterator Interface” on page 546	“ICollectionsBase Module” on page 545
“IKey Interface” on page 709	“IManagedLocal Module” on page 707
“IKeyedReferenceCollection Interface” on page 705	“IManagedCollections Module” on page 701
“ILocalOnly Interface” on page 659	“ILifeCycleLocalObjectImpl Module” on page 649

Table 2. Alphabetical listing of interfaces and classes (continued)

Interface or Class Name	Module Name
“ILocalOnly Interface” on page 712	“IManagedLocal Module” on page 707
“IManageable Interface” on page 695	“IManagedClient Module” on page 689
“IManagedObject Interface” on page 718	“IManagedServer Module” on page 717
“IManagedObjectWithCachedDataObject Interface” on page 719	“IManagedServer Module” on page 717
“IManagedObjectWithDataObject Interface” on page 723	“IManagedServer Module” on page 717
“IManagedObjectWithoutDataObject Interface” on page 726	“IManagedServer Module” on page 717
“IMForMixinAndMO Interface” on page 538	“IBOIMInstanceManagerFriendQOS Module” on page 537
“IMIterable Interface” on page 551	“ICollectionsBase Module” on page 545
“IMIterableUpdate Interface” on page 551	“ICollectionsBase Module” on page 545
“IMKeyable Interface” on page 553	“ICollectionsBase Module” on page 545
“IMKeyedUpdate Interface” on page 555	“ICollectionsBase Module” on page 545
“IMLocalFriend Interface” on page 538	“IBOIMInstanceManagerFriendQOS Module” on page 537
“IMMixin Interface” on page 539	“IBOIMInstanceManagerFriendQOS Module” on page 537
“IMNonKeyedUpdate Interface” on page 557	“ICollectionsBase Module” on page 545
“IMNonUniqueKeyable Interface” on page 559	“ICollectionsBase Module” on page 545
“IMOnlyForMO Interface” on page 539	“IBOIMInstanceManagerFriendQOS Module” on page 537
“IMOrderedIterableAdd Interface” on page 559	“ICollectionsBase Module” on page 545
“IMOrderedNonKeyedAdd Interface” on page 560	“ICollectionsBase Module” on page 545
“ImplementationDef Interface” on page 126	“CORBA Module” on page 7
“ImplRepository Class” on page 123	“CORBA Module” on page 7
“IMQueryable Interface” on page 562	“ICollectionsBase Module” on page 545
“IMTyped Interface” on page 565	“ICollectionsBase Module” on page 545
“INonManageable Interface” on page 713	“IManagedLocal Module” on page 707
“InterfaceDef Interface” on page 128	“CORBA Module” on page 7
“IPrimaryKey Interface” on page 715	“IManagedLocal Module” on page 707
“IQueryableDataObject Interface” on page 535	“IBOIMExtLocalToServer Module” on page 529
“IQueryableIterableHome Interface” on page 680	“IManagedAdvancedClient Module” on page 679
“IRObject Interface” on page 138	“CORBA Module” on page 7
“IReferenceCollection Interface” on page 705	“IManagedCollections Module” on page 701
“ISpecializedHome Interface” on page 684	“IManagedAdvancedServer Module” on page 683
“ISpecializedHomeDataObject Interface” on page 684	“IManagedAdvancedServer Module” on page 683
“ISpecializedHomeManagedObject Interface” on page 685	“IManagedAdvancedServer Module” on page 683
“ISpecializedIterableHome Interface” on page 685	“IManagedAdvancedServer Module” on page 683

Table 2. Alphabetical listing of interfaces and classes (continued)

Interface or Class Name	Module Name
“ISpecializedIterableHomeManagedObject Interface” on page 685	“IManagedAdvancedServer Module” on page 683
“ISpecializedQueryableIterableHome Interface” on page 686	“IManagedAdvancedServer Module” on page 683
“ISpecializedQueryableIterableHomeManagedObject Interface” on page 686	“IManagedAdvancedServer Module” on page 683
“ITwoWayIterator Interface” on page 566	“ICollectionsBase Module” on page 545
“IUniqueKey Interface” on page 715	“IManagedLocal Module” on page 707
“IUUIDCopyHelperBase Interface” on page 526	“IBOIMExtLocal Module” on page 525
“IUUIDCopyHelperBase Interface” on page 686	“IManagedAdvancedServer Module” on page 683
“IUUIDPrimaryKey Interface” on page 526	“IBOIMExtLocal Module” on page 525
“IUUIDPrimaryKey Interface” on page 687	“IManagedAdvancedServer Module” on page 683
“IView Interface” on page 680	“IManagedAdvancedClient Module” on page 679
“IWrappable Interface” on page 728	“IManagedServer Module” on page 717
“LifeCycleObject Interface” on page 343	“CosLifeCycle Module” on page 339
“Location Interface” on page 584	“IExtendedLifeCycle Module” on page 575
“LockCoordinator Interface” on page 287	“CosConcurrencyControl Module” on page 287
“LockSet Interface” on page 289	“CosConcurrencyControl Module” on page 287
“LockSetFactory Interface” on page 296	“CosConcurrencyControl Module” on page 287
“LoginHelper Interface” on page 641	“IExtendedSecurity Module” on page 639
“ModuleDef Interface” on page 141	“CORBA Module” on page 7
“NamedValue Class” on page 143	“CORBA Module” on page 7
“NamingContext Interface” on page 353	“CosNaming Module” on page 347
“NamingContext Interface” on page 607	“IExtendedNaming Module” on page 601
“NVList Class” on page 147	“CORBA Module” on page 7
“Object Class” on page 156	“CORBA Module” on page 7
“OperationDef Interface” on page 208	“CORBA Module” on page 7
“ORB Class” on page 171	“CORBA Module” on page 7
“OrderedLocation Interface” on page 586	“IExtendedLifeCycle Module” on page 575
“OrderedLocation Interface” on page 660	“ILifeCycleLocalObjectImpl Module” on page 649
“OrderedLocation Interface” on page 676	“ILifeCycleManagedClient Module” on page 675
“OrderedLocationHome Interface” on page 587	“IExtendedLifeCycle Module” on page 575
“OrderedLocationHome Interface” on page 676	“ILifeCycleManagedClient Module” on page 675
“ParameterListBuilder Interface” on page 628	“IExtendedQuery Module” on page 621
“Policy Interface” on page 218	“CORBA Module” on page 7
“PrimitiveDef Interface” on page 218	“CORBA Module” on page 7
“Principal Interface” on page 220	“CORBA Module” on page 7
“PrincipalAuthenticator Interface” on page 800	“SecurityLevel2 Module” on page 787

Table 2. Alphabetical listing of interfaces and classes (continued)

Interface or Class Name	Module Name
“ProxyConsumer Interface” on page 401	“CosNotifyChannelAdmin Module” on page 377
“ProxyPullConsumer Interface” on page 315	“CosEventChannelAdmin Module” on page 309
“ProxyPullSupplier Interface” on page 317	“CosEventChannelAdmin Module” on page 309
“ProxyPushConsumer Interface” on page 319	“CosEventChannelAdmin Module” on page 309
“ProxyPushSupplier Interface” on page 321	“CosEventChannelAdmin Module” on page 309
“ProxySupplier Interface” on page 403	“CosNotifyChannelAdmin Module” on page 377
“PullConsumer Interface” on page 328	“CosEventComm Module” on page 327
“PullSupplier Interface” on page 329	“CosEventComm Module” on page 327
“PushConsumer Interface” on page 333	“CosEventComm Module” on page 327
“PushSupplier Interface” on page 336	“CosEventComm Module” on page 327
“QoSAdmin Interface” on page 373	“CosNotification Module” on page 367
“QueryEvaluator Interface” on page 461	“CosQuery Module” on page 461
“QueryEvaluator Interface” on page 633	“IExtendedQuery Module” on page 621
“RecoveryCoordinator Interface” on page 519	“CosTransactions Module” on page 483
“Repository Interface” on page 220	“CORBA Module” on page 7
“Request Class” on page 228	“CORBA Module” on page 7
“RequestSeq Class” on page 241	“CORBA Module” on page 7
“Resource Interface” on page 519	“CosTransactions Module” on page 483
“Resource Interface” on page 762	“ISessions Module” on page 741
“ScopeManipulator Interface” on page 593	“IExtendedLifeCycle Module” on page 575
“ScopeManipulator Interface” on page 667	“ILifeCycleLocalObjectImpl Module” on page 649
“SecurableObject Interface” on page 645	“IExtendedSecurityClient Module” on page 645
“SequenceDef Interface” on page 245	“CORBA Module” on page 7
“ServerRequest Class” on page 249	“CORBA Module” on page 7
“SessionableObject Interface” on page 766	“ISessions Module” on page 741
“SingleLocation Interface” on page 596	“IExtendedLifeCycle Module” on page 575
“SingleLocation Interface” on page 669	“ILifeCycleLocalObjectImpl Module” on page 649
“SingleLocation Interface” on page 677	“ILifeCycleManagedClient Module” on page 675
“SingleLocationHome Interface” on page 598	“IExtendedLifeCycle Module” on page 575
“SingleLocationHome Interface” on page 677	“ILifeCycleManagedClient Module” on page 675
“StandardSyntaxModel Interface” on page 619	“IExtendedNamingStringSyntax Module” on page 619
“StandardSyntaxModel Interface” on page 769	“NamingStringSyntax Module” on page 769
“Streamable Interface” on page 467	“CosStream Module” on page 467
“StreamIO Interface” on page 470	“CosStream Module” on page 467
“StreamIO Interface” on page 647	“IExtendedStream Module” on page 647
“StringDef Interface” on page 254	“CORBA Module” on page 7

Table 2. Alphabetical listing of interfaces and classes (continued)

Interface or Class Name	Module Name
“StringName Interface” on page 774	“NamingStringSyntax Module” on page 769
“StructDef Interface” on page 256	“CORBA Module” on page 7
“StructuredEvent Data Structure” on page 370	“CosNotification Module” on page 367
“StructuredProxyPullConsumer Interface” on page 405	“CosNotifyChannelAdmin Module” on page 377
“StructuredProxyPullSupplier Interface” on page 407	“CosNotifyChannelAdmin Module” on page 377
“StructuredProxyPushConsumer Interface” on page 409	“CosNotifyChannelAdmin Module” on page 377
“StructuredProxyPushSupplier Interface” on page 411	“CosNotifyChannelAdmin Module” on page 377
“StructuredPullConsumer Interface” on page 425	“CosNotifyComm Module” on page 423
“StructuredPullSupplier Interface” on page 426	“CosNotifyComm Module” on page 423
“StructuredPushConsumer Interface” on page 429	“CosNotifyComm Module” on page 423
“StructuredPushSupplier Interface” on page 432	“CosNotifyComm Module” on page 423
“SupplierAdmin Interface” on page 323	“CosEventChannelAdmin Module” on page 309
“SupplierAdmin Interface” on page 415	“CosNotifyChannelAdmin Module” on page 377
“Synchronization Interface” on page 519	“CosTransactions Module” on page 483
“SystemException Class” on page 259	“CORBA Module” on page 7
“Terminator Interface” on page 520	“CosTransactions Module” on page 483
“TransactionalLockSet Interface” on page 301	“CosConcurrencyControl Module” on page 287
“TransactionalObject Interface” on page 523	“CosTransactions Module” on page 483
“TransactionFactory Interface” on page 523	“CosTransactions Module” on page 483
“TypeCode Class” on page 262	“CORBA Module” on page 7
“TypedefDef Interface” on page 272	“CORBA Module” on page 7
“UnionDef Interface” on page 275	“CORBA Module” on page 7
“UnknownUserException Class” on page 279	“CORBA Module” on page 7
“UserException Class” on page 282	“CORBA Module” on page 7
“WstringDef Interface” on page 283	“CORBA Module” on page 7

Chapter 1. CBSeriesGlobal in the Managed Object Framework

The other modules in the Managed Object Framework are:

- IBOIMExtLocal
- IBOIMExtLocalToServer
- IBOIMInstanceManagerFriendQOS
- IBOIMManagedObjectFriendQOS
- IBOIMServiceFriendQOS
- ICollectionsBase
- IManagedAdvancedClient
- IManagedAdvancedServer
- IManagedClient
- IManagedCollections
- IManagedLocal
- IManagedServer
- IRDBIMExtLocalToServer

CBSeriesGlobal Module

This module defines the convenience utilities for the Managed Object Framework. The interfaces in this module are intended to be used in all aspects of the application. Interfaces are provided in VisualAge C++, Microsoft C++, and Java.

File name

CBSeriesGlobal.hh for C++
com.ibm.CBCUtil.CBSeriesGlobal in somojor.zip for Java

Interfaces

CBSeriesGlobal Interface

CBSeriesGlobal Interface

This interface provides convenience methods for using Component Broker in a C++, Java BOs, and Java client environments. Initialization methods are provided specifically for C++, Java application, and Java applet environments. The hostName(), nameService(), and orb() methods are available in all environments. The serverName() method is only valid for those applications that are running in the Component Broker server process. It will return an empty string if called in other environments.

CBSeriesGlobal is a convenience interface that is required for most client programs. If the client program uses either a copy helper or primary key class with an attribute that is an object reference, then initializing CBSeriesGlobal is a requirement. This is because the implementation of the copy helper and primary key depend on CBSeriesGlobal:orb() when using the ORB object_to_string() operation. It is also a requirement to use CBSeriesGlobal if security or object handles will be used in he Component Broker client. In general it is a good idea to use CBSeriesGlobal in all Component Broker client programs.

Note: This interface does not conform to standard CORBA programming style. Actual pointers to data internal to CBSeriesGlobal is returned instead of copies of the data. Therefore, the application program can unintentionally modify these values and cause subsequent program failures. Follow these rules when using CBSeriesGlobal:

- Return the values into an _ptr instead of an _var. When an _var goes out of scope, a release() operation will be performed on it. This operation will release the value maintained by CBSeriesGlobal and make it an invalid value.
- Never perform a release() operation on it.
- Never modify the value of the _ptr.

Supported operations

CBSeriesGlobal::hostName
CBSeriesGlobal::Initialize
CBSeriesGlobal::nameService
CBSeriesGlobal::orb
CBSeriesGlobal::serverName

CBSeriesGlobal::hostName

This operation returns a CORBA::String that contains the name of the host machine. This value can be used for information purposes when displaying additional information; for example, when debugging. One of the initialization methods must be called before this operation is called.

Original interface

CBSeriesGlobal

IDL syntax

```
CORBA::String hostName();
```

Return value

CORBA::String
The name of the host machine.

CBSeriesGlobal::Initialize

This operation performs all the necessary initialization of CBSeriesGlobal so that the remaining operations have available information when they are called. This operation can be performed more than once, if necessary, but it should only be performed once per process.

Invoking the Initialize() operation is the only operation required to initialize the C++ client environment. For Java applications, either Initialize(String host), Initialize(String

host, String port), Initialize(String[] args), Initialize(String[] args, Properties props) must be called. For Java applets, either Initialize(Applet applet), or Initialize(Applet applet, Properties props) must be called. Once one of these initialization methods have been called the values in CBSeriesGlobal are set and cannot be updated. None of the initialization methods should be called in server code because the Component Broker frameworks have already initialized the environment.

Original interface

CBSeriesGlobal

IDL syntax

```
static void Initialize();
static void Initialize(String host);
static void Initialize(String host, String port);
static void Initialize(String[] args);
static void Initialize(String[] args, Properties props);
static void Initialize(Applet applet);
static void Initialize(Applet applet, Properties props);
```

Parameters

applet The Java Applet in which CBSeriesGlobal is initializing the Java ORB

args The list of arguments from the command line which need to be passed to the ORB initialization

host The host on which the CBConnector naming services' Bootstrap Daemon is running (the property that is used is `com.ibm.CORBA.BootstrapHost`)

port The port number on the CBConnector naming services' Bootstrap Daemon that should be used instead of the default of *900* (the property that is used is `com.ibm.CORBA.BootstrapPort`)

props The Java Properties which need to be passed to the ORB initialization. See the appendix "Java ORB properties" in the *Component Broker Programming Guide* for more information.

CBSeriesGlobal::nameService

This operation returns an IExtendedNaming::NamingContext_ptr. This pointer can be used to obtain additional information. This pointer is primarily used for finding managed objects using the FactoryFinder interfaces. One of the initialization methods must be called before this operation is called.

Original interface

CBSeriesGlobal

IDL syntax

```
static IExtendedNaming::NamingContext_ptr nameService();
```

Return value

IExtendedNaming::NamingContext_ptr
A pointer to an instance of the IExtendedNaming::NamingContext being used.

CBSeriesGlobal::orb

This operation returns a CORBA::ORB_ptr. This pointer can be used to obtain additional information; for example, to obtain a pointer to the CORBA::Current object for transactions processing. One of the initialization methods must be called before this operation is called.

Original interface

CBSeriesGlobal

IDL syntax

```
CORBA::ORB_ptr orb();
```

Return value

CORBA::ORB_ptr
A pointer to an instance of the CORBA::ORB being used.

CBSeriesGlobal::serverName

This operation returns a CORBA::String that contains the generic name of the server within which this operation is running. The return value corresponds to the name of the server definition supplied to Systems Management. On OS/390, this will not be the specific server instance in a specific address space or process. This operation is only valid within server-side code. This value can be used for informational purposes when displaying additional information; for example, when debugging. One of the initialization methods must be called before this operation is called.

Original interface

CBSeriesGlobal

IDL syntax

```
CORBA::String serverName();
```

Return value

CORBA::String

The name of the server on which this operation is run.

Chapter 2. CORBA in Object Request Broker

The CORBA Module is the only module in Object Request Broker.

CORBA Module

Encompasses the interfaces that make up the CORBA-compliant ORB, the TypeCode library, and the Interface Repository Framework.

File name

orb.idl
corba.h

Intended usage

The interfaces within this module are intended to be used to write CORBA-compliant, distributed client-server applications, in which objects can be accessed across address spaces, even across different machines. These interfaces constitute a CORBA-compliant Object Request Broker (ORB), a standardized transport for distributed object interaction. CORBA is an industry standard defined by the Object Management Group (OMG) consortium.

The TypeCode and Interface Repository interfaces contained in the CORBA module are intended to be used to write client applications using the Dynamic Invocation Interface (wherein the interfaces to be used by the client are not know at compile time). The TypeCode library provides run-time access to descriptions of IDL data types. The Interface Repository Framework allows run-time access to information specified in IDL.

The portions of the CORBA module that can be referenced in application-specific IDL is contained in orb.idl. The C++ language mapping for the CORBA module is contained in corba.h. This file includes not only C++ mappings for the interfaces defined in orb.idl, but also C++ mappings for CORBA *pseudo-objects* (objects that cannot be accessed remotely nor referenced in application IDL, but which provide services used in-process by client and server applications).

Types

```
typedef sequence<octet, 1024> ReferenceData;
typedef string ScopedName;
typedef string RepositoryId;
typedef string Identifier;
typedef string VersionSpec;
typedef sequence<InterfaceDef> InterfaceDefSeq;
typedef sequence<Contained> ContainedSeq;
typedef sequence<StructMember> StructMemberSeq;
typedef sequence<UnionMember> UnionMemberSeq;
typedef sequence<Identifier> EnumMemberSeq;
typedef sequence<ParameterDescription> ParDescriptionSeq;
typedef Identifier ContextIdentifier;
```

```
typedef sequence<ContextIdentifier> ContextIdSeq;
typedef sequence<ExceptionDef> ExceptionDefSeq;
typedef sequence<ExceptionDescription> ExcDescriptionSeq;
typedef sequence<RepositoryId> RepositoryIdSeq;
typedef sequence<OperationDescription> OpDescriptionSeq;
typedef sequence<AttributeDescription> AttrDescriptionSeq;
struct StructMember
{
   Identifier name;
   TypeCode   type;
   IDLType    type_def;
};
struct UnionMember
{
   Identifier name;
   any        label;
   TypeCode   type;
   IDLType    type_def;
};
struct ModuleDescription
{
   Identifier   name;
   RepositoryId id;
   RepositoryId defined_in;
   VersionSpec  version;
};
struct ConstantDescription
{
   Identifier   name;
   RepositoryId id;
   RepositoryId defined_in;
   VersionSpec  version;
   TypeCode     type;
   any          value;
};
struct TypeDescription
{
   Identifier   name;
   RepositoryId id;
   RepositoryId defined_in;
   VersionSpec  version;
   TypeCode     type;
};
struct ExceptionDescription
{
   Identifier   name;
   RepositoryId id;
   RepositoryId defined_in;
   VersionSpec  version;
   TypeCode     type;
};
struct AttributeDescription
{
   Identifier    name;
   RepositoryId  id;
```

```
   RepositoryId  defined_in;
   VersionSpec   version;
   TypeCode      type;
   AttributeMode mode;
};
struct ParameterDescription
{
   Identifier    name;
   TypeCode      type;
   IDLType       type_def;
   ParameterMode mode;
};
struct OperationDescription
{
   Identifier        name;
   RepositoryId      id;
   RepositoryId      defined_in;
   VersionSpec       version;
   TypeCode          result;
   OperationMode     mode;
   ContextIdSeq      contexts;
   ParDescriptionSeq parameters;
   ExcDescriptionSeq exceptions;
};
struct InterfaceDescription
{
   Identifier      name;
   RepositoryId    id;
   RepositoryId    defined_in;
   VersionSpec     version;
   RepositoryIdSeq base_interfaces;
};
enum TCKind
{
   tk_null,     tk_void,
   tk_short,    tk_long,      tk_ushort,   tk_ulong,
   tk_float,    tk_double,    tk_boolean,  tk_char,
   tk_octet,    tk_any,       tk_TypeCode, tk_Principal, tk_objref,
   tk_struct,   tk_union,     tk_enum,     tk_string,
   tk_sequence, tk_array,     tk_alias,    tk_except,
   tk_longlong, tk_ulonglong,
   tk_wchar,    tk_wstring,   tk_fixed
};
enum DefinitionKind
{
   dk_none,       dk_all,
   dk_Attribute,  dk_Constant,  dk_Exception, dk_Interface,
   dk_Module,     dk_Operation, dk_Typedef,
   dk_Alias,      dk_Struct,    dk_Union,     dk_Enum,
   dk_Primitive,  dk_String,    dk_Sequence,  dk_Array,
   dk_Repository  dk_Wstring
};
enum PrimitiveKind
{
   pk_null,   pk_void,     pk_short,     pk_long,      pk_ushort,
```

```
    pk_ulong,  pk_float,    pk_double,    pk_boolean,   pk_char,
    pk_octet,  pk_any,      pk_TypeCode,  pk_Principal, pk_string,
    pk_objref, pk_longlong, pk_ulonglong, pk_longdouble,
    pk_wchar,  pk_wstring
};
enum AttributeMode {ATTR_NORMAL, ATTR_READONLY};
enum OperationMode {OP_NORMAL, OP_ONEWAY};
enum ParameterMode {PARAM_IN,  PARAM_OUT, PARAM_INOUT};
```

Interfaces

AliasDef Interface
Any Class
ArrayDef Interface
AttributeDef Interface
BOA Class
CORBA Class
ConstantDef Interface
Contained Interface
Container Interface
Context Class
ContextList Class
Current Class
DynamicImplementation Class
EnumDef Interface
Environment Class
Exception Class
ExceptionDef Interface
ExceptionList Class
IDLType Interface
ImplementationDef Interface
ImplRepository Class
InterfaceDef Interface
IRObject Interface
ModuleDef Interface
NamedValue Class
NVList Class
Object Class
OperationDef Interface
ORB Class
Policy Interface
PrimitiveDef Interface
Principal Interface
Repository Interface
Request Class
RequestSeq Class
SequenceDef Interface
ServerRequest Class
StringDef Interface
StructDef Interface
SystemException Class
TypeCode Class

TypedefDef Interface
UnionDef Interface
UnknownUserException Class
UserException Class
WstringDef Interface

AliasDef Interface

The AliasDef interface is used by the Interface Repository to represent an OMG IDL definition that aliases another definition.

File name

somir.idl

Intended usage

An instance of an AliasDef object is used within the Interface Repository to represent an OMG IDL type that aliases another type. The AliasDef is typically used within the Interface Repository to represent a typedef statement as defined in OMG IDL. An instance of an AliasDef object can be created via the create_alias operation of the Container interface.

Local-only

True

Ancestor interfaces

TypedefDef Interface

Exceptions

CORBA::SystemException

IDL syntax

```
module CORBA
   {
      interface AliasDef:TypedefDef
      {
         attribute IDLType orginal_type_def;
      };
   };
```

Supported operations

AliasDef::original_type_def
IDLType::type

AliasDef::original_type_def

The original_type_def read and write operations provide for the access and update of the type being aliased by an OMG IDL alias definition (AliasDef) in the Interface Repository.

Original interface

AliasDef Interface

IDL syntax

```
attribute IDLType original_type_def;
```

Parameters

Read

None.

Write

CORBA::IDLType_ptr original_type_def

This parameter is used to modify the type aliased within the alias definiton. Setting the original_type_def attribute also updates the inherited type attribute.

Return value

Read

CORBA::IDLType_ptr

The returned pointer references an IDLType that represents the type aliased by the AliasDef. The memory is owned by the caller and can be released by invoking CORBA::release.

Write

None.

Exceptions

CORBA::SystemException
CORBA::NO_IMPLEMENT thrown by Write operation on OS/390

Remarks

The original_type_def attribute identifies the type being aliased. Read and write operations are provided with parameter definitions as defined below.

Example

```
// C++
  // assume that 'this_alias' and 'this_struct'
  // have already been initialized
  CORBA::AliasDef * this_alias;
  CORBA::StructDef * this_struct;

  // change 'this_alias' to be an alias for 'this_struct'
  this_alias-> original_type_def (this_struct);

  // obtain the aliased type from the alias definition
  CORBA::IDLType * returned_aliased_type;
  returned_aliased_type = this_alias-> original_type_def ();
```

Any Class

Represents a value having an arbitrary data type.

File name

any.h

Intended usage

The Any class constitutes the C++ mapping of the IDL data type “any”. An Any object can be used by a client or server application to represent application data whose type is not known at compile time. The Any contains both a data structure and a TypeCode that describes the data structure.

The Any class provides a non-default constructor whose parameters are: a CORBA::TypeCode_ptr (describing the type of data held by the Any), a void* pointer (the value to be contained by the Any) and a CORBA::Boolean indicating whether the Any is to assume ownership of the data. (The Any always duplicates the TypeCode rather than assuming ownership of the original.) After the Any assumes ownership of a value, the application should make no assumptions about the continued lifetime of the value. The default value for the Boolean flag is zero (indicating that the Any does not assume ownership of the value). The void* pointer given to the Any non-default constructor can be NULL.

The default constructor creates an Any with a tk_null TypeCode and no value. The copy constructor and assignment operator for Any perform deep copies of both the TypeCode and the value contained by the Any being copied.

The Any class provides insertion (<<=) and extraction (>>=) operators that allow it to hold data of simple IDL types, while maintaining type safety (preventing one from creating an Any whose TypeCode and value do not match). The operators are also convenient because they alleviate the programmer from manipulating TypeCodes; the programmer simply streams data structures into or out of the Any, and the TypeCode is implied by the C++ type of the value being inserted or extracted.

Since the IDL boolean, octet, and char types do not map to distinct C++ types, the Any class introduces helper types for each, to allow each IDL type to have distinct insertion and extration operators. These types (from_boolean, to_boolean, from_octet, to_octet, from_char, and to_char) are shown in the Types section below. To insert a boolean, octet, or char into an Any, or to extract one from an Any, construct a helper object (by passing the data to be inserted/extracted to the helper's constructor), then use the helper object with the corresponding Any insertion/extraction operator.

Similarly, because both bounded and unbounded strings in IDL map to char* in C++, the to_string and from_string helper types are introduced (see below) for inserting/extracting both bounded and unbounded strings to/from an Any. (Unbounded strings are signified by constructing the to_string or from_string helper with a bound of zero.) The nocopy_flag of the from_string helper is used for non-copying insertion of a string into an Any (in which the Any assumes ownership of the string). Unbounded strings can also be inserted into or extracted from an Any without the use of the from_string helper type by using the char* insertion and deletion operators.

The to_object helper type (see below) is used to extract an object reference from an Any as a generic CORBA::Object type. The Any extraction operator corresponding to the to_object helper type widens its contained object reference to CORBA::Object (if it contains one). No duplication of the object reference is performed by the to_object extraction operator.

In addition to the insertion/extraction operators defined in the Any class, corresponding to the basic IDL data types, the emitters generate global insertion/extraction operators for all types defined in IDL. This allows any type that can be defined in IDL to be inserted into or extracted from an Any in a type-safe manner.

In cases where the type-safe Any operators cannot be used, the Any non-default constructor (described above), the replace method, the type method, and the value method can be used to explicitly set or get the TypeCode and value contained by the Any.

Types

```
struct from_boolean
    {
       from_boolean(Boolean b) : val(b) {}
       Boolean val;
    };
struct from_octet
    {
       from_octet(Octet o) : val(o) {}
       Octet val;
    };
struct from_char
    {
       from_char(Char c) : val(c) {}
       Char val;
    };
struct from_string
    {
```

```
        from_string(char *s, ULong b, Boolean nocopy = 0);
        char *val;
        ULong bound;
        Boolean nocopy_flag;
    };
struct to_boolean
    {
        to_boolean(Boolean &b) : ref(b) {}
        Boolean &ref;
    };
struct to_char
    {
        to_char(Char &c) : ref(c) {}
        Char &ref;
    };
struct to_octet
    {
        to_octet(Octet &o) : ref(o) {}
        Octet &ref;
    };
struct to_object
    {
        to_object(Object_ptr &obj) : ref(obj) {}
        Object_ptr &ref;
    };
struct to_string
    {
        to_string(char * &s, ULong b) : val(s), bound(b) {}
        char *&val;
        ULong bound;
    };
```

Supported methods

Any::_nil
Any::operator<<
Any::operator>>
Any::replace
Any::type

Any::_nil

Returns a nil CORBA::Any reference.

Original class

CORBA::Any

IDL syntax

```
static CORBA::Any_ptr _nil ();
```

Return value

CORBA::Any_ptr
A nil Any reference.

Remarks

This method is intended to be used by client and server applications to create a nil Any reference.

Example

```
#include "corba.h"
...
CORBA::Any* any_ptr;
any_ptr = CORBA::Any::_nil();
...
```

Any::operator<<

Inserts data into an Any.

Original class

CORBA::Any

IDL syntax

```
void operator<<= (CORBA::Short data);
void operator<<= (CORBA::UShort data);
void operator<<= (CORBA::Long data);
void operator<<= (CORBA::ULong) data;
void operator<<= (CORBA::Float data;
void operator<<= (CORBA::Double data);
void operator<<= (const CORBA::Any &data);
void operator<<= (const char* data);
void operator<<= (const WChar* data);
void operator<<= (CORBA::Any::from_boolean data);
void operator<<= (CORBA::Any::from_char data);
void operator<<= (CORBA::Any::from_octet data);
void operator<<= (CORBA::Any::from_string data);
```

Parameters

data The data to be inserted into the Any.

Remarks

This operator is intended to be used for type-safe insertion of a data value into an Any. The C++ type of the data being inserted determines what TypeCode is automatically created and stored in the Any. The operators are type-safe in that they insure that an Any is not created with a TypeCode that doesn't match the value it holds.

When inserting a value into an Any, the previous value held by the Any is deallocated.

The from_boolean, from_char, and from_octet helper types are used to distinguish between the IDL types boolean, char, and octet, since these IDL types are mapped to the same C++ type. To insert a boolean, octet, or char into an Any, construct a helper object (by passing the data to be inserted to the helper's constructor), then use the helper object with the corresponding Any insertion operator.

The from_string helper type is used to insert a bounded or unbounded string into an Any, since both IDL types are mapped to char* in C++. (Unbounded strings are signified by constructing the from_string helper with a bound of zero.) The nocopy_flag of the from_string helper is used for non-copying insertion of a string into an Any (in which the Any assumes ownership of the string). Unbounded strings can also be inserted into an Any without the use of the from_string helper type.

In addition to the insertion operators defined in the Any class, corresponding to the basic IDL data types, the emitters generate global insertion operators for all types defined in IDL. This allows any type that can be defined in IDL to be inserted into an Any in a type-safe manner. Both copying and non-copying insertion operators are defined in the bindings.

To insert an array into an Any, the <array-name>_forany helper type (defined in the emitted C++ bindings) should be used. In the C++ bindings, an array within a function argument list decays into a pointer to the first element, thus Any-insertion operators cannot be overloaded to distinguish between arrays of different sizes. Instead, Any-insertion operators are provided for each distinct <array-name>_forany type. To insert an array into an Any, create an appropriate <array-name>_forany object, initializing it from the array to be inserted, then use the global operator<< (the Any insertion operator) defined for that <array-name>_forany type. There is no from_object helper type corresponding to the to_object helper type.

Example

```
#include "corba.h"
#include   ...
/* Assert a value of short type properly insert or
   extract from an Any
 */
CORBA::Any any;
CORBA::Short s1 = 1, s2 = 2;
/* s1 extracted from any */
any <<= s2 /* insert s2 into any */ any>>= s1
assert(s1 == s2);

/* Assert a value of the from/to helper type methods
   which properly inserted or extracted
 */
CORBA::Any anyc;
char my_char = 'z';
CORBA::Char x_char = 'u';
/* insert my_char into anyc */
```

```
anyc <<= corba::any::from_char(my_char);
/* x_char extracted from anyc */
anyc>>= CORBA::Any::to_char(x_char);
assert(x_char == my_char);
```

Any::operator>>

Extracts data from an Any.

Original class

CORBA::Any

IDL syntax

```
CORBA::Boolean operator>>= (CORBA::Short& data) const;
CORBA::Boolean operator>>= (CORBA::UShort& data) const;
CORBA::Boolean operator>>= (CORBA::Long& data) const;
CORBA::Boolean operator>>= (CORBA::ULong& data) const;
CORBA::Boolean operator>>= (CORBA::Float& data) const;
CORBA::Boolean operator>>= (CORBA::Double& data) const;
CORBA::Boolean operator>>= (CORBA::Any& data) const;
CORBA::Boolean operator>>= (char*& data) const;
CORBA::Boolean operator>>= (WChar*& data) const;
CORBA::Boolean operator>>= (CORBA::Any::to_boolean data) const;
CORBA::Boolean operator>>= (CORBA::Any::to_char data) const;
CORBA::Boolean operator>>= (CORBA::Any::to_octet data) const;
CORBA::Boolean operator>>= (CORBA::Any::to_object data) const;
CORBA::Boolean operator>>= (CORBA::Any::to_string data) const;
```

Parameters

data The data whose value is to be extracted from the Any.

Return value

CORBA::Boolean

A non-zero result indicates successful extraction and that the Any actually contained the type of data requested. A zero return value indicates that the Any's TypeCode does not match the C++ type of the operator's parameter and that nothing was extracted. For primitive types, a zero return value indicates that the parameter has not been modified. For non-primitive types, a zero return value indicates that the pointer passed to the operator has been set to NULL.

Remarks

This operator is intended to be used for type-safe extraction of a data value from an Any. If the C++ type of the data being extracted matches the TypeCode in the Any, the operator's parameter is updated with the Any's value. For simple types, the Any's value is copied to the parameter passed to the extraction operator. Non-primitive types are

extracted by pointer; if the extraction is successful, the pointer passed to the extraction operator is modified to point to the Any's value. The Any retains owernship of the value and the caller must not delete it, and should not use the value after the Any is destroyed or given a new value. (For this reason, avoid using Any extraction operators to extract values into <type>_var variables.)

The to_boolean, to_char, and to_octet helper types are used to distinguish between the IDL types boolean, char, and octet, since these IDL types are mapped to the same C++ type. To extract a boolean, octet, or char from an Any, construct a helper object (by passing the data to be inserted to the helper's constructor), then use the helper object with the corresponding Any extraction operator.

The to_string helper type is used to extract a bounded or unbounded string from an Any, since both IDL types are mapped to char* in C++. (Unbounded strings are signified by constructing the from_string helper with a bound of zero.) Unbounded strings can also be extracted from an Any without the use of the from_string helper type.

In addition to the extraction operators defined in the Any class, corresponding to the basic IDL data types, the emitters generate global extraction operators for all types defined in IDL. This allows any type that can be defined in IDL to be extracted from an Any in a type-safe manner.

To extract an array from an Any, the <array-name>_forany helper type (defined in the emitted C++ bindings) should be used. In the C++ bindings, an array within a function argument list decays into a pointer to the first element, thus Any-extraction operators cannot be overloaded to distinguish between arrays of different sizes. Instead, Any-extraction operators are provided for each distinct <array-name>_forany type. To extract an array from an Any, create an appropriate <array-name>_forany object, initializing it from the array to be extraction , then use the global operator>> (the Any extraction operator) defined for that <array-name>_forany type.

After extracting a bounded string or an array from an Any, applications are responsible for checking the Any's TypeCode (using the Any::type() method) to insure they do not overstep the bounds of the array or string when using the extracted value.

The to_object helper type is used to extract an object reference from an Any as a generic CORBA::Object type. The Any extraction operator corresponding to the to_object helper type widens its contained object reference to CORBA::Object (if it contains one). No duplication of the object reference is performed by the to_object extraction operator.

Example

See example in "Any::operator<<" on page 16.

Any::replace

Replaces the data value and TypeCode held by an Any.

Original class

CORBA::Any

IDL syntax

```
void replace (CORBA::TypeCode_ptr tc, void * value,
              CORBA::Boolean release = 0);
```

Parameters

tc The TypeCode describing the value parameter. The Any duplicates this TypeCode, so the caller retained ownership of the input TypeCode.

value The new data to be stored in the Any. The type of this data must be described by the tc parameter. This parameter can be NULL.

If the value is a simple type (char, octet, float, etc.), the *value* parameter should be a pointer to the data. If the value is a string, the *value* parameter should be of type char**. If the value is an object corresponding to an IDL interface (such as MyInterface), the *value* parameter should be of type MyInterface_ptr. If the value is a TypeCode, the *value* parameter should be of type CORBA::TypeCode_ptr. If the value is a constructed IDL type (struct, sequence, union), the value parameter should be a pointer to the data. If the value is an Any, the *value* parameter should be of type CORBA::Any*. If the value is an IDL array, the *value* parameter should be a pointer to the first element of the array.

release

Specifies whether the Any should assume ownership of the value parameter (whether the value should be deallocated when the Any is released). Default is zero (meaning that the Any does not assume ownership of the value).

Remarks

This method is intended to be used to reiniitalize an Any to hold a new TypeCode and data value. This method is not type-safe (no checking is done to insure that the given TypeCode actually matches the given data value.) This method is intended to be used only when the type-safe Any insertion operators cannot be used.

The replace interface mirrors the interface to Any's non-default constructor.

If the Any previously owned the value it contained, that value is deleted and the new value is stored in the Any. The TypeCode previously contained by the Any is released, and the input TypeCode is duplicated and stored in the Any. If the release parameter is nonzero, then the Any assumes ownership of the new value, and the application should make no assumptions about the continued lifetime of the value.

Example

```
#include "corba.h"
...
CORBA::Any any;
```

```
const Val7 = 7;
CORBA::ULong ul_val = Val7;
/* release == > any owns value memory */
CORBA::Boolean any_owns_p = 0;
/* put a Ulong into any */
any. replace(CORBA::_tc_ulong, (void*) &ul_val, any_owns_p);
...
```

Any::type

Accesses the TypeCode contained by an Any.

Original class

CORBA::Any

IDL syntax

```
TypeCode_ptr type() const;
```

Return value

CORBA::TypeCode_ptr
The TypeCode contained by the Any. The caller must subsequently release the TypeCode using CORBA::release(TypeCode_ptr).

Remarks

This method is intended to be used to access the TypeCode associated with an Any (the TypeCode that describes the data held by the Any). The caller must subsequently release the TypeCode using CORBA::release(TypeCode_ptr).

In many cases, Any objects can be used without explicit manipulation of TypeCodes, using the type-safe insertion/extraction operators defined for Any. The type() method is for situations in which the type-safe Any interface is not applicable, or to determine the type of variable needed for extraction.

Example

```
#include "corba.h"
...
CORBA::TypeCode_ptr tcp;
CORBA::Any constAnyVar6;
constAnyVar6 <<= (corba::short)(6);
tcp="constAnyVar6.type();"
...
```

ArrayDef Interface

An ArrayDef represents an OMG IDL array type.

File name

somir.idl

Intended usage

The ArrayDef interface is used by the Interface Repository to represent an OMG IDL array data type. The ArrayDef is not a named Interface Repository data type (it is in a group of interfaces known as Anonymous types). An ArrayDef may be created using the create_array operation of the Repository interface, by specifying the length of the array and a CORBA::IDLType* indicating the array element type.

Since an ArrayDef object only represents a single dimenstion of an array, multi-dimensional IDL arrays are represented by multiple ArrayDef objects, one per array dimension. The element_type_def attribute of the ArrayDef representing the index that is on the farest left side of the array, as defined in IDL, refers to the ArrayDef representing the next index to the right, and so on. The innermost ArrayDef represents the rightmost index and the element type of the multi-dimensional OMG IDL array.

Local-only

True

Ancestor interfaces

IDLType Interface

Exceptions

CORBA::SystemException

IDL syntax

```
module CORBA
   {
      interface ArrayDef:IDLType
      {
         attribute unsigned long length;
         readonlyattribute TypeCode element_type;
         attribute IDLType element_type_def;
      };
   };
```

Supported operations

ArrayDef::element_type
ArrayDef::element_type_def
ArrayDef::length
IDLType::type

ArrayDef::element_type

The element_type operation returns a type (CORBA::TypeCode *) representative of the array element of an ArrayDef.

Original interface

ArrayDef Interface

IDL syntax

```
readonly attribute TypeCode element_type;
```

Parameters

None.

Return value

TypeCode *

The returned value is a pointer to a copy of the CORBA::TypeCode referenced by the element_type attribute. The memory is owned by the caller and can be returned by invoking CORBA::release.

Exceptions

CORBA::SystemException

Remarks

The element_type attribute of an ArrayDef object points to a CORBA::TypeCode that represents the type of the array element. The element_type read operation returns a copy of the CORBA::TypeCode referenced by the element_type attribute.

Example

```
// C++
   // assume that 'this_array' has already been initialized
   CORBA::ArrayDef * this_array;

   // retrieve the TypeCode which represents the array element
   CORBA::TypeCode * array_element_type;
   array_element_type = this_array-> element_type ();
```

ArrayDef::element_type_def

The element_type_def read and write operations allow the access and update of the element type definition of an array definition (ArrayDef) in the Interface Repository.

Original interface

ArrayDef Interface

IDL syntax

```
attribute IDLType element_type_def;
```

Parameters

element_type_def

In Read operation, no input parameters are defined.

In Write operation, CORBA::IDLType_ptr element_type_def. The element_type_def parameter represents the new element definition for the ArrayDef.

Return value

IDLType_ptr

In Read operation, CORBA::IDLType_ptr. The returned object is a pointer to a copy of the IDLType referenced by the element_type_def attribute of the ArrayDef object. The returned object is owned by the caller and can be released by invoking CORBA::release.

In Write operation, no value is returned.

Exceptions

CORBA::SystemException
CORBA::NO_IMPLEMENT thrown by Write operation on OS/390

Remarks

The type of the elements within an array definition is identified by the element_type_def attribute.

Example

```
// C++
   // assume that 'this_array' and 'this_union' have already been initialized
   CORBA::ArrayDef * this_array;
   CORBA::UnionDef * this_union;

   // change the array element type definition to 'this_union'
   this_array-> element_type_def (this_union);

   // read the element type definition from 'this_array'
   CORBA::IDLType * returned_element_type_def;
   returned_element_type_def = this_array-> element_type_def ();
```

ArrayDef::length

The length read and write operations allow the access and update of the length attribute of an array definition (CORBA::ArrayDef) within the Interface Repository.

Original interface

ArrayDef Interface

IDL syntax

```
attribute unsigned long length;
```

Parameters

length In Read operation, no input parameters are defined.

In Write operation, CORBA::ULong length. The length parameter is the new value to which the length attribute of the CORBA::ArrayDef object will be set.

Return value

ULong In Read operation, CORBA::ULong. The returned value is the current value of the length attribute of the array definition (CORBA::ArrayDef) object.

In Write operation, no value is returned.

Exceptions

CORBA::SystemException
CORBA::NO_IMPLEMENT thrown by Write operation on OS/390

Remarks

The length attribute specifies the number of elements in the array. Read and write length operations are supported.

Example

```
// C++
   // assume that 'this_array' has already been initialized
   CORBA::ArrayDef * this_array;

   // change the length attribute of the array definition
   CORBA::ULong new_length = 51;
   this_array-> length (new_length);

   // obtain the length of an array definition
   CORBA::ULong returned_length;
   returned_length = this_array-> length ();
```

AttributeDef Interface

The AttributeDef interface is used within the Interface Repository to represent the information that defines an attribute of an interface.

File name

somir.idl

Intended usage

The AttributeDef object is used to represent the information that defines an attribute of an interface. An AttributeDef may be created by calling the create_attribute operation of the InterfaceDef interface. The create_attribute parameters include the unique RepositoryId (CORBA::RepositoryId), the name (CORBA::Identifier), the version (CORBA::VersionSpec), the type (CORBA::IDLType*) to indicate the type of the attribute, and a parameter to indicate the mode of the attribute (read. read/write, etc.).

Local-only

True

Ancestor interfaces

Contained Interface

Exceptions

CORBA::SystemException

IDL syntax

```
module CORBA
   {
      enum AttributeMode {ATTR_NORMAL, ATTR_READONLY};
      interface AttributeDef:Contained
      {
         readonlyattribute TypeCodetype;
         attribute IDLType type_def;
         attribute AttributeMode mode;
      };
      struct AttributeDescription
      {
         Identifier name;
         RepositoryId id;
         RepositoryId defined_in;
         VersionSpec version;
         TypeCode type;
         AttributeMode mode;
      };
   };
```

Supported operations

AttributeDef::describe
AttributeDef::mode
IDLType::type
AttributeDef::type_def

AttributeDef::describe

The describe operation returns a structure containing information about a CORBA::AttributeDef Interface Repository object.

Original interface

AttributeDef Interface

IDL syntax

```
struct AttributeDescription
   {
      Identifier name;
      RepositoryId id;
      RepositoryId defined_in;
      VersionSpec version;
      TypeCode type;
      AttributeMode mode;
   };
      struct Description
   {
      DefinitionKind kind;
      any value;
   };
   Description describe ();
```

Parameters

None.

Return value

Description *

The returned value is a pointer to a CORBA::Contained::Description structure. The memory is owned by the caller and can be removed by invoking delete.

Exceptions

CORBA::SystemException

Remarks

The inherited describe operation returns a structure (CORBA::Contained::Description) that contains information about a CORBA::AttributeDef Interface Repository object. The

CORBA::Contained::Description structure has two fields: kind (CORBA::DefinitionKind data type), and value (CORBA::Any data type).

The kind of definition described by the returned structure is provided using the kind field, and the value field is a CORBA::Any that contains the description that is specific to the kind of object described. When the describe operation is invoked on an attribute (CORBA::AttributeDef) object, the kind field is equal to CORBA::dk_Attribute and the value field contains the CORBA::AttributeDescription structure.

Example

```
// C++
   // assume that 'this_attribute' has already been initialized
   CORBA::AttributeDef * this_attribute;

   // retrieve a description of the attribute
   CORBA::AttributeDef::Description * returned_description;
   returned_description = this_attribute-> describe ();

   // retrieve the attribute description from the returned description
   // structure
   CORBA::AttributeDescription * attribute_description;
   attribute_description = (CORBA::AttributeDescription *)
     returned_description-> value.value ();
```

AttributeDef::mode

The mode read and write operations allow the access and update of the mode attribute of an attribute definition (CORBA::AttributeDef) within the Interface Repository.

Original interface

AttributeDef Interface

IDL syntax

```
attribute AttributeMode mode;
```

Parameters

mode In Read operation, no input parameters are defined.

In Write operation, CORBA::AttributeMode mode. The mode parameter is the new value to which the mode attribute of the CORBA::AttributeDef object will be set. Valid mode values include CORBA::ATTR_NORMAL (read/write access) and CORBA::ATTR_READONLY (read only access).

Return value

mode In Read operation, CORBA::AttributeMode mode. The returned value is the current value of the mode attribute of the attribute definition (CORBA::AttributeDef) object.

In Write operation, no value is returned.

Exceptions

CORBA::SystemException
CORBA::NO_IMPLEMENT thrown by Write operation on OS/390

Remarks

The mode attribute specifies read only or read/write access for this attribute. Read and write mode operations are supported with parameters as defined below.

Example

```
// C++
   /// assume that 'this_attribute' has already been initialized
   CORBA::AttributeDef * this_attribute;

   // set the new mode in the attribute definition
   CORBA::AttributeMode new_mode = CORBA::ATTR_READONLY;
   this_attribute-> mode (new_mode);

   // retrieve the mode from the attribute definition
   CORBA::AttributeMode returned_mode;
   returned_mode = this_attribute-> mode ();
```

AttributeDef::type_def

The type_def operation returns a pointer to an IDLType that is representative of the type of the attribute defined by the AttributeDef.

Original interface

AttributeDef Interface

IDL syntax

```
attribute IDLType type_def;
```

Parameters

type_def

In Read operation, no input parameters are defined.

In Write operation, CORBA::IDLType_ptr type_def. The type_def input parameter identifies the new setting for the type_def attribute.

Return value

IDLType_ptr

In Read operation, CORBA::IDLType_ptr. The returned CORBA::IDLType * is a pointer to a copy of the information referenced by the type_def attribute. The object and the associated memory are owned by the caller and can be released by invoking CORBA::release.

Exceptions

CORBA::SystemException
CORBA::NO_IMPLEMENT thrown by Write operation on OS/390

Remarks

The type_def attribute within an AttributeDef references an IDLType that identifies the type of attribute. Both read and write type_def operations are supported, the parameters of which are identified below.

Example

```
// C++
   // assume that 'this_attribute' and 'pk_long_def'
   // have already been initialized
   CORBA::AttributeDef * this_attribute;
   CORBA::PrimitiveDef * pk_long_def;

   // set the type_def attribute of the AttributeDef
   // to represent a CORBA::Long
   this_attribute-> type_def (pk_long_def);

   // retrieve the type_def attribute from the AttributeDef
   CORBA::IDLType * attributes_type_def;
   attributes_type_def = this_attribute-> type_def();
```

BOA Class

Provides services for writing server applications.

File name

boa.h

Intended usage

The BOA (Basic Object Adapter) class is intended to be used by application-specific server programs, to access server-side services of the ORB. The BOA class provides methods for activating and deactivating the server and executing remote requests from client applications. The BOA class also provides methods that allow the server application to participate in the exporting and importing of object references and the selection of threads on which remote requests are dispatched. Most of the BOA methods are intended to be called from an application-specific server program. The

BOA::get_principal method, however, is typically called from an implementation of an IDL interface residing in a server, to determine the identity of the remote caller.

Note: Component Broker provides a default server program on all platforms that essentially nullifies the need for direct usage of the BOA interfaces and methods by a customer server application. Refer to the *Component Broker Programming Guide* for further explanation.

Nested classes

CORBA::BOA::DynamicImplementation

Types

```
typedef CORBA::ReferenceData * SOMLINK
                  somdTD_obj_to_refdata (CORBA::Object_ptr obj);
typedef CORBA::Object_ptr SOMLINK
                  somdTD_refdata_to_obj (CORBA::ReferenceData *refdata);
typedef void SOMLINK somdTD_thread_dispatch (CORBA::Request_ptr req);
```

Constants

```
static const CORBA::Flags SOMD_WAIT;
static const CORBA::Flags SOMD_NO_WAIT;
```

Supported methods

BOA::_duplicate
BOA::_nil
BOA::create
BOA::deactivate_impl
BOA::dispose
BOA::execute_next_request
BOA::execute_request_loop
BOA::get_id
BOA::get_principal
BOA::impl_is_ready
BOA::request_pending

Methods introduced by BOA in the CORBA specification but not implemented in this product (if invoked, a CORBA::SystemException is thrown):

change_implementation
deactivate_obj
obj_is_ready

BOA::_duplicate

Duplicates a BOA object.

Original class

CORBA::BOA

IDL syntax

```
static CORBA::BOA_ptr _duplicate (CORBA::BOA_ptr p);
```

Parameters

p The BOA object to be duplicated. The reference can be nil, in which case the return value will also be nil.

Return value

CORBA::BOA_ptr
The new BOA object reference.

Remarks

This method is intended to be used by client and server applications to duplicate a reference to a BOA object.

Example

See example in CORBA::“Object::_duplicate” on page 159.

BOA::_nil

Returns a nil CORBA::BOA reference.

Original class

CORBA::BOA

IDL syntax

```
static CORBA::BOA_ptr _nil ();
```

Return value

CORBA::BOA_ptr
A nil BOA reference.

Remarks

This method is intended to be used by client and server applications to create a nil BOA reference.

Example

See example in CORBA::“Object::_nil” on page 167.

BOA::create

Maps ReferenceData to a local object, and prepares that object for export.

Original class

CORBA::BOA

IDL syntax

```
virtual CORBA::Object_ptr create
  (const CORBA::ReferenceData& refdata,
  CORBA::InterfaceDef_ptr intf,
  CORBA::ImplementationDef_ptr impldef);
```

Parameters

refdata The application-specific ReferenceData of an object residing in a server.

intf The InterfaceDef object, retrieved from the Interface Repository, that describes the interface supported by the object described by the refdata parameter. Currently, this parameter is unused and can be NULL. The caller retains ownership of this object .

impldef
The ImplementationDef of the server in which the call is being made. Currently, this parameter is unused and can be NULL. The caller retains ownership of this object.

Return value

CORBA::Object_ptr
The local object in the server that corresponds to the input ReferenceData, after it has been prepared for export. Ownership of this object reference is transferred to the caller, and should be subsequently released using CORBA::release.

Exceptions

CORBA::SystemException
CORBA::NO_IMPLEMENT (OS/390)

Remarks

Typical server applications need not ever use this method.

This method is part of the CORBA specification.

Example

```
#include "corba.h"
extern CORBA::BOA_ptr srvboa; /* assume previously initialized
                                 using CORBA::ORB::BOA_init */
```

```
...
::CORBA::ReferenceData * rd = (::CORBA::ReferenceData *) NULL;
rd = srvboa->get_id(this);
::CORBA::Object_ptr objPtr =
                srvboa->create(*rd,
                               CORBA::InterfaceDef::_nil(),
                               CORBA::ImplementationDef::_nil());
...
```

BOA::deactivate_impl

Causes a server to stop accepting incoming request messages and informs the somorbd deamon that it is no longer active.

Original class

CORBA::BOA

IDL syntax

```
virtual void deactivate_impl (
                CORBA::ImplementationDef_ptr impldef);
```

Parameters

impldef

The ImplementationDef of the server making the call. This should be the same ImplementationDef originally passed to BOA::impl_is_ready. The caller retains ownership of this parameter.

Exceptions

CORBA::SystemException

Remarks

This method is intended to be used by every server application, to indicate that it no longer wishes to accept incoming request messages from remote clients, prior to server termination (whether normal or abnormal). It should only be invoked if the server has previously successfully called BOA::impl_is_ready.

The method informs the somorbd daemon that the server is no longer active, and that subsequent requests to locate that logical server should cause a new instance of the server to be automatically activated. It also prevents any new request messages from being received by the server; clients issuing such requests will receive SystemExceptions indicating a communications failure. Any requests received by the server prior to calling BOA::deactivate_impl that have not yet been serviced (by a call to BOA::execute_request_loop or BOA::execute_next_request) will be deleted; no response will be sent to the clients that sent them.

This method is part of the CORBA specification.

Example

```
#include "corba.h"
void main(int argc, char* argv[])
{
  /* Initialize the server's ImplementationDef, ORB, and BOA: */
  CORBA::ImplRepository_ptr implrep = new CORBA::ImplRepository;
  /* assume dummyServer is already registered in
     the implementation repository */
  CORBA::ImplementationDef_ptr imp =
            implrep->find_impldef_by_alias ("dummyServer");
  /* Assume op-parm is initialized. For workstation initialize to "DSOM" */
  char * op-parm;
  /* Assume bp-parm is initialized. For workstation initialize to "DSOM_BOA" */
  char * bp-parm;
  static CORBA::ORB_ptr op = CORBA::ORB_init(argc, argv, op-parm);
  static CORBA::BOA_ptr bp = op->BOA_init(argc, argv, bp-parm);
  bp->impl_is_ready(imp);
  ...

  bp->deactivate_impl(imp);
  ...
}
```

BOA::dispose

Destroys an object residing in a server.

Original class

CORBA::BOA

IDL syntax

```
virtual void dispose(CORBA::Object_ptr obj);
```

Parameters

obj The object to be deleted.

Exceptions

CORBA::NO_IMPLEMENT (OS/390)

Remarks

This method can be used to destroy (delete) a local object residing in a server. All outstanding references to the object are henceforth invalid. Outstanding remote references (proxies) to the object are valid only if the server is capable of reactivating to the object. The current implementation of this method simply deletes the input object.

This method is part of the CORBA specification.

Example

```
#include "corba.h"
void main(int argc, char* argv[])
{
  /* Initialize the server's ImplementationDef, ORB, and BOA: */
  CORBA::ImplRepository_ptr implrep = new CORBA::ImplRepository;
  /* Assume dummyServer is already registered in
     the implementation repository */
  CORBA::ImplementationDef_ptr imp =
            implrep->find_impldef_by_alias ("dummyServer");
  extern static CORBA::ORB_ptr op; /* assume previously initialized */
  extern static CORBA::BOA_ptr bp; /* assume previously initialized */
  bp->impl_is_ready(imp);
  ...
  /* Assume that p is a local object pointer already declared
     and defined */
  bp->dispose(p);
  ...
}
```

BOA::execute_next_request

Executes the next pending remote request in a server application.

Original class

CORBA::BOA

IDL syntax

```
virtual CORBA::Status execute_next_request (CORBA::Flags waitFlag);
```

Parameters

waitFlag

Whether the application wants to wait (block), if there is no request currently available to process. Valid values are CORBA::BOA::SOMD_WAIT and CORBA::BOA::SOMD_NO_WAIT.

Return value

CORBA::Status

A zero return value indicates success. If the input parameter is CORBA::BOA::SOMD_NO_WAIT, a return value of SOMDERROR_NoMessages indicates that there is no available request to service.

Exceptions

CORBA::SystemException
CORBA::NO_IMPLEMENT (OS/390)

Remarks

This method is intended to be used by a server application to execute the next remote request received from a remote client, and send the response to the waiting client. Both blocking and non-blocking calls are supported. Requests are executed in first-in-first-out order only. This method should be called only after CORBA::BOA::impl_is_ready has been called successfully.

This method is an IBM extension to the CORBA specification.

Note: On OS/390, the intent of this method is embedded within the CORBA::BOA::impl_is_ready(). So on OS/390 the call to impl_is_ready() does not return control until the server is terminated.

Example

```
#include "corba.h"
void main(int argc, char* argv[])
{
  /* Initialize the server's ImplementationDef, ORB, and BOA: */
  CORBA::ImplRepository_ptr implrep = new CORBA::ImplRepository;
  /* Assume dummyServer is already registered in
     the implementation repository */
  CORBA::ImplementationDef_ptr imp =
            implrep->find_impldef_by_alias ("dummyServer");
  extern static CORBA::ORB_ptr op; /* assume previously initialized */
  extern static CORBA::BOA_ptr bp; /* assume previously initialized */
  bp->impl_is_ready(imp);
  ...
  /* Execute the next pending remote request */
  while(1)
  bp->execute_next_request(CORBA::BOA::SOMD_WAIT);
  ...
}
```

BOA::execute_request_loop

Repeatedly executes remote requests in a server application.

Original class

CORBA::BOA

IDL syntax

```
virtual CORBA::Status execute_request_loop (CORBA::Flags waitFlag);
```

Parameters

waitFlag
Whether the application wants to wait (block), when there are no more requests available to process. Valid values are CORBA::BOA::SOMD_WAIT and CORBA::BOA::SOMD_NO_WAIT.

Return value

CORBA::Status
If the input parameter is CORBA::BOA::SOMD_NO_WAIT, SOMDERROR_NoMessages is returned when there are no more available requests to service. Otherwise, this method never returns to the caller.

Exceptions

CORBA::SystemException

Remarks

This method is intended to be used by a server application to repeatedly execute remote requests as they are received from remote clients, and sends the responses to the waiting clients. Both blocking and non-blocking calls are supported. Requests are executed in first-in-first-out order only, by calling CORBA::BOA::execute_next_request. This method should be called only after CORBA::BOA::impl_is_ready has been called successfully.

This method is an IBM extension to the CORBA specification.

Note: On OS/390, the intent of this method is embedded within the CORBA::BOA::impl_is_ready(). So on OS/390 the call to impl_is_ready() does not return control until the server is terminated.

Example

See example in CORBA::“ORB::BOA_init” on page 173.

BOA::get_id

Returns the ReferenceData associated with a local object in a server.

Original class

CORBA::BOA

IDL syntax

```
virtual CORBA::ReferenceData *get_id (CORBA::Object_ptr obj);
```

Parameters

obj The local object for which ReferenceData is needed. If this parameter is NULL or is a proxy object (rather than a local object in a server), an exception is thrown.

Return value

CORBA::ReferenceData*
The ReferenceData associated with the given object. Ownership of the ReferenceData is transferred to the caller.

Exceptions

CORBA::SystemException
CORBA::NO_IMPLEMENT (OS/390)

Remarks

This method is intended to be used by a server application, to access the ReferenceData used by that server to identify the object.

Typical server applications would only need to call this method to obtain the ReferenceData required by the CORBA::BOA::create method.

This method is part of the CORBA specification.

Example

See example in CORBA::"BOA::create" on page 33.

BOA::get_principal

Returns a Principal object identifying, in a server, the client of a remote request.

Original class

CORBA::BOA

IDL syntax

```
virtual CORBA::Principal_ptr get_principal (
                  CORBA::Object_ptr obj,
                  CORBA::Environment_ptr env);
```

Parameters

obj The target of a remote invocation in a server. Typically, it is the ″this″ pointer in C++ for the method calling CORBA::BOA:get_principal. Currently this parameter is not used and NULL can be passed.

env Currently unused (NULL can be passed).

Return value

CORBA::Principal_ptr
A Principal object identifying the client that initiated the remote request. If CORBA::BOA::get_principal is called outside the context of any remote request, NULL is returned. The caller does not assume ownership of the returned Principal object and should not delete it.

Exceptions

CORBA::SystemException
CORBA::NO_IMPLEMENT (OS/390)

Remarks

This method is intended to be used by implementations of IDL interfaces, residing in a server process, to find the identity of the calling client. This might be used, for example, to implement security authorization checks.

This method is part of the CORBA specification.

Example

```
#include "corba.h"
/* Assume previously initialized using CORBA::ORB::BOA_init () */
extern CORBA::BOA_ptr srvboa;
...
::CORBA::Boolean tmpBoolean;
::CORBA::Principal_ptr prncpl_ptr;
/* Assume the following is called from within an implementation
   of some IDL operation */
prncpl_ptr = srvboa->get_principal(this,
                                   CORBA::Environment::_nil());
tmpBoolean = ::CORBA::is_nil(prncpl_ptr);
/* Error checking */
...
```

BOA::impl_is_ready

Initializes an application as a server, allows it to accept incoming request messages, and registers it with the somorbd daemon.

Original class

CORBA::BOA

IDL syntax

```
virtual void impl_is_ready(CORBA::ImplementationDef_ptr impldef,
                           CORBA::Boolean registration = 1);
```

Parameters

impldef

The ImplementationDef, obtained from the Implementation Repository, that describes the server. The ImplementationDef is typically obtained using the CORBA::ImplRepository find_impldef or find_impldef_by_alias method. On-the-fly servers that are not registered in the Implementation Repository can create ImplementationDef objects using operator new.

registration

The default value (1) indicates that the server should register itself with the somorbd daemon. A zero value indicates that the server should not register itself with the somorbd daemon; this should only be done for lightweight servers of transient objects. This parameter is an IBM extension to the CORBA specification. OS/390 Component Broker does not support this parameter. By default, all servers register themselves with the somorb daemon.

Exceptions

CORBA::SystemException

Remarks

This method is intended to be used by all server applications, to initialize themselves. A server application cannot receive remote requests and cannot export objects (for instance, using CORBA::ORB::object_to_string or CORBA::BOA::create) without first calling CORBA::BOA::impl_is_ready. This method initializes the server's communications resources so that it can accept incoming request messages, and (optionally) registers the server with the somorbd daemon so that client applications can locate it via the daemon.

After a server has called CORBA::BOA::impl_is_ready, it should call CORBA::BOA::deactivate_impl before termination (either normal or abnormal), to inform the somorbd daemon that it is no longer active.

This method is part of the CORBA specification.

Note: On OS/390, after CORBA::BOA::impl_is_ready is invoked, control is given to the ORB runtime to process method requests. Control is not returned to the calling program until the server is terminated.

Example

See example in CORBA::"ORB::BOA_init" on page 173.

BOA::request_pending

Determines whether there are any requests in a server waiting to be serviced.

Original class

CORBA::BOA

IDL syntax

```
virtual CORBA::Boolean request_pending ();
```

Return value

CORBA::Boolean
A zero return value indicates that there are no outstanding requests waiting to be processed. A one return value indicates that there is at least one request pending.

Exceptions

CORBA::NO_IMPLEMENT (OS/390)

Remarks

This method is intended to be used by a server application to determine whether there are any outstanding requests (from remote clients) waiting to be serviced. Hence, this method can be used to determine whether a blocking call to CORBA::BOA::execute_next_request will block.

This call does not modify the queue of waiting requests.

This method is an IBM extension to the CORBA specification.

Example

```
#include "corba.h"
void main(int argc, char* argv[])
{
  /* Initialize the server's ImplementationDef, ORB, and BOA: */
  CORBA::ImplRepository_ptr implrep = new CORBA::ImplRepository;
  /* Assume dummyServer is already registered in the
     implementation repository */
  CORBA::ImplementationDef_ptr imp =
            implrep->find_impldef_by_alias ("dummyServer");
  extern static CORBA::ORB_ptr op; /* assume previously initialized */
  extern static CORBA::BOA_ptr bp; /* assume previously initialized */
  bp->impl_is_ready(imp);
  ...
  /* Determine if there's request waiting */
  CORBA::Boolean retval = bp->request_pending();
  ...
  /* Stop processing requests */
  bp->interrupt_server();
  ...
}
```

ConstantDef Interface

The ConstantDef interface defines a named constant.

File name

somir.idl

Intended usage

The ConstantDef interface is used within the Interface Repository to represent a constant as defined within OMG IDL. An instance of a ConstantDef object defines the data type of the constant, the constant value, and the constant name. A ConstantDef object can be created using the create_constant operation of the Container interface.

Local-only

True

Ancestor interfaces

Contained Interface

Exceptions

CORBA::SystemException

IDL syntax

```
module CORBA
   {
      interface ConstantDef:Contained
         {
            readonlyattribute TypeCode type;
            attribute IDLType type_def;
            attribute anyvalue;
         };
      struct ConstantDescription
         {
            Identifier name;
            RepositoryId id;
            VersionSpec version;
            TypeCode type;
            anyvalue;
         };
   };
```

Supported operations

ConstantDef::describe
IDLType::type
ConstantDef::type_def
ConstantDef::value

ConstantDef::describe

The describe operation returns a structure containing information about a CORBA::ConstantDef Interface Repository object.

Original interface

ConstantDef Interface

IDL syntax

```
struct ConstantDescription
   {
      Identifier name;
      RepositoryId id;
      RepositoryId defined_in;
      VersionSpec version;
      TypeCode type;
      any value;
   };
struct Description
   {
      DefinitionKind kind;
      any value;
   };
Description describe ();
```

Parameters

None.

Return value

Description *

The returned value is a pointer to a CORBA::Contained::Description structure. The memory is owned by the caller and can be removed using delete.

Exceptions

CORBA::SystemException

Remarks

The inherited describe operation returns a structure (CORBA::Contained::Description) that contains information about a CORBA::ConstantDef Interface Repository object. The CORBA::Contained::Description structure has two fields: kind (CORBA::DefinitionKind data type), and value (CORBA::Any data type).

The kind of definition described by the returned structure is provided using the kind field, and the value field is a CORBA::Any that contains the description that is specific to the kind of object described. When the describe operation is invoked on a constant

(CORBA::ConstantDef) object, the kind field is equal to CORBA::dk_Constant and the value field contains the CORBA::ConstantDescription structure.

Example

```
// C++
   // assume that 'this_constant' has already been initialized
   CORBA::ConstantDef * this_constant;

   // retrieve a description of the constant
   CORBA::ConstantDef::Description * returned_description;
   returned_description = this_constant-> describe ();

   // retrieve the constant description from the returned description
   // structure
   CORBA::ConstantDescription * constant_description;
   constant_description = (CORBA::ConstantDescription *)
     returned_description-> value.value ();
```

ConstantDef::type_def

The type_def operation returns a pointer to an IDLType that is representative of the type within a ConstantDef.

Original interface

ConstantDef Interface

IDL syntax

```
attribute IDLType type_def;
```

Parameters

IDLType_ptr

In Read operation, no input parameters are required.

In Write operation, CORBA::IDLType_ptr. The CORBA::IDLType_ptr must reference a simple type (a PrimitiveDef of kind CORBA::pk_long, CORBA::pk_short, CORBA::pk_ulong, CORBA::pk_ushort, CORBA::pk_float, CORBA::pk_double, CORBA::pk_char, CORBA::pk_wchar, CORBA::pk_octet, CORBA::pk_boolean, CORBA::pk_wstring, or CORBA::pk_string).

Return value

IDLType_ptr

In Read operation, CORBA::IDLType_ptr. The returned CORBA::IDLType * is a pointer to a copy of the information referenced by the type_def attribute. The object and the associated memory are owned by the caller and can be released by invoking CORBA::release.

Exceptions

CORBA::SystemException
CORBA::NO_IMPLEMENT thrown by Write operation on OS/390

Remarks

The type_def attribute within a ContstantDef references an IDLType that identifies the definition of the type of the constant. Both read and write type_def operations are supported, the parameters of which are identified below.

Example

```
// C++
   // assume that 'this_constant' and 'pk_long_def'
   // have already been initialized
   CORBA::ConstantDef * this_constant;
   CORBA::PrimitiveDef * pk_long_def;

   // set the type_def attribute of the constant
   // to represent a CORBA::Long
   this_constant-> type_def (pk_long_def);

   // retrieve the type_def attribute from the constant
   CORBA::IDLType * constants_type_def;
   constants_type_def = this_constant-> type_def();
```

ConstantDef::value

The value read and write operations allow the access and update of the value attribute of a ConstantDef.

Original interface

ConstantDef Interface

IDL syntax

```
any value;
```

Parameters

value In Read operation, no input parameters are required.

In Write operation, CORBA::Any & value.The value parameter is a reference to a CORBA::Any data type that provides the constant value to change the value attribute of the ConstantDef. When setting the value attribute, the TypeCode of the supplied CORBA::Any must be equal to the type attribute of the ConstantDef.

Return value

Any * In Read operation, CORBA::Any *. The returned pointer to a CORBA::Any data

type represents the value attribute of the constant. The object memory belongs to the caller, and can be removed by invoking delete.

In Write operation, no value is returned.

Exceptions

CORBA::SystemException
CORBA::NO_IMPLEMENT thrown by Write operation on OS/390

Remarks

The value attribute contains the value of the constant, not the computation of the value (for example, the fact that it was defined as “1+2”). Read and write value operations are provided with parameters as defined below.

Example

```
// C++
   // assume that 'constant_409' has already been initialized
   CORBA::ConstantDef * constant_409;

   // set the value '409' in the Any,
   // and invoke the value operation to update the constant
   CORBA::Any new_value;
   new_value <<= (CORBA::Long) 409;
   constant_409-> value (new_value);

   // read the constant value from the ConstantDef
   CORBA::Any * returned_value;
   returned_value = constant_409-> value ();
```

Contained Interface

The Contained interface is inherited by all Interface Repository interfaces that are contained by other objects. All objects within the Interface Repository, except the root object (Repository) and definitions of anonymous types (ArrayDef, StringDef, and SequenceDef), and primitive types are contained by other objects.

File name

somir.idl

Intended usage

The Contained interface is not itself instantiated as a means of accessing the Interface Repository. As an ancestor to certain Interface Repository objects, it provides a specific list of operations as noted below. Those Interface Repository objects that inherit (directly or indirectly) the operations defined in Contained include: ModuleDef, ConstantDef, StructDef, UnionDef, EnumDef, AliasDef, ExceptionDef, AttributeDef, OperationDef, and InterfaceDef.

Local-only

True

Ancestor interfaces

IRObject Interface

Exceptions

CORBA::SystemException

IDL syntax

```
module CORBA
   {
      typedef string VersionSpec;
      interface Contained:IRObject
      {

         // read/write interface
         attribute RepositoryId id;
         attribute Identifier name;
         attribute VersionSpec version;

         // read interface
         readonly attribute Container defined_in;
         readonly attribute ScopedName absolute_name;
         readonly attribute Repository containing_repository;
         struct Description
            {
               DefinitionKind kind;
               any value;
            };
         Description describe ();

      };
   };
```

Supported operations

Contained::absolute_name
Contained::containing_repository
Contained::defined_in
Contained::describe
Contained::id
Contained::name
Contained::version

Contained::absolute_name

The absolute_name operation retrieves the absolute ScopedName that identifies a Contained object within its enclosing Repository.

Original interface

Contained Interface

IDL syntax

```
readonly attribute ScopedName absolute_name;
```

Parameters

None.

Return value

ScopedName

The returned value is a CORBA::ScopedName data type, the memory of which is owned by the caller. The caller can release this memory by invoking the CORBA::string_free function.

Exceptions

CORBA::SystemException

Remarks

The absolute_name attribute is an absolute ScopedName that identifies a Contained object uniquely within its enclosing Repository. If the Container within which this object is defined is a Repository, the absolute name is formed by concatenating the string ″::″ and this object's name attribute. Otherwise, the absolute_name is formed by concatenating the absolute_name attribute of the object referenced by this object's defined_in attribute, the string ″::″, and this object's name attribute.

A read operation is provided to retrieve the absolute_name value for all Interface Repository objects that have a name attribute.

Example

```
// C++
   // assume the interface_def_ptr has already been initialized
   CORBA::InterfaceDef * interface_def_ptr;

   // the following call returns the absolute name associated with the
   // interface
   CORBA::ScopedName returned_absolute_name;
   returned_absolute_name = interface_def_ptr-> absolute_name();
```

Contained::containing_repository

The containing_repository attribute identifies the Repository that contains this object.

Original interface

Contained Interface

IDL syntax

```
readonly attribute Repository containing_repository;
```

Parameters

None.

Return value

COBRA::Repository_ptr
A pointer to the Repository object is returned.

Exceptions

CORBA::SystemException

Remarks

A Contained object has a defined_in attribute that identifies the Container within which it is contained. The containing_repository attribute identifies the Repository that is eventually reached by recursively following the object's defined_in attribute.

The containing_repository attribute read operation retrieves a pointer to the Repository.

Example

```
// C++
   /// assume that 'interface_1' has already been initialized
   CORBA::InterfaceDef * interface_1;

   // retrieve a pointer to the Repository object
   CORBA::Repository * repository_ptr;
   repository_ptr = interface_1-> containing_repository();
```

Contained::defined_in

The defined_in operation returns the Container object of a Contained object.

Original interface

Contained Interface

IDL syntax

```
readonly attribute Container defined_in;
```

Parameters

None.

Return value

Container *

A pointer to the Container object of the defined_in attribute is returned. The caller owns the memory associated with this object, that can later be released using CORBA::release.

Exceptions

CORBA::SystemException

Remarks

Contained objects have a defined_in attribute that identifies the Container within which they are defined. Objects can be contained either because they are defined within the containing object (for example, an interface is defined within a module) or because they are inherited by the containing object (for example, an operation may be contained by an interface because the interface inherits the operation from another interface). If an object is contained through inheritance, the defined_in attribute identifies the InterfaceDef from which the object is inherited.

The defined_in operation is read-only and returns a pointer to a copy of the Container object identified by the defined_in attribute.

Example

```
// C++
   // assume the interface_def_ptr has already been initialized
   CORBA::InterfaceDef * interface_def_ptr;

   // the following call returns the defined_in Container * for the interface
   CORBA::Container * defined_in_container;
   defined_in_container = interface_def_ptr-> defined_in ();
```

Contained::describe

The describe operation returns a structure containing information about a CORBA::Contained Interface Repository object.

Original interface

Contained Interface

IDL syntax

```
struct Description
   {
      DefinitionKind kind;
      any value;
   };
Description describe ();
```

Parameters

None.

Return value

Description *

The returned value is a pointer to a CORBA::Contained::Description structure. The memory is owned by the caller and can be removed using delete.

Exceptions

CORBA::SystemException

Remarks

The describe operation returns a structure that contains information about an Interface Repository object. The CORBA::Description structure has two fields: kind (CORBA::Contained::DefinitionKind data type), and value (CORBA::Any data type).

The kind of definition described by the returned structure is provided using the kind field, and the value field is a CORBA::Any that contains the description that is specific to the kind of object described. For example, if the describe operation is invoked on an attribute (CORBA::AttributeDef) object, the kind field is equal to CORBA::dk_Attribute and the value field contains the AttributeDescription structure.

The list of Interface Repository object types on which the describe operation may be called includes: modules (CORBA::ModuleDefs), constants (CORBA::ConstantDefs), type definitions (CORBA::TypedefDefs), exceptions (CORBA::ExceptionDefs), attributes (CORBA::AtttributeDefs), operations (CORBA::OperationDefs), and interfaces (CORBA::InterfaceDefs). For further information on the describe operation, please reference the describe operation descriptions for the object types listed above.

CORBA 2.0 specifies that the describe method on named IR objects will return a decription structure that includes the repository ID of the container within which the IR object is defined. However, one common container has no repository ID, that is the Repository itself. In this situation, the IBM implementation returns a pointer to the empty string.

Example

```
// C++
   // assume that 'this_attribute' has already been initialized
   CORBA::AttributeDef * this_attribute;

   // retrieve a description of the attribute
   CORBA::AttributeDef::Description * returned_description;
   returned_description = this_attribute-> describe ();
```

Contained::id

The id operations provide read and write capability for the id attribute of a Contained Interface Repository object.

Original interface

Contained Interface

IDL syntax

```
void id (CORBA::RepositoryId repositoryid)
CORBA::RepositoryId id;
```

Parameters

new_id For Read operation, no input parameters are defined.

For Write operation, CORBA::RepositoryId new_id. The new_id parameter defines the new CORBA::RepositoryId value that will be used to uniquely identify the Contained object in the Interface Repository.

Return value

RepositoryId

For Read operation, CORBA::RepositoryId. The returned CORBA::RepositoryId is a copy of the id attribute of the Contained object. The associated memory is owned by the caller and can be freed by invoking CORBA::string_free.

For Write operation, no value is returned.

Exceptions

CORBA::SystemException
CORBA::NO_IMPLEMENT thrown by Write operation on OS/390

Remarks

An object that is contained by another object has a unique id attribute that identifies it globally within the Interface Repository. The id read (Get) operation provides the ability to retrieve a copy of the id attribute, and the id write (Set) operation allows the unique id attribute to be changed.

Example

```
// C++
   // assume that 'this_union' has already been initialized
   CORBA::UnionDef * this_union;

   // change the 'id' attribute of the union (which is a contained object)
   CORBA::RepositoryId new_repid = CORBA::string_dup ("new_repid_test");
   this_union-> id (new_repid);
   CORBA::string_free (new_repid);

   // query the union to get a copy of the 'id' attribute
   CORBA::RepositoryId returned_rep_id;
   returned_rep_id = this_union-> id ();
```

Contained::name

The name operations are used to read and write the name attribute of an Interface Repository object.

Original interface

Contained Interface

IDL syntax

```
attribute identifier name;
```

Parameters

name For Read operations, no input parameters are required.

For Write operation, CORBA::Identifier name. A name that identifies the new name for the Interface Repository object.

Return value

Identifier

For Read operation, CORBA::Identifier. This operation returns a copy of the name of the object, that is owned by the caller. The caller may later free this memory by invoking CORBA::string_free.

For Write operation, no value is returned.

Exceptions

CORBA::SystemException
CORBA::NO_IMPLEMENT thrown by Write operation on OS/390

Remarks

An object that is contained by another object has a name attribute that identifies it uniquely within the enclosing Container object. Both Read and Write operations are supported, with parameters listed.

Example

```
// C++
   // assume 'interface_1' has already been created
   CORBA::InterfaceDef * interface_1;

   // establish a new name for the interface
   interface_1-> name ("interface_409");

   // retrieve the interface name
   CORBA::Identifier retrieved_name;
   retrieved_name = interface_1-> name ();
```

Contained::version

The version read and write operations allow access and update of the version attribute of an Interface Repository Object.

Original interface

Contained Interface

IDL syntax

```
void version (CORBA::VersionSpec versionspec)
CORBA::VersionSpec version;
```

Parameters

version

For Read operation, no input parameters are required.

For Write operation, CORBA::Versionspec version. The version parameter specifies the new version attribute value for the object.

Return value

VersionSpec

For Read operation, CORBA::VersionSpec. The returned value is owned by the caller. It can be freed using CORBA::string_free.

For Write operation, no value is returned.

Exceptions

CORBA::SystemException
CORBA::NO_IMPLEMENT thrown by Write operation on OS/390

Remarks

The version attribute distinguishes an object from the other versions. Both Read and Write methods are supported, with parameters listed below.

Example

```
// C++
   // assume that this interface has already been initialized.
   CORBA::InterfaceDef * this_interface;
   // change the version of this interface
   this_interface-> version (<<2.0>>);
   // retrieve the version from the interface
   CORBA::VersionSpec returned_version;
   returned_version = this_interface-> version();
```

Container Interface

The Container interface is used to form a containment hierarchy in the Interface Repository.

File name

somir.idl

Intended usage

A Container can contain any number of objects derived from the Contained interface. All Containers, except for Repository, are also derived from Contained. The Container interface is not itself instantiated as a means of accessing the Interface Repository. As an ancestor to certain Interface Repository objects, it provides a specific list of operations as noted below. Those Interface Repository objects that inherit (directly or indirectly) the operations defined in Container include: Repository, ModuleDef, and InterfaceDef.

Local-only

True

Ancestor interfaces

IRObject Interface

Exceptions

CORBA::SystemException

IDL syntax

```
module CORBA
   {
      typedef sequence  ContainedSeq;
      Interface Container:IRObject
         {
            //read interface
            Contained lookup (in ScopedName search_name);
            ContainedSeq contents (in DefinitionKind limit_type
                                   in boolean exclude_inherited);
            ContainedSeq lookup_name (in Identifier search_name,
                                      in long levels_to_search,
                                      in DefinitionKind limit_type,
                                      in boolean exclude_inherited);
            struct Description
               {
                  Contained contained_object;
                  DefinitionKind kind;
                  any value;
               };
            typedef sequence DescriptionSeq;
            DescriptionSeq describe_contents (in DefinitionKind limit_type,
                                              in boolean exclude_inherited,
                                              in long max_returned_objs);
            //write interface
            ModuleDef create_module (in RepositoryId id,
                                     in Identifier name,
                                     in VersionSpec version);
            ConstantDef create_constant (in RepositoryId Id,
                                         in Identifier name,
                                         in VersionSpec version,
                                         in IDLType type,
                                         in any value);
            StructDef create_struct (in RepositoryId id,
                                     in Identifier name,
                                     in VersionSpec version,
                                     In StructMember Seqmembers);
            UnionDef create_union (in RepositoryId id,
                                   in Identifier name,
                                   in VersionSpec version,
                                   in IDLType discriminator_type,
                                   in UnionMemberSeq members);
            EnumDef create_enum (in RepositoryId id,
                                 in Identifier name,
                                 in VersionSpec version,
                                 in EnumMemberSeq members);
            AliasDef create_alias (in Repositoryid id,
                                   in Identifier name,
                                   in VersionSpec version,
                                   in IDLType original_type);
            InterfaceDef create_interface (in RepositoryId id,
```

```
                                                    in Identifier name,
                                                    in VersionSpec version,
                                                    in InterfaceDefSeq base_interfaces);
        };
    };
```

Supported operations

Container::create_alias
Container::create_constant
Container::create_enum
Container::create_exception
Container::create_interface
Container::create_module
Container::create_struct
Container::create_union
Container::contents
Container::describe_contents
“Container::lookup” on page 74

“Container::lookup_name” on page 75

Container::contents

The contents operation returns the list of objects directly contained by or inherited into the object.

Original interface

Container Interface

IDL syntax

```
ContainedSeq contents (in DefinitionKind limit_type
                       in boolean exclude_inherited);
```

Parameters

exclude_inherited

An object can be contained within another object because it is defined within the containing object (for example, an interface is contained within a module). It may also be defined as contained because it is inherited by the containing object (for example, an operation may be contained by an interface because the interface inherits the operation from another interface).

When exclude_inherited is TRUE, inherited objects, if present, are not returned. If exclude_inherited is FALSE, all contained objects (whether contained due to inheritance or because they were defined within the object) are returned.

limit_type

If the limit_type is set to CORBA::dk_all, objects of all interface types are

returned. For example, if this is an InterfaceDef, the attribute, operation, and exception objects are returned. If limit_type is set to a specific interface, only objects of that interface type are returned. For example, only attribute objects are returned if limit_type is set to CORBA::dk_Attribute. The accepted values for limit_type are: CORBA::dk_all, CORBA::Attribute, CORBA::dk_Constant, CORBA::dk_Exception, CORBA::dk_Interface, CORBA::dk_Module, CORBA::dk_Operation, CORBA::dk_Typedef, CORBA::dk_Alias, CORBA::dk_Struct, CORBA::dk_Union, and CORBA::dk_Enum.

Return value

ContainedSeq *

The returned value is a pointer to a ContainedSeq that is the list of Contained objects retrieved from the Interface Repository based upon the input criteria. The memory associated with the ContainedSeq is owned by the caller. The caller can invoke delete on the ContainedSeq * to delete the memory when no longer needed.

Exceptions

CORBA::SystemException

Remarks

The contents operation can be used to navigate through the hierarchy of objects. Starting with the Repository object, a client uses this operation to list all of the objects contained by the Repository, all of the objects contained by the modules within the Repository, all of the interfaces within a specific module, and so on.

Example

```
// C++
   // assume that 'repository_ptr' has already been initialized
   CORBA::Repository * repository_ptr;

   // retrieve all the objects conained by the Repository
   CORBA::ContainedSeq * returned_sequence;
   returned_sequence = repository_ptr-> contents (CORBA::dk_all, 0);
```

Container::create_alias

The create_alias operation creates a new alias definition (AliasDef) in the Interface Repository.

Original interface

Container Interface

IDL syntax

```
AliasDef create_alias (in Repositoryid id,
                       in Identifier name,
                       in VersionSpec version,
                       in IDLType original_type);
```

Parameters

name The name that will be associated with this AliasDef object in the Interface Repository.

original_type
The original_type identifies the original type to which this AliasDef refers. The original_type may be an instance of a SequenceDef, ArrayDef, StringDef, PrimitiveDef, UnionDef, StructDef, AliasDef, EnumDef, or InterfaceDef.

id The id represents the CORBA::RepositoryId that will uniquely identify this AliasDef within the Interface Repository.

version
The version number that will be associated with this AliasDef object in the Interface Repository.

Return value

AliasDef_ptr
A pointer to the created AliasDef object is returned to the caller. The memory associated with this object can later be released by invoking CORBA::release.

Exceptions

CORBA::SystemException

Remarks

The create_alias operation creates a new alias definition in the Interface Repository persistent database, and returns a pointer to a new AliasDef object associated with the alias definition. An AliasDef is typically used by the Interface Repository to represent an OMG IDL 'typedef'.

Example

```
// C++
   // assume the 'repository_ptr' and 'structure_1' objects
   // and these pointers have already been established
   CORBA::Repository * repository_ptr;
   CORBA::StructDef * structure_1;

   // establish the id, name, and version values for the alias definition
   CORBA::RepositoryId rep_id;
   CORBA::Identifier name;
   CORBA::VersionSpec version;
   rep_id = CORBA::string_dup ("unique RepositoryID for this alias");
   name = CORBA::string_dup ("alias_new");
```

```
version = CORBA::string_dup ("1.0");

// create the new alias for 'structure_1' . . .
CORBA::AliasDef * new_alias;
new_alias = repository_ptr-> create_alias (rep_id, name, version,
   structure_1);
```

Container::create_constant

The create_constant operation creates a new ConstantDef object.

Original interface

Container Interface

IDL syntax

```
ConstantDef create_constant (in RepositoryId Id,
                             in Identifier name,
                             in VersionSpec version,
                             in IDLType type,
                             in any value);
```

Parameters

value The value parameter is an CORBA::Any reference. The Any contains the value of the constant.

name The name that is associated with this ConstantDef object in the Interface Repository.

type The type parameter is a CORBA::IDLType * that specifies the type of the ConstantDef. The type should be a CORBA::PrimitiveDef object of a simple type (pk_short, pk_long, pk_ushort, pk_ulong, pk_float, pk_double, pk_boolean, pk_char, pk_wchar, pk_string, pk_wstring, or pk_octet).

id The id represents the CORBA::RepositoryId that will uniquely identify this ConstantDef within the Interface Repository.

version
The version number that will be associated with this ConstantDef object in the Interface Repository.

Return value

ConstantDef_ptr
A pointer to the created ConstantDef object is returned to the caller. The memory associated with this object can later be released by invoking CORBA::release.

Exceptions

CORBA::SystemException

Remarks

The create_constant operation creates a new ConstantDef object with the specified type and value. A representation of the new ConstantDef object is created in the Interface Repository persistent database and a pointer to the memory representation of the ContstantDef object is returned to the caller.

Example

```
// C++
   // repository_ptr and module_one has already been initialized . . .
   CORBA::Repository * repository_ptr;
   CORBA::ModuleDef * module_one;

   CORBA::RepositoryId constants_rep_id;
   CORBA::Identifier constants_name;
   CORBA::VersionSpec version;
   CORBA::ConstantDef * constant_def_one;
   CORBA::Any constants_value;
   CORBA::PrimitiveDef * primitive_long;

   // establish the id, name, and version values for the constant
   constants_rep_id = CORBA::string_dup ("unique RepositoryID for my
      constant");
   constants_name = CORBA::string_dup ("constant_of_2001");
   version = CORBA::string_dup ("1.0");

   // establish the Any with a 'value' of 2001
   constants_value <<= (CORBA::Long) 2001;

   // create a PrimitiveDef that represents a CORBA::Long data type
   primitive_long = repository_ptr-> get_primitive (CORBA::pk_long);

   // create the new constant that will be contained in module_one
   constant_def_one = module_one-> create_constant (constants_rep_id,
   constants_name, version, primitive_long, constants_value);
```

Container::create_enum

The create_enum operation creates a new enumeration definition (EnumDef) in the Interface Repository.

Original interface

Container Interface

IDL syntax

```
EnumDef create_enum (in RepositoryId id,
                     in Identifier name,
                     in VersionSpec version,
                     in EnumMemberSeq members);
```

Parameters

members
This is a reference to a CORBA::EnumMemberSeq that provides the list of the elements will comprise the new enumeration (EnumDef). The length of the CORBA::EnumMemberSeq must be greater than zero. The CORBA::EnumMemberSeq contains a distinct name for each possible value of the enumeration.

name The name that will be associated with this EnumDef object in the Interface Repository.

id The id represents the CORBA::RepositoryId that will uniquely identify this EnumDef within the Interface Repository.

version
The version number that will be associated with this EnumDef object in the Interface Repository.

Return value

EnumDef_ptr
A pointer to the created EnumDef object is returned to the caller. The memory associated with this object can later be released by invoking CORBA::release.

Exceptions

CORBA::SystemException

Remarks

The create_enum operation creates a new enumeration definition in the Interface Repository persistent database, and returns a pointer to a new EnumDef object associated with the enumeration definition.

Example

```
// C++
   // assume that 'repository_ptr' had already been intitialized . . .
   CORBA::Repository * repository_ptr;

   // establish the id, name, and version values for the enumeration
   CORBA::RepositoryId rep_id;
   CORBA::Identifier name;
   CORBA::VersionSpec version;
   rep_id = CORBA::string_dup ("unique RepositoryID for my enumeration");
   name = CORBA::string_dup ("enumeration new");
   version = CORBA::string_dup ("1.0");
```

```
/// instantiate an EnumMemberSeq and set the length to 2
CORBA::EnumMemberSeq enum_members;
enum_members.length(2);

// establish the EnumMemberSeq to represent an enumeration with two
// elements
enum_members[0] = (char *) CORBA::string_dup ("enum_value_0");
enum_members[1] = (char *) CORBA::string_dup ("enum_value_1");

// create the new enumeration . . .
CORBA::EnumDef * new_enum;
new_enum = repository_ptr-> create_enum (rep_id, name, version,
   enum_members);
```

Container::create_exception

The create_exception operation returns a new exception definition (CORBA::ExceptionDef) contained in the CORBA::Container on which it is invoked.

Original interface

Container Interface

IDL syntax

```
ExceptionDef * create_exception(RepositoryId id
                                Indentifier name
                                VersionSpec version
                                StructMemberSeq & members);
```

Parameters

name The name that will be associated with this CORBA::ExceptionDef object in the Interface Repository.

id The id represents the CORBA::RepositoryId that will uniquely identify this CORBA::ExceptionDef within the Container.

members
The members parameter defines the members of the exception definition. Each element of the members parameter has 3 fields. The name field is the name of the element. The type field (a reference to a CORBA::TypeCode *) is not used for create_exception and should be set to CORBA::_tc_void. The type_def field references a CORBA::IDLType * that defines the type definition of the member element.

The members parameter can have a length of zero to indicate that the exception definition has no members.

version
The version number that will be associated with this CORBA::ExceptionDef object in the Interface Repository.

Return value

ExceptionDef *

The return value is a pointer to the newly created CORBA::ExceptionDef. The memory is owned by the caller and can be released using CORBA::release.

Exceptions

CORBA::SystemException

Remarks

The create_exception operation returns a new CORBA::ExceptionDef contained in the CORBA::Container on which it is invoked. A representation of the new CORBA::ExceptionDef object is created in the Interface Repository persistent database and a pointer to the memory representation of the CORBA::ExceptionDef object is returned to the caller.

The id, name, version, and members attributes are set as specified. The type attribute is also set.

An error is returned if an object with the specified id already exists within the Interface Repository, or if an object with the specified name already exists within the CORBA::Container on which the create_exception is invoked.

Example

```
// C++
   // assume that 'this_module' and 'this_struct'
   // have already been initialized
   CORBA::ModuleDef * this_module;
   CORBA::StructDef * this_struct;

   // establish the 'create_exception' rep_id, name, and version parameters
   CORBA::RepositoryId rep_id = "UniqueRepositoryId";
   CORBA::Identifier name = "this_exception";
   CORBA::VersionSpec version = "1.0";

   // establish and initialize a CORBA::StructMemberSeq
   // with which to create the CORBA::ExceptionDef
   CORBA::StructMemberSeq members_list;
   members_list length (1);
   members_list[0].name = CORBA::string_dup ("exception_0");
   members_list[0].type = CORBA::_tc_void;
   members_list[0].type_def = this_struct;

   // create the exception definition
   this_module-> create_exception ( rep_id, name, version, members_list);
   delete (members_list);
   //cleanup.
```

Container::create_interface

The create_interface operation is used to create a new interface definition (InterfaceDef) within the Interface Repository.

Original interface

Container Interface

IDL syntax

```
InterfaceDef create_interface (in RepositoryId id,
                               in Identifier name,
                               in VersionSpec version,
                               in InterfaceDefSeq base_interfaces);
```

Parameters

name The name that will be associated with this InterfaceDef object in the Interface Repository.

base_interface
The base_interfaces parameter lists all of the interfaces from which this interface inherits.

id The id represents the CORBA::RepositoryId that will uniquely identify this InterfaceDef within the Interface Repository.

version
The version number that will be associated with this InterfaceDef object in the Interface Repository.

Return value

InterfaceDef_ptr
A pointer to the created InterfaceDef object is returned to the caller. The memory associated with this object can later be released by invoking CORBA::release.

Exceptions

CORBA::SystemException

Remarks

The create_interface operation returns a new empty InterfaceDef with the specified base_interfaces. Type, exception, and constant definitions can be added using the Container::create_, Container::create_exception, and Container::create_constant operations respectively on the new InterfaceDef. OperationDefs can be added using InterfaceDef::create_operation and AttributeDefs can be added using Interfacedef::create_attribute. Definitions can also be added using the Contained::move operation.

Example

```
// C++
   // assume that 'module_1', 'interface_A', and 'interface_B'
   // have already been initialized
   CORBA::ModuleDef * module_1;
   CORBA::InterfaceDef * interface_A;
   CORBA::InterfaceDef * interface_B;

   // establish the id, name, and version values for the interface
   CORBA::RepositoryId rep_id;
   CORBA::Identifier name;
   CORBA::VersionSpec version;
   rep_id = CORBA::string_dup ("unique RepositoryID for my interface");
   name = CORBA::string_dup ("interface_C");
   version = CORBA::string_dup ("1.0");

   // establish the base interfaces from which the new interface will
   // inherit
   CORBA::InterfaceDefSeq base_interfaces;
   base_interfaces.length(2);
   base_interfaces[0] = CORBA::InterfaceDef::_duplicate (interface_A);
   base_interfaces[1] = CORBA::InterfaceDef::_duplicate (interface_B);

   // create a new interface 'interface_C' . . .
   CORBA::InterfaceDef * interface_C;
   interface_C = module_1-> create_interface (rep_id, name, version,
      base_interfaces);
```

Container::create_module

The create_module operation creates a new module definition (ModuleDef) in the Interface Repository.

Original interface

Container Interface

IDL syntax

```
ModuleDef create_module (in RepositoryId id,
                         in Identifier name,
                         in VersionSpec version);
```

Parameters

name The name that will be associated with this ModuleDef object in the Interface Repository.

id The id represents the CORBA::RepositoryId that will uniquely identify this ModuleDef within the Interface Repository.

version
The version number that will be associated with this ModuleDef object in the Interface Repository.

Return value

ModuleDef_ptr
A pointer to the created ModuleDef object is returned to the caller. The memory associated with this object can later be released by invoking CORBA::release.

Exceptions

CORBA::SystemException

Remarks

The create_module operation returns a new empty ModuleDef object after it creates a corresponding new module representation in the Interface Repository persistent database. Definitions can be added using Container::create_ operations on the new module, or by using the Contained::move operation.

Example

```
// C++
   // repository_ptr has already been initialized . . .
   CORBA::Repository * repository_ptr;

   CORBA::ModuleDef * new_module;
   CORBA::RepositoryId modules_rep_id;
   CORBA::Identifier modules_name;
   CORBA::VersionSpec version;

   // establish the id, name, and version values for the module
   modules_rep_id = CORBA::string_dup ("unique RepositoryID for my module");
   modules_name = CORBA::string_dup ("module new");
   version = CORBA::string_dup ("1.0");

   // create the new module
   new_module = repository_ptr-> create_module (modules_rep_id, modules_name
```

Container::create_struct

The create_struct operation creates a new structure definition (StructDef) in the Interface Repository.

Original interface

Container Interface

IDL syntax

```
StructDef create_struct (in RepositoryId id,
                         in Identifier name,
                         in VersionSpec version,
                         in StructMemberSeq members);
```

Parameters

name The name that will be associated with this StructDef object in the Interface Repository.

id The id represents the CORBA::RepositoryId that will uniquely identify this StructDef within the Interface Repository.

members
This is a reference to a CORBA::StructMemberSeq that provides the list of the elements will comprise the new structure (StructDef). The length of the CORBA::StructMemberSeq must be greater than zero.

Each CORBA::StructMember within the CORBA::StructMemberSeq has 3 fields. The name field identifies the name of the StructMember. The type field of the StructMember is ignored by create_struct, and should be set to CORBA::_tc_void. The type_def field is a CORBA::IDLType * that represents the type definition of the StructMember.

version
The version number that will be associated with this StructDef object in the Interface Repository.

Return value

StructDef_ptr
A pointer to the created StructDef object is returned to the caller. The memory associated with this object can later be released by invoking CORBA::release.

Exceptions

CORBA::SystemException

Remarks

The create_struct operation creates a new structure definition in the Interface Repository persistent database, and returns a pointer to a new StructDef object associated with the struct definition.

Example

```
// C++
   // assume 'primitive_long', 'primitive_double',
   // and 'repository_ptr' have already been instantiated
   CORBA::PrimitiveDef * primitive_double;
   CORBA::PrimitiveDef * primitive_long;
   CORBA::Repository * repository_ptr;
```

```
// establish the id, name, and version values for the structure
CORBA::RepositoryId rep_id;
CORBA::Identifier name;
CORBA::VersionSpec version;
rep_id = CORBA::string_dup ("unique RepositoryId for my structure");
name = CORBA::string_dup ("structure_new");
version = CORBA::string_dup ("1.0");

/// instantiate a StructMemberSeq and set the length to 2
CORBA::StructMemberSeq struct_members;
struct_members.length(2);

// establish the StructMemberSeq to represent a structure with two
// elements: a CORBA::Double called 'x', and a CORBA::Long called 'y'.
struct_members[0].name = CORBA::string_dup ("x");
struct_members[0].type_def =
   CORBA::IDLType::_duplicate (primitive_double);

struct_members[1].name = CORBA::string_dup ("y");
struct_members[1].type_def =
   CORBA::IDLType::_duplicate (primitive_long);

// create the new structure . . .
CORBA::StructDef * new_struct;
new_struct = repository_ptr-> create_struct (rep_id, name, version,
   struct_members);
```

Container::create_union

The create_union operation creates a new union definition (UnionDef) in the Interface Repository.

Original interface

Container Interface

IDL syntax

```
UnionDef create_union (in RepositoryId id,
                       in Identifier name,
                       in VersionSpec version,
                       in IDLType discriminator_type,
                       in UnionMemberSeq members);
```

Parameters

name The name that will be associated with this UnionDef object in the Interface Repository.

discriminator_type
This is a CORBA::IDLType * that identifies the union's discriminator type. The

discriminator type can be a PrimitiveDef (with a primitive kind of CORBA::pk_long, CORBA::pk_short, CORBA::pk_ulong, CORBA::pk_ushort, CORBA::pk_char, CORBA::pk_wchar, or CORBA::pk_boolean) or an EnumDef (which represents an enumerator definition).

id
The id represents the CORBA::RepositoryId that will uniquely identify this UnionDef within the Interface Repository.

members
This is a reference to a CORBA::UnionMemberSeq that provides the list of the elements that will comprise the new union (UnionDef). The length of the CORBA::UnionMemberSeq must be greater than zero.

Each CORBA::UnionMember within the CORBA::UnionMemberSeq has 4 fields. The name field identifies the name of the UnionMember. The type field of the UnionMember is ignored by create_union, and should be set to CORBA::_tc_void. The label field of each UnionMember is a CORBA::Any data type that represents a distinct value of the discriminator_type (a label of type CORBA::Octet and value 0 indicates the default union member). The type_def field is a CORBA::IDLType * that represents the type definition of the UnionMember.

version
The version number that will be associated with this UnionDef object in the Interface Repository.

Return value

UnionDef_ptr
A pointer to the created UnionDef object is returned to the caller. The memory associated with this object can later be released by invoking CORBA::release.

Exceptions

CORBA::SystemException

Remarks

The create_union operation creates a new union definition in the Interface Repository persistent database, and returns a pointer to a new UnionDef object associated with the union definition.

Example

```
// C++
   // assume 'primitive_long', 'primitive_double'
   // 'structure_1', and 'repository_ptr' have already been instantiated
   CORBA::PrimitiveDef * primitive_long;
   CORBA::PrimitiveDef * primitive_double;
   CORBA::StructDef * structure_1;
   CORBA::Repository * repository_ptr;

   // establish the id, name, and version values for the union
   CORBA::RepositoryId rep_id;
```

```
CORBA::Identifier name;
CORBA::VersionSpec version;
rep_id = CORBA::string_dup ("unique RepositoryId for my union");
name = CORBA::string_dup ("union new");
version = CORBA::string_dup ("1.0");

// the discriminator type is in this case CORBA::Long
CORBA::IDLType * discriminator_type;
discriminator_type = primitive_long;

// instantiate a UnionMemberSeq and set the length to 2
CORBA::UnionMemberSeq union_members;
union_members.length(2);

// establish the UnionMemberSeq to represent a union with two
// elements: a CORBA::Double called 'x', and a previously created
// structure called 'y'.
union_members[0].name = CORBA::string_dup ("x");
union_members[0].type_def =
   CORBA::IDLType::_duplicate (primitive_double);
union_members[0].label <<= (CORBA::Long) 1;

union_members[1].name = CORBA::string_dup ("y");
union_members[1].type_def = CORBA::IDLType::_duplicate (structure_1);
union_members[1].label <<= (CORBA::Long) 2;

// create the new union . . .
CORBA::UnionDef * new_union;
new_union = repository_ptr-> create_union (rep_id, name, version,
   discriminator_type, union_members);
```

Container::describe_contents

The describe_contents operation combines the contents operation and the describe operation.

Original interface

Container Interface

IDL syntax

```
struct Description
   {
      Contained contained_object;
      DefinitionKind kind;
      any value;
   };

DescriptionSeq describe_contents (in DefinitionKind limit_type,
                                  in boolean exclude_inherited,
                                  sin long max_returned_objs);
```

Parameters

exclude_inherited

CORBA::Contained objects have a defined_in attribute that identifies the CORBA::Container within which they are defined. Objects can be contained either because they are defined within the containing object (for example, an interface is defined within a module) or because they are inherited by the containing object (for example, an operation may be contained by an interface because the interface inherits the operation from another interface).

If the exclude_inherited parameter is set to TRUE, inherited objects (if there are any) are not returned. If exclude_inherited is set to FALSE, all contained objects are returned (whether contained due to inheritance or because they were defined within the object).

limit_type

The limit_type identifies the interface types for which descriptions are returned. If limit_type is set to CORBA::dk_all, objects of all interface types within the CORBA::Container are returned. If limit_type is set to a specific interface, only objects of that interface are returned.

Valid values for limit_type include CORBA::dk_all, CORBA::dk_Attribute, CORBA::dk_Constant, CORBA::dk_Exception, CORBA::dk_Interface, CORBA::dk_Module, CORBA::dk_Operation, CORBA::dk_Typedef, CORBA::dk_Union, CORBA::dk_Alias, CORBA::dk_Struct, and CORBA::dk_Enum.

max_returned_objs

The max_returned_objs parameter limits the number of objects that can be returned in an invocation of the call to the number provided. Setting the parameter to -1 indicates that all contained objects should be returned.

Return value

DescriptionSeq *

The returned value is a pointer to a sequence of descriptions of Interface Repository objects. The memory is owned by the caller and can be removed using delete.

Exceptions

CORBA::SystemException

Remarks

The describe_contents operation combines the contents operation and the describe operation. For each object returned by the contents operation, the description of the object is returned (i.e., the object's describe operation is invoked and the results returned).

Example

```
// C++
   // assume that 'this_module' has already been initialized
   CORBA::ModuleDef * this_module;

   // retrieve information about all objects contained
   // within the module using 'describe_contents'
   CORBA::Container::DescriptionSeq * returned_descriptions;
   returned_descriptions =
      this_module-> describe_contents (CORBA::dk_all, 1, -1);
```

Container::lookup

The lookup operation locates a definition relative to this container given a scoped name using OMG IDL's name scoping rules.

Original interface

Container Interface

IDL syntax

```
Contained lookup (in ScopedName search_name);
```

Parameters

search_name
The search_name is the scoped name of the object using OMG IDL's name scoping rules. This name is used as the search criteria for locating the object within the Interface Repository.

Return value

Contained_ptr
The return value is a pointer to a CORBA::Contained object resulting from the search. If the search_name was not located within the Interface Repository, a nil object is returned. If a non nil CORBA::Contained object pointer is returned, the memory associated with the object is owned by the caller and can be released by invoking CORBA::release.

Exceptions

CORBA::SystemException

Remarks

The lookup operation locates a definition relative to this container given a scoped name using OMG IDL's name scoping rules. An absolute scoped name (beginning with ″::″) locates the definition relative to the enclosing Repository. If no object is found, a nil object reference is returned.

Example

```
// C++
   // assume that 'module_1' has already been initialized
   CORBA::ModuleDef * module_1;

   // use the scoped name to lookup an operation . . .
   CORBA::Contained * ret_contained;
   ret_contained = module_1-> lookup ("Module2::Interface6::Operation7");;
```

Container::lookup_name

The lookup_name operation is used to locate an object by name within a particular object or within the objects contained by that object.

Original interface

Container Interface

IDL syntax

```
ContainedSeq lookup_name (in Identifier search_name,
                          in long levels_to_search,
                          in DefinitionKind limit_type,
                          in boolean exclude_inherited);
```

Parameters

exclude_inherited

An object can be contained within another object because it is defined within the containing object (for example, an interface is contained within a module). It may also be defined as contained because it is inherited by the containing object (for example, an operation may be contained by an interface because the interface inherits the operation from another interface).

When exclude_inherited is TRUE, inherited objects, if present, are not returned. If exclude_inherited is FALSE, all contained objects (whether contained due to inheritance or because they were defined within the object) are returned.

limit_type

The limit_type parameter indicates which type of object should be examined while performing the name search. The accepted values for limit_type are: CORBA::dk_all, CORBA::dk_Attribute, CORBA::dk_Constant, CORBA::dk_Exception, CORBA::dk_Interface, CORBA::dk_Module, CORBA::dk_Operation, CORBA::dk_Typedef, CORBA::dk_Alias, CORBA::dk_Struct, CORBA::dk_Union, and CORBA::dk_Enum. If the limit_type is CORBA::dk_all, objects of all interface types are returned (for example, attributes, operations, and exceptions are all returned). If limit_type is set to a specific interface, only objects of that interface are returned.

search_name
The search_name specifies the name for which the search is to be made.

levels_to_search
The levels_to_search parameter controls the the depth of the search. When levels_to_search is set to -1 the current object is searched as well as all contained objects. Setting levels_to_search to 1 will limit the search to the current object only.

Return value

ContainedSeq *
The returned value is a pointer to a ContainedSeq that is the list of Contained objects retrieved from the Interface Repository based upon the input criteria. The memory associated with the ContainedSeq is owned by the caller. The caller can invoke delete on the ContainedSeq * to delete the memory when no longer needed.

Exceptions

CORBA::SystemException

Remarks

The lookup_name operation is used to locate an object by name within a particular object or within the objects contained by that object. The parameters to the lookup_name operation specify the name for the search, the number of levels to search, the type of objects to be examined in the search, and whether containment by inheritance should be included.

Example

```
// C++
   // assume 'repository_ptr' is already initialized
   CORBA::Repository * repository_ptr;

   // search for a specific interface name
   CORBA::ContainedSeq * cont_seq;
   cont_seq = repository_ptr-> lookup_name ("Interface1", -1,
      CORBA::dk_Interface, 0);
```

Context Class

Contains a list of properties that represent information about the client environment.

File name

context.h

Intended usage

The Context class is used to pass information from the client environment to the server environment, specifically information that is inconvenient to pass as method parameters. The information is specified as a list of properties. Each property consists of a name and a string value associated with that name. An IDL operation specification may contain a clause specifying context properties that should be passed to the server when the method is invoked. The Context object associated with an operation is passed as a distinguished parameter. The ORB::get_default_context method is called to get a reference to the default process context. The Context class provides methods to add and delete properties, as well as query information about a context.

Contexts may be “chained” together to achieve a particular default behavior. Property searches on a child context recursively look in the parent context. Contexts may optionally be named. A context name can be used to specify a starting search scope.

Supported methods

Context::_duplicate
Context::_nil
Context::context_name
Context::create_child
Context::delete_values
Context::get_values
Context::parent
Context::set_one_value
Context::set_values

Context::_duplicate

Duplicates a Context object.

Original class

CORBA::Context

IDL syntax

```
static CORBA::Context_ptr _duplicate (CORBA::Context_ptr p);
```

Parameters

p The Context object to be duplicated. The reference can be nil, in which case the return value will also be nil.

Return value

CORBA::Context_ptr
The new Context object reference. This value should subsequently be released using CORBA::release(Context_ptr).

Remarks

This method is intended to be used by client and server applications to duplicate a reference to a Context object. Both the original and the duplicate reference should subsequently be released using CORBA::release(Context_ptr).

Context::_nil

Returns a nil CORBA::Context reference.

Original class

CORBA::Context

IDL syntax

```
static CORBA::Context_ptr _nil ();
```

Return value

CORBA::Context_ptr
A nil Context reference.

Remarks

This method is intended to be used by client and server applications to create a nil Context reference.

Context::context_name

Retrieves the name of a context.

Original class

CORBA::Context

IDL syntax

```
const char * context_name() const;
```

Return value

const char *
A pointer to the name of the context, if any, or a null pointer. Ownership of the return value is maintained by the Context; the return value must not be freed by the caller.

Remarks

Context objects may optionally be named. A context name can be used to specify a starting search scope. The context_name method retrieves the name of a context.

Context::create_child

Creates a child Context object.

Original class

CORBA::Context

IDL syntax

```
CORBA::Status create_child(const char *ctx_name,
                           CORBA::Context_ptr &child_ctx);
```

Parameters

ctx_name
The name of the child context to be created, if any, or a null pointer. If specified, the input context name should follow the rules for OMG IDL identifiers. The caller retains ownership of the input name (the Context makes its own copy).

child_ctx
A pointer for a Context object, passed by reference, to be updated by the create_child method to point to the newly created child context. Ownership of this parameter transfers to the caller and must be freed by calling CORBA::release(Context_ptr).

Return value

CORBA::Status
A zero return code indicates the child Context object was successfully created.

Exceptions

CORBA::SystemException

Remarks

Context objects may be ″chained″ together to achieve a particular default behavior. Property searches on a child context recursively look in the parent context. The create_child method creates a new child Context object. The child context is chained to the parent context, which is the target object.

Context::delete_values

Deletes one or more properties.

Original class

CORBA::Context

IDL syntax

```
CORBA::Status delete_values(const char *prop_name);
```

Parameters

prop_name
An identifier specifying the properties to be deleted. To specify multiple properties, pass an identifier with a trailing wildcard character. If a null pointer is passed for this parameter, a system exception is raised.

Return value

CORBA::Status
A zero return code indicates the properties were successfully deleted. If the input property name is not found, a system exception is raised.

Exceptions

CORBA::SystemException

Remarks

The delete_values method deletes the specified properties from a Context object. If the property name has a trailing wildcard character (″*″), then all property names that match are deleted. Search scope is always limited to the specified context.

Context::get_values

Retrieves one or more property values.

Original class

CORBA::Context

IDL syntax

```
CORBA::Status get_value(const char *start_scope,
                        CORBA::Flags op_flags,
                        const char *prop_name,
                        CORBA::NVList_ptr &values);
```

Parameters

start_scope
An identifier specifying the name of the context at which the search should begin. If the search scope is not specified, the search begins with the target Context object. If the specified search scope is not found, a system exception is raised.

op_flags
CORBA::CTX_RESTRICT_SCOPE may be used to indicate searching is limited to the specified search scope. By default, the entire context tree is searched.

prop_name
An identifier specifying the name of the properties to return. To specify multiple properties, pass an identifier with a trailing wildcard character.

values The pointer for an NVList object, passed by reference, to be updated by the get_values method to point to the resulting NVList. Ownership of the returned object transfers to the caller. Memory should be freed by calling CORBA::release(NVList_ptr).The pointer for an NVList object, passed by reference, to be updated by the get_values method to point to the resulting NVList. Ownership of the returned object transfers to the caller. Memory should be freed by calling CORBA::release(NVList_ptr).

Return value

CORBA::Status
A zero return code indicates that one or more property values were successfully returned. If no matching property name is found, -1 is returned.

Exceptions

CORBA::SystemException

Remarks

The get_values method retrieves the specified context property values. If the property name has a trailing wildcard character (″*″), then all matching properties and their values are returned. If a matching property is not found at the starting scope, the search optionally continues up the context tree until a match is found or all contexts in the chain have been exhausted.

Context::parent

Retrieves the parent context.

Original class

CORBA::Context

IDL syntax

```
CORBA::Context_ptr parent() const;
```

Return value

Context_ptr
A pointer to the parent context, if any, or a null pointer. Ownership of the return value is maintained by the Context; the return value must not be freed by the caller.

Remarks

Context objects may be ″chained″ together to achieve a particular defaulting behavior. Property searches on a child ontext recursively look in the parent context. The parent method retrieves the parent of the target Context object.

Context::set_one_value

Adds a single property.

Original class

CORBA::Context

IDL syntax

```
CORBA::Status set_one_value(const char *prop_name,
                            const CORBA::Any &value);
```

Parameters

prop_name
The name of the property to be added. Context properties follow the rules for OMG IDL identifiers. Property names should not end with an asterisk. If a null pointer is passed for this parameter, a system exception is raised. The caller retains ownership of the input string (the Context makes its own copy).

value The address of the value of the property to be added. Currently, only strings are supported as property values. It is legal to pass a null pointer. The caller retains ownership of the input Any.

Return value

CORBA::Status
A zero return code indicates the property was successfully added.

Exceptions

CORBA::SystemException

Remarks

The set_one_value method adds a single property to a context. If the input property name is not found in the property list, a new NamedValue (with the input property name and value) is added. If the input property name is found, the associated NamedValue is removed, then a new NamedValue (with the input property name and value) is added.

Context::set_values

Adds one or more properties.

Original class

CORBA::Context

IDL syntax

```
CORBA::Status set_values(CORBA::NVList_ptr values);
```

Parameters

values A pointer to an NVList containing the properties to be set. Context properties follow the rules for OMG IDL identifiers. The property names should not end with an asterisk. Property names must be non-null, or a system exception is raised. Currently, only strings are supported as property values. It is legal to pass a null property value. In the NVList, the flags field must be set to zero. The caller retains ownership of this parameter.

Return value

CORBA::Status
A zero return code indicates the properties were successfully added.

Exceptions

CORBA::SystemException

Remarks

The set_values method adds one or more properties to a context. If an input property names is not found in the property list, a new NamedValue (wtih the input property name and value) is added. If the input property name is found, the associated NamedValue is removed, then a new NamedValue (wtih the input property name and value) is added.

ContextList Class

Specifies the list of properties sent with a DII request.

File name

contextl.h

Intended usage

When a client assembles a Dynamic Invocation Interface request, a ContextList is optionally included. A ContextList specifies the list of properties sent with a request and is used to improve performance. When invoking a request without a ContextList, the ORB looks up context information in the Interface Repository. The ORB::create_context_list method is called to create an empty context list. The ContextList class provides methods to add and delete a property, as well as query information about a context list. Note that a context list contains only property names, not property values. Associations between property names and property values are maintained in a Context object. For additional information, see the Context and Request class descriptions.

Supported methods

ContextList::_duplicate
ContextList::_nil
ContextList::add
ContextList::add_consume
ContextList::count
ContextList::item
ContextList::remove

ContextList::add

Adds a single property name to a context list.

Original class

CORBA::ContextList

IDL syntax

```
void add(const char *ctxt);
```

Parameters

ctxt The name of the property to be added. Property names follow the rules for OMG IDL identifiers and should not end with an asterisk. A system exception is raised if the input property name is null.

Exceptions

CORBA::SystemException

Remarks

The add method is used by a client program to populate the ContextList associated with a DII request. The add method adds a single property name to a context list. The add and add_consume methods perform the same task but differ in memory management. The add method does not assume ownership of the input property name; the add_consume method does.

ContextList::add_consume

Adds a single property name to a context list.

Original class

CORBA::ContextList

IDL syntax

```
void add_consume(char *ctxt);
```

Parameters

ctxt The name of the property to be added. Property names follow the rules for OMG IDL identifiers and should not end with an asterisk. The property name must be allocated using the CORBA::string_alloc method. Ownership of this parameter transfers to the ContextList. A system exception is raised if the input property name is null.

Exceptions

CORBA::SystemException

Remarks

The add_consume method is used by a client program to populate the ContextList associated with a DII request. The add_consume method adds a single property name to a context list. The add and add_consume methods perform the same task but differ in memory management. The add_consume method assumes ownership of the input property name; the add method does not. The caller must not access the memory referred to by the input parameter after it has been passed in.

ContextList::count

Retrieves the number of elements in a context list.

Original class

CORBA::ContextList

IDL syntax

```
CORBA::ULong count();
```

Return value

CORBA::ULong
The number of elements in the context list.

Remarks

The count method is used by a client program when querying the ContextList associated with a DII request. The count method returns the number of elements in a context list.

ContextList::_duplicate

Duplicates a ContextList object.

Original class

CORBA::ContextList

IDL syntax

```
static CORBA::ContextList_ptr _duplicate (CORBA::ContextList_ptr p);
```

Parameters

p The ContextList object to be duplicated. The reference can be nil, in which case the return value will also be nil.

Return value

CORBA::ContextList_ptr
The new ContextList object reference. This value should subsequently be released using CORBA::release(ContextList_ptr).

Remarks

This method is intended to be used by client and server applications to duplicate a reference to a ContextList object. Both the original and the duplicate reference should subsequently be released using CORBA::release(ContextList_ptr).

ContextList::item

Retrieves the property name associated with an input index.

Original class

CORBA::ContextList

IDL syntax

```
const char * item(CORBA::ULong index)
```

Parameters

index The index of the desired property name, starting at zero. A system exception is raised if the input index is greater than or equal to the number of elements in the context list.

Return value

const char *
The property name associated with the input index. Ownership of the return value is maintained by the ContextList; the return value must not be freed by the caller.

Exceptions

CORBA::SystemException

Remarks

The item method is used by a client program when querying the ContextList associated with a DII request. The item method returns the property name associated with an input index.

ContextList::_nil

Returns a nil CORBA::ContextList reference.

Original class

CORBA::ContextList

IDL syntax

```
static CORBA::ContextList_ptr _nil ();
```

Return value

CORBA:ContextList_ptr
A nil ContextList reference.

Remarks

This method is intended to be used by client and server applications to create a nil ContextList reference.

ContextList::remove

Deletes the property name associated with an input index.

Original class

CORBA::ContextList

IDL syntax

```
CORBA::Status remove(CORBA::ULong index);
```

Parameters

index The index of the property name to be deleted, starting at zero. A system exception is raised if the input index is greater than or equal to the number of elements in the context list.

Return value

CORBA::Status
A zero return code indicates the property name was successfully deleted.

Exceptions

CORBA::SystemException

Remarks

The remove method is used by a client program when deleting an element of the context list associated with a DII request. The remove method deletes the property name associated with an input index. The remaining property names are re-indexed.

CORBA Class

Encompasses the interfaces that make up the CORBA-compliant ORB, the TypeCode library, and the Interface Repository Framework.

File name

corba.h

Intended usage

The intended use of the CORBA class is the same as that of the CORBA module; see “Chapter 2. CORBA in Object Request Broker” on page 7.

Nested classes

AliasDef Interface
Any Class
ArrayDef Interface
AttributeDef Interface
BOA Class
ConstantDef Interface
Contained Interface
Container Interface
Context Class
ContextList Class
Current Class
DynamicImplementation Class
EnumDef Interface
Environment Class
Exception Class
ExceptionDef Interface
ExceptionList Class
IDLType Interface
IRObject Interface
ImplRepository Class
ImplementationDef Interface
InterfaceDef Interface
ModuleDef Interface
NVList Class
NamedValue Class
ORB Class
Object Class
OperationDef Interface
Policy Interface
PrimitiveDef Interface
Principal Interface
Repository Interface
Request Class
RequestSeq Class
SequenceDef Interface
ServerRequest Class
StringDef Interface
StructDef Interface
SystemException Class
TypeCode Class
TypedefDef Interface
UnionDef Interface
UnknownUserException Class
UserException Class

Types

```
typedef unsigned char Boolean;
    typedef unsigned char Char;
    typedef wchar_t WChar;
    typedef unsigned char Octet;
    typedef short Short;
    typedef unsigned short UShort;
    typedef long Long;
    typedef unsigned long ULong;
    typedef float Float;
    typedef double Double;
    typedef char* String;
    typedef WChar_t* WString;
    typedef void Void;
    typedef ULong Status;
    typedef ULong Flags;
typedef Environment* Environment_ptr;
    typedef Request* Request_ptr;
    typedef ServerRequest* ServerRequest_ptr;
    typedef Object* Object_ptr;
    typedef BOA* BOA_ptr;
    typedef ORB* ORB_ptr;
    typedef Context* Context_ptr;
    typedef ContextList* ContextList_ptr;
    typedef Exception* Exception_ptr;
    typedef ExceptionList* ExceptionList_ptr;
    typedef NamedValue* NamedValue_ptr;
    typedef NVList* NVList_ptr;
    typedef ImplementationDef* ImplementationDef_ptr;
    typedef ImplRepository* ImplRepository_ptr;
    typedef InterfaceDef* InterfaceDef_ptr;
    typedef OperationDef* OperationDef_ptr;
    typedef Principal* Principal_ptr;
    typedef TypeCode* TypeCode_ptr;
    typedef Any* Any_ptr;
typedef char* ORBid;
typedef UShort ServiceType;
typedef ULong ServiceOption;
typedef ULong ServiceDetailType;
struct ServiceDetail {
       ServiceDetailType service_detail_type;
       _IDL_SEQUENCE_Octet service_detail;
       void encodeOp (Request &r) const;
       void decodeOp (Request &r);
       void decodeInOutOp (Request &r);
       }; // struct ServiceDetail
struct ServiceInformation {
       _IDL_SEQUENCE_CORBA_ULong_0 service_options;
       SeqServiceDetail service_details;
       void encodeOp (Request &r) const;
       void decodeOp (Request &r);
       void decodeInOutOp (Request &r);
```

```
       }; // struct ServiceInformation
enum CompletionStatus { COMPLETED_YES, COMPLETED_NO, COMPLETED_MAYBE };
enum exception_type { NO_EXCEPTION, USER_EXCEPTION, SYSTEM_EXCEPTION };
enum PolicyType { SecClientInvocationAccess, SecTargetInvocationAccess,
        SecApplicationAccess, SecClientInvocationAudit,
        SecTargetInvocationAudit, SecApplicationAudit, SecDelegation,
        SecClientSecureInvocation, SecTargetSecureInvocation,
        SecNonRepudiation, SecConstruction,
        PolicyType_force_long=(long)0x7fffffff };
```

Constants

```
static const int ARG_IN;
    static const int ARG_OUT;
    static const int ARG_INOUT;
    static const int IN_COPY_VALUE;
    static const int DEPENDENT_LIST;
    static const int ARG_FLAGS;
    static const int OBJECT_NIL;
    static const int OUT_LIST_MEMORY;
    static const int INV_TERM_ON_ERR;
    static const int RESP_NO_WAIT;
    static const int INV_NO_RESPONSE;
    static const int CTX_RESTRICT_SCOPE;
    static const int CTX_DELETE_DESCENDENTS;
const static TypeCode_ptr _tc_null;
const static TypeCode_ptr _tc_void;
const static TypeCode_ptr _tc_short;
const static TypeCode_ptr _tc_long;
const static TypeCode_ptr _tc_ushort;
const static TypeCode_ptr _tc_ulong;
const static TypeCode_ptr _tc_float;
const static TypeCode_ptr _tc_double;
const static TypeCode_ptr _tc_boolean;
const static TypeCode_ptr _tc_char;
const static TypeCode_ptr _tc_wchar;
const static TypeCode_ptr _tc_octet;
const static TypeCode_ptr _tc_any;
const static TypeCode_ptr _tc_TypeCode;
const static TypeCode_ptr _tc_Principal;
const static TypeCode_ptr _tc_Object;
const static TypeCode_ptr _tc_string;
const static TypeCode_ptr _tc_wstring;
const static TypeCode_ptr _tc_longlong
const static TypeCode_ptr _tc_ulonglong
const static TypeCode_ptr _tc_CORBA_NamedValue;
const static TypeCode_ptr _tc_CORBA_InterfaceDescription;
const static TypeCode_ptr _tc_CORBA_OperationDescription;
const static TypeCode_ptr _tc_CORBA_AttributeDescription;
const static TypeCode_ptr _tc_CORBA_ParameterDescription;
const static TypeCode_ptr _tc_CORBA_ModuleDescription;
const static TypeCode_ptr _tc_CORBA_ConstantDescription;
const static TypeCode_ptr _tc_CORBA_ExceptionDescription;
const static TypeCode_ptr _tc_CORBA_TypeDescription;
const static TypeCode_ptr _tc_CORBA_FullInterfaceDescription;
```

```
static const ServiceType Security;
static const ServiceOption SecurityLevel1;
static const ServiceOption SecurityLevel2;
static const ServiceOption NonRepudiation;
static const ServiceOption SecurityORBServiceReady;
static const ServiceOption SecurityServiceReady;
static const ServiceOption ReplaceORBServices;
static const ServiceOption ReplaceSecurityServices;
static const ServiceOption StandardSecureInteroperability;
static const ServiceOption DCESecureInteroperability;
static const ServiceDetailType SecurityMechanismType;
static const ServiceDetailType SecurityAttribute;
```

Supported methods

CORBA::_boa
CORBA::is_nil
CORBA::ORB_init
CORBA::release
CORBA::string_alloc
CORBA::string_dup
CORBA::string_free
CORBA::wstring_alloc
CORBA::wstring_dup
CORBA::wstring_free

CORBA::_boa

Returns a pointer to the BOA object within a server.

Original class

CORBA

IDL syntax

```
static BOA_ptr _boa();
```

Return value

CORBA::BOA_ptr
A pointer to the BOA object residing in the server. The return value should be released using CORBA::release(BOA_ptr).

Remarks

Implementations of IDL interfaces, running in a server process, can use this method to obtain a BOA_ptr to the BOA object residing in the server. In the IBM implementation, _boa() is a static member function in the CORBA class, but CORBA specifies that , to

be ORB-portable, applications should not assume where _boa() is defined, only that it is available within the implementation of the IDL interface. The return value should be released using CORBA::release(BOA_ptr).

Example

See the example for the CORBA::is_nil method.

CORBA::is_nil

Indicated whether the input object pointer represents a nil object.

Original class

CORBA

IDL syntax

```
static Boolean is_nil(Any_ptr p);
static Boolean is_nil(BOA_ptr p);
static Boolean is_nil(ContextList_ptr p);
static Boolean is_nil(Context_ptr p);
static Boolean is_nil(Current_ptr p);
static Boolean is_nil(Environment_ptr p);
static Boolean is_nil(ExceptionList_ptr p);
static Boolean is_nil(Exception_ptr p);
static Boolean is_nil(NamedValue_ptr p);
static Boolean is_nil(NV_ptr p);
static Boolean is_nil(ORB_ptr p);
static Boolean is_nil(Object_ptr p);
static Boolean is_nil(Principal_ptr p);
static Boolean is_nil(Request_ptr p);
static Boolean is_nil(ServerRequest_ptr p);
static Boolean is_nil(TypeCode_ptr p);
```

Parameters

p The object pointer to be tested. This pointer can be NULL.

Return value

CORBA::Boolean

0 The input object pointer is valid.

1 The input object pointer refers to a nil object

Remarks

This method is intended to be used by client and server applications to determine whether an object pointer is nil. This test should be used to verify the validity of the object prior to invoking any methods on it. This method has different signatures for different types of objects.

Example

```
/* The following is a C++ example */
#include "corba.h"
...
/* Retrieve the pointer in BOA object */
CORBA::BOA_ptr pBOA;
pBOA = CORBA::_boa();
/* Test if the pointer refers to a nil object */
CORBA::Boolean bool;
bool = CORBA::is_nil(pBOA);
if (bool == TRUE)
  {
  /* pBOA refers to a nil object, return or generate exception */
  ...
  }
else
  {
  /* proceed, using pBOA */
  ...
```

CORBA::ORB_init

Initializes the ORB, if necessary, and returns a pointer to it. For workstation implementations, be sure to specify `DSOM` using the parameter *argv* or *orb_identifier* as described below.

Original class

CORBA

IDL syntax

```
typedef char* ORBid;
static ORB_ptr ORB_init (int& argc, char** argv, ORBid orb_identifier);
```

Parameters

argc The number of strings in the *argv* array of strings. This is typically the *argc* parameter passed in to the main() function of the application.

argv An array of strings, whose size is indicated by the *argc* parameter. This is typically the *argv* parameter passed in to the main() function of the application. [**Workstation Implementation**: If one of the strings in argv matches `-ORBid "DSOM"`, then ORB initialization is performed, the matching string is consumed,

and *argc* is decremented. (The remaining strings in *argv* may be reordered as part of consuming the `-ORBid "DSOM"` string.) If *argv* is NULL or contains no string that matches `-ORBid "DSOM"`, then the ORB is initialized only if the *orb_identifier* parameter is `"DSOM"`.]

orb_identifier
A string that indicates which ORB to initialize. [**Workstation Implementation**: If no string in the *argv* parameter matches `-ORBid "DSOM"`, the ORB is initialized only if the *orb_identifier* parameter is `DSOM`.]

Return value

CORBA::ORB_ptr
A pointer to the ORB object. The return result should be released using CORBA::release(ORB_ptr).

Exceptions

CORBA::SystemException

Remarks

This method is intended to be used by client or server applications to both initialize the ORB and obtain a pointer to it. This method can be called multiple times without adverse effect. The return value should be released using CORBA::release(ORB_ptr).

Initialization of data structures used for code-set translation between clients and servers is not done until ORB_init() is called, so that the application has an opportunity to first initialize its locale (for instance, using the XPG4 setlocale() function). Therefore, if a client or server process is configured to communicate with another process using a different character code set, or is configured to use ISO-Latin 1 as the transmission code set for remote messages, the application should initialize its locale (for example, using setlocale()), then call ORB_init(), prior to using the ORB or making remote method invocations.

Note: In the Component Broker environment, client code developers are strongly encouraged to use the CBSeriesGlobal::Initialize() method to properly initialize the client environment, regardless of platform. On the server side, all Component Broker execution threads are properly initialize by the runtime without assistance from the application program. That is, the use of CBSeriesGlobal::Initialize() by a server application is not required (nor is the use of ORB_init).

Example

```
/* The following is a C++ example */
#include "corba.h"
...
int main(int argc, char *argv[])
{
 char * orbid
 /* Initialize orbid. For CB workstation initialize to "DSOM" */
 int rc = 0;
  /* Initialize the ORB and obtain a pointer to it */
```

```
  CORBA::ORB_ptr p = CORBA::ORB_init(argc, argv, orbid);
  /* use p in the code */
  ...
  return rc:
}
```

CORBA::release

Release resources associated with an object or pseudo-object reference.

Original class

CORBA

IDL syntax

```
static void release(BOA_ptr p);
static void release(ContextList_ptr p);
static void release(Context_ptr p);
static void release(Current_ptr p);
static void release(Environment_ptr p);
static void release(ExceptionList_ptr p);
static void release(Exception_ptr p);
static void release(NamedValue_ptr p);
static void release(NV_ptr p);
static void release(ORB_ptr p);
static void release(Object_ptr p);
static void release(Principal_ptr p);
static void release(Request_ptr p);
static void release(ServerRequest_ptr p);
static void release(TypeCode_ptr p);
```

Parameters

p The object reference to be released.

Remarks

This method is intended to be used by client and server applications to release resources associated with object (or pseudo-object) references. CORBA::release() should be used regardless of whether the object is local or remote. A release does not necessarily perform a delete operation, and in general the delete operator should not be used for CORBA objects and pseudo-objects

When CORBA::release() is performed on a proxy to a remote implementation, the release() method only releases resources associated with the proxy; the remote implementation object is neither affected nor notified. When all resources associated with the proxy object are released, as determined by a reference count, the proxy object is automatically destroyed (but the remote object is unaffected). Likewise, when all local references to a local object are released, the object is automatically destroyed, regardless of how many remote (proxy) references to the object exist.

Managed objects are not destroyed when all their local references are released; instead, they are passivated and removed from memory when the Instance Manager determines the in-memory copy of the managed object is no longer needed.

The CORBA::release() method has different signatures for different types of objects and pseudo-objects.

See also the CORBA::Object::_duplicate() method, which is used to increase the reference count of an object reference. The _narrow() methods defined by the C++ bindings also do an implicit CORBA::Object::_duplicate().

Example

```
/* The following is a C++ example */
#include "corba.h"
#include <string.h>
...
int main(int argc, char *argv[])
{
  CORBA::Object_ptr objPtr;
  string str;
  /* Construct the string */
  CORBA::ORB_ptr op; /* assume op is initialized */
  /* Make string to object */
  objPtr = op->string_to_object(str);
  /* Proceed with objPtr */
  ...
  CORBA::string_free(string);
  CORBA::release(objPtr);
}
```

CORBA::string_alloc

Allocates storage for a string.

Original class

CORBA

IDL syntax

```
static char* string_alloc(CORBA::ULong len);
```

Parameters

len The size of the string whose storage is to be allocated. An additional byte is also allocated for the terminating NULL character.

Return value

char* The uninitialized string storage. This storage should later be freed using CORBA::string_free(). NULL is returned if the storage cannot be allocated.

Remarks

This method is intended to be used by client and server applications to dynamically allocate storage for data of type CORBA::String. The returned storage should subsequently be freed using CORBA::string_free(). Strings can also be copied using CORBA::string_dup().

Strings to be passed on remote method invocations or whose ownership is to be transferred by one library to another should be allocated using CORBA::string_alloc() (or CORBA::string_dup()) rather than the C++ new[] operator. This insures that string memory is deleted using the same C++ run time that originally allocated it.

Example

```
/* The following is a C++ example */
#include "corba.h"
...
/* Allocate 8 byes for string buf */
char* buf = CORBA::string_alloc(8);
/* String copy buf */
char* buf_dup = CORBA::string_dup(buf);
/* Use buf and buf_dup */
...
/* free buf and buf_dup */
CORBA::string_free(buf);
CORBA::string_free(buf_dup);
```

CORBA::string_dup

Copies a string.

Original class

CORBA

IDL syntax

```
static char* string_dup(const char* str);
```

Parameters

str The NULL-terminated string to be copied.

Return value

char* A copy of the input string. This storage should be subsequently freed using CORBA::string_free() rather than the C++ delete[] operator. NULL is returned if the storage cannot be allocated.

Remarks

This method is intended to be used by client and server applications to duplicate (copy) data of type CORBA::String. The resulting string should be subsequently freed using CORBA::string_free(). If the input value is NULL, the return value will be NULL.

Strings to be passed on remote method invocations or whose ownership is to be transferred from one library to another should be allocated using CORBA::string_alloc() or CORBA::string_dup() rather than the C++ new[] operator. This insures that string memory is deleted using the same C++ run time that originally allocated it.

Example

See the example for the CORBA::string_alloc method.

CORBA::string_free

Frees a string allocated by CORBA::string_alloc() or CORBA::string_dup().

Original class

CORBA

IDL syntax

```
static void string_free(char* str);
```

Parameters

str The string to be freed. If this parameter is NULL, the method has no effect.

Remarks

This method is intended to be used by client and server applications to release storage originally allocated using CORBA::string_alloc() or CORBA::string_dup().

Example

See the example for the CORBA::string_alloc method.

CORBA::wstring_alloc

Allocates storage for a string.

Not supported by OS/390 Component Broker

Original class

CORBA

IDL syntax

```
static wchar_t* wstring_alloc(CORBA::ULong len);
```

Parameters

len The size of the string whose storage is to be allocated.

Return value

wchar_t*

The uninitialized wstring storage. This storage should later be freed using CORBA::wstring_free(). NULL is returned if the storage cannot be allocated.

Remarks

This method is intended to be used by client and server applications to dynamically allocate storage for data of type CORBA::String. The returned storage should subsequently be freed using CORBA::wstring_free(). Strings can also be copied using CORBA::wstring_dup().

WStrings to be passed on remote method invocations or whose ownership is to be transferred from one library to another should be allocated using this method (or CORBA::wstring_dup()) rather than the C++ new[] operator. This insures that string memory is deleted using the same C++ run time that originally allocated it.

Example

```
/* The following is a C++ example */
#include "corba.h"
...
/* Allocate 8 wchars for string buf */
wchar_t* buf = CORBA::wstring_alloc(8);
/* String copy buf */
wchar_t* buf_dup = CORBA::wstring_dup(buf);
/* Use buf and buf_dup */
...
/* Free buf and buf_dup */
CORBA::wstring_free(buf);
CORBA::wstring_free(buf_dup);
```

CORBA::wstring_dup

Copies a WString.

Not supported by OS/390 Component Broker

Original class

CORBA

IDL syntax

```
static wchar_t* * wstring_dup(const wchar_t* * str);
```

Parameters

str The NULL-terminated WString to be copied.

Return value

wchar_t*

A copy of the input WString. This storage should be subsequently freed using CORBA::wstring_free() rather than the C++ delete[] operator. NULL is returned if the storage cannot be allocated.

Remarks

This method is intended to be used by client and server applications to duplicate (copy) data of type CORBA::WString. The resulting string should be subsequently freed using CORBA::wstring_free(). If the input value is NULL, the return value is NULL.

WStrings to be passed on remote method invocations or whose ownership is to be transferred from one library to another should be allocated using this method or CORBA::wstring_alloc() rather than the C++ new[] operator. This insures that string memory is deleted using the same C++ run time that originally allocated it.

Example

See the example for the CORBA::wstring_alloc method.

CORBA::wstring_free

Frees a WString allocated by CORBA::wstring_alloc() or CORBA::wstring_dup().

Not supported by OS/390 Component Broker

Original class

CORBA

IDL syntax

```
static void wstring_free (wchar_t * str);
```

Parameters

str The WString to be freed. If this parameter is NULL, the method has no effect.

Remarks

This method is intended to be used by client and server applications to release storage originally allocated using CORBA::wstring_alloc() or CORBA::wstring_dup().

Example

See the example for the CORBA::wstring_alloc method.

Current Class

Describes the current execution context.

File name

principl.h

Intended usage

This abstract base class is intended to be subclassed by various object services, such as the Security Service and the Object Transaction Service. Before a Current object can be used, it must be narrowed to the appropriate subtype. Current objects are obtained using:

AIX 390 WIN the ORB::resolve_initial_references method on AIX, O/S390, and Windows NT

SOLARIS the ORB::get_current method on Solaris

Current objects should be released using the CORBA::release(Current_ptr) method rather than the C++ delete operator.

Supported methods

Current::_duplicate
Current::_nil

Current::_duplicate

Duplicates a Current object.

Original class

CORBA::Current Class

IDL syntax

```
static CORBA::Current_ptr _duplicate (CORBA::Current_ptr p);
```

Parameters

p The Current object to be duplicated. The reference can be nil, in which case the return value will also be nil.

Return value

CORBA::Current_ptr
The new Current object reference. This value should subsequently be released using CORBA::release(Current_ptr).

Remarks

This method is intended to be used by client and server applications to duplicate a reference to a Current object. Both the original and the duplicate reference should subsequently be released using CORBA::release(Current_ptr).

Current::_nil

Returns a nil CORBA::Current reference.

Original class

CORBA::Current Class

IDL syntax

```
static CORBA::Current_ptr _nil ();
```

Parameters

p The Current object to be duplicated. The reference can be nil, in which case the return value will also be nil.

Return value

CORBA::Current_ptr
A nil Current reference.

Remarks

This method is intended to be used by client and server applications to create a nil Current reference.

DynamicImplementation Class

Allows an object to be dynamically dispatched in a server.

File name

boa.h

Intended usage

This class is intended to be subclassed by implementations of IDL interfaces so that those implementations can be invoked dynamically in a server that was not statically linked with the C++ bindings for that IDL interface. For example, this technique might be used to implement objects residing in an inter-ORB bridge server, or gateway. This class is part of the Dynamic Skeleton Interface (DSI).

Subclasses of CORBA::BOA::DynamicImplementation must implement the invoke method.

Supported methods

DynamicImplementation::invoke

DynamicImplementation::invoke

Invokes a method dynamically within a server.

Original class

CORBA::BOA::DynamicImplementation

IDL syntax

```
virtual void invoke (ServerRequest_ptr request,
                     Environment &env) throw () = 0;
```

Parameters

request
A ServerRequest object that provides information about the operation to be invoked, the target object, and the values of the in and inout parameters.

env An Environment object, to be used only when calling BOA::get_principal.

Exceptions

This method must not throw any exceptions. Instead, exceptions should be stored on the input ServerRequest object, using the exception method.

Remarks

This pure virtual method is intended to be overridden in subclasses of CORBA::BOA::DynamicImplementation. It is never called directly by applications; rather, the BOA residing in a server calls this method to dispatch remote calls on objects that inherit from CORBA::BOA::DynamicImplementation. This method is part of the Dynamic Skeleton Interface (DSI), used primarily to construct inter-ORB bridges or gateway servers.

When a remote invocation arrives at a server, if the target of the invocation is an object that inherits from CORBA::BOA::DynamicImplementation, BOA calls the invoke method on the target object. BOA constructs and passes in a ServerRequest object that contains all the information about the incoming request that is needed for the object to dispatch it. As an example, an implementation of the invoke method could do the following:

- Obtain the name of the operation to be invoked from the ServerRequest.
- Discover the signature of the operation to be invoked (for example, using the Interface Repository or a cache of InterfaceDef objects, or by invoking _get_interface on the target object).
- Create an NVList containing the TypeCodes (but not the values) corresponding to the signature of the operation to be invoked.
- Call ServerRequest::params, passing in the NVList; the ServerRequest::params method stores the in and inout parameter values in the NVList.
- Dispatch the operation on the target object using the in and inout parameter values now available from the NVList.
- Store the inout and out parameter values in the NVList.
- Call ServerRequest::result to record the operation result.

The BOA then sends the response to the calling client. If an exception is thrown by the dispatched operation, the invoke method must catch it and record it by calling ServerRequest::exception; CORBA::BOA::DynamicImplementation::invoke must never throw any exceptions.

EnumDef Interface

The EnumDef interface is used within the Interface Repository to represent an OMG IDL enumeration definition.

File name

somir.idl

Intended usage

An instance of an EnumDef object is used within the Interface Repository to represent an OMG IDL enumeration definition. An instance of an EnumDef object can be created using the create_enum operation of the Container interface.

Local-only

True

Ancestor interfaces

TypedefDef Interface

Exceptions

CORBA::SystemException

IDL syntax

```
module CORBA
   {
      typedef sequence  EnumMemberSeq;
      interface EnumDef:TypedefDef
      {
         attribute EnumMemberSeq members;
      };
   };
```

Supported operations

EnumDef::members
IDLType::type

EnumDef::members

The members read and write operations provide for the access and update of the list of elements of an OMG IDL enumeration definition (EnumDef) in the Interface Repository.

Original interface

EnumDef Interface

IDL syntax

```
attribute EnumMemberSeq members;
```

Parameters

EnumMemberSeq & members

In Read operation, no input parameters are required.

In Write operation, CORBA::EnumMemberSeq & members. The members parameter provides the list of enumeration members with which to update the EnumDef.

Return value

EnumMemberSeq *

In Read operation, CORBA::EnumMemberSeq *. The returned pointer references a sequence that is representative of the enumeration members. The memory is owned by the caller and can be released by invoking delete.

In Write operation, no value is returned.

Exceptions

CORBA::SystemException
CORBA::NO_IMPLEMENT thrown by Write operation on OS/390

Remarks

The members attribute contains a distinct name for each possible value of the enumeration. The members read operation provides access to a copy of the contents of this enumeration member list, and the members write operation provides the ability to update the members attribute.

Example

```
// C++
   // assume that 'this_enum' has already been initialized
   CORBA::EnumDef * this_enum;

   // establish and initialize 'seq_update'
   CORBA::EnumMemberSeq seq_update;
   seq_update.length (3);
   seq_update[0] = CORBA::string_dup ("enumerator_0");
   seq_update[1] = CORBA::string_dup ("enumerator_1");
   seq_update[2] = CORBA::string_dup ("enumerator_2");

   // change the 'members' information in the enumeration
   this_enum-> members (seq_update);
   // read the 'members' information from the enumeration
   CORBA::EnumMemberSeq * returned_seq;
   returned_seq = this_enum-> members ();
```

Environment Class

Holds an Exception.

File name

environm.h

Intended usage

The Environment class holds a single Exception and is used for error handling in those cases where catch/throw exception handling cannot be used. For example, Dynamic Invocation Interface uses an Environment to report errors back to the client. The ORB::create_environment method is called to create an empty Environment. The Environment class provides methods to get and set an Exception, as well as clear an Exception. For additional information, see the Exception and Request class descriptions.

Supported methods

Environment::_duplicate
Environment::_nil
Environment::clear
Environment::exception

Environment::_duplicate

Duplicates an Environment object.

Original class

CORBA::Environment

IDL syntax

```
static CORBA::Environment_ptr _duplicate (CORBA::Environment_ptr p);
```

Parameters

p The Environment object to be duplicated. The reference can be nil, in which case the return value will also be nil.

Return value

CORBA::Environment_ptr
The new Environment object reference. This value should subsequently be released using CORBA::release(Environment_ptr).

Remarks

This method is intended to be used by client and server applications to duplicate a reference to an Environment object. Both the original and the duplicate reference should subsequently be released using CORBA::release(Environment_ptr).

Environment::_nil

Returns a nil CORBA::Environment reference.

Original class

CORBA::Environment

IDL syntax

static CORBA::Environment_ptr _nil ();

Return value

CORBA::Environment_ptr
A nil Environment reference.

Remarks

This method is intended to be used by client and server applications to create a nil Environment reference.

Environment::clear

Deletes the Exception held by an Environment.

Original class

CORBA::Environment

IDL syntax

```
void clear();
```

Exceptions

CORBA::SystemException

Remarks

The clear method is used to delete the Exception, if any, held by an Environment.

Environment::exception

Gets and sets an Exception.

Original class

CORBA::Environment

IDL syntax

```
void exception(CORBA::Exception *new_exception);
CORBA::Exception *exception() const;
```

Parameters

new_exception
A pointer to the new Exception to be held in the Environment. It is valid to pass a null pointer. Ownership of this parameter transfers to the Environment.

Return value

CORBA::Exception *
A pointer to the Exception currently held in the Environment, if any, or a null pointer. Ownership of the return value is maintained by the Environment; the return value must not be freed by the caller.

Remarks

The exception method is used to get and set the Exception held by an Environment. The Exception returned by the get method continues to be owned by the Environment. Once the Environment is destroyed, the Exception previously returned from the get method is invalid. If the Environment does not hold an Exception, the get function returns a null pointer. The set method assumes ownership of the input Exception.

Exception Class

Describes an exception condition that has occurred.

File name

except.h

Intended usage

This class is intended to be caught in the catch clause of a try/catch block that encompasses remote method invocations or calls to ORB services. Typically Exception instances will actually be instances of either the SystemException or UserException subclass.

Supported methods

Exception::_duplicate
Exception::_nil
Exception::id

Exception::_duplicate

Duplicates an Exception object.

Original class

CORBA::Exception

IDL syntax

```
static CORBA::Exception_ptr _duplicate (CORBA::Exception_ptr p);
```

Parameters

p The Exception object to be duplicated. The reference can be nil, in which case the return value will also be nil.

Return value

CORBA::Exception_ptr
The new Exception object reference. This value should subsequently be released using CORBA::release(Exception_ptr).

Remarks

This method is intended to be used by client and server applications to duplicate a reference to an Exception object. Both the original and the duplicate reference should subsequently be released using CORBA::release(Exception_ptr).

Exception::_nil

Returns a nil CORBA::Exception reference.

Original class

CORBA::Exception

IDL syntax

```
static CORBA::Exception_ptr _nil ();
```

Return value

CORBA::Exception_ptr
A nil Exception reference.

Remarks

This method is intended to be used by client and server applications to create a nil Exception reference.

Exception::id

Indicates the runtime type of an Exception.

Original class

CORBA::Exception

IDL syntax

```
const char * id() const;
```

Return value

const char *
The string name of the runtime type of the Exception object. The Exception object retains ownership of this string and the caller should not attempt to free it.

Remarks

This method is intended to be used when an Exception is caught (in the catch clause of a try/catch block), to determine the exact type of Exception that was thrown. For example, if a CORBA::NO_MEMORY exception is thrown and caught as a generic CORBA::Exception, the id() method can be invoked on the Exception, which will yield “CORBA::NO_MEMORY”.

ExceptionDef Interface

The ExceptionDef interface is used by the Interface Repository to represent an exception definition.

File name

somir.idl

Intended usage

The ExceptionDef object is used to represent an exception definition. An ExceptionDef object may be created in the Interface Repository database and an associated memory image of the object by calling the create_exception operation of the Container interface. The create_exception parameters include the unique RepositoryId (CORBA::RepositoryId), the name (CORBA::Identifier), the version (CORBA::VersionSpec), and a sequence indicating the exception members (CORBA::StructMemberSeq). The sequence may have zero elements to allow the ExceptionDef to have no members.

Local-only

True

Ancestor interfaces

Contained Interface

Exceptions

CORBA::SystemException

IDL syntax

```
module CORBA
   {
      interface ExeptionDef:Contained
      {
         readonlyattribute TypeCode type;
         attribute StructMemberSeq members;
      };
      struct ExceptionDescription
      {
         Identifier name;
         RepositoryId id;
         RepositoryId defined_in;
         VersionSpec version;
         TypeCode type;
      };
   };
```

Supported operations

ExceptionDef::describe
ExceptionDef::members
IDLType::type

ExceptionDef::describe

The describe operation returns a structure containing information about a CORBA::ExceptionDef Interface Repository object.

Original interface

ExceptionDef Interface

IDL syntax

```
struct ExceptionDescription
   {
      Identifier name;
      RepositoryId id;
      RepositoryId defined_in;
      VersionSpec version;
      TypeCode type;
   };
struct Description
   {
      DefinitionKind kind;
      any value;
   };
Description describe ();
```

Parameters

None.

Return value

Description *

The returned value is a pointer to a CORBA::Contained::Description structure. The memory is owned by the caller and can be removed by invoking delete.

Exceptions

CORBA::SystemException

Remarks

The inherited describe operation returns a structure (CORBA::Contained::Description) that contains information about a CORBA::ExceptionDef Interface Repository object. The CORBA::Contained::Description structure has two fields: kind (CORBA::DefinitionKind data type), and value (CORBA::Any data type).

The kind of definition described by the returned structure is provided using the kind field, and the value field is a CORBA::Any that contains the description that is specific to the kind of object described. When the describe operation is invoked on an exception (CORBA::ExceptionDef) object, the kind field is equal to CORBA::dk_Exception and the value field contains the CORBA::ExceptionDescription structure.

Example

```
// C++
   // assume that 'this_exception' has already been initialized
   CORBA::ExceptionDef * this_exception;
```

```
// retrieve a description of the exception
CORBA::ExceptionDef::Description * returned_description;
returned_description = this_exception-> describe ();

// retrieve the exception description from the returned description
// structure
CORBA::ExceptionDescription * exception_description;
exception_description =
   (CORBA::ExceptionDescription *) returned_description value.value ();
```

ExceptionDef::members

The members read and write operations provide for the access and update of the list of elements of an OMG IDL exception definition (CORBA::ExceptionDef) in the Interface Repository.

Original interface

ExceptionDef Interface

IDL syntax

```
attribute StructMemberSeq members;
```

Parameters

StructMemberSeq & members

In Read operation, no input parameters are defined.

In Write operation, CORBA::StructMemberSeq & members. The members parameter provides the list of exception members with which to update the ExceptionDef. Setting the members attribute also updates the type attribute.

Return value

StrucMemberSeq *

In Read operation, CORBA::StructMemberSeq *. The returned pointer references a sequence that is representative of the exception members. The memory is owned by the caller and can be released by invoking delete.

In Write operation, no value is returned.

Exceptions

CORBA::SystemException

Remarks

The members attribute contains a description of each exception member. The members read and write operations allow the access and update of the members attribute.

This method is implemented in the OS/390 IR for use by IDL compiler emitted IR install client program. Use of this method in a generalized programming environment is strongly discouraged since it could jeopardize coherency of the IR. In general, the IR install client programs should be the only means for updating the contents of the Interface Repository.

Example

```
// C++
   // assume 'this_exception_def', 'struct_1', and 'struct_2'
   // have already been initialized
   CORBA::ExceptionDef * this_exception_def;
   CORBA::StructDef * struct_1;
   CORBA::StructDef * struct_2;

   // establish and initialize the StructMemberSeq . . .
   CORBA::StructMemberSeq seq_update;
   seq_update.length (2);
   seq_update[0].name = CORBA::string_dup ("struct_1");
   seq_update[0].type_def = CORBA::IDLType::_duplicate (struct_1);
   seq_update[1].name = CORBA::string_dup ("struct_2");
   seq_update[1].type_def = CORBA::IDLType::_duplicate (struct_2);

   // set the members attribute of the ExceptionDef
   this_exception_def-> members (seq_update);

   // read the members attribute information from the ExceptionDef
   CORBA::StructMemberSeq * returned_members;
   returned_members = this_exception_def-> members ();
```

ExceptionList Class

Specifies the list of user-defined exceptions that can be raised when a DII request is executed.

File name

excp_lst.h

Intended usage

When a client assembles a Dynamic Invocation Interface request, an ExceptionList is optionally included. An ExceptionList specifies the list of TypeCodes for all user-defined exceptions that can be raised when a request is executed. The ExceptionList class is used to improve performance. When invoking a request without an ExceptionList, the ORB looks up user-defined exception information in the Interface Repository. The ORB::create_exception_list method is called to create an empty exception list. The ExceptionList class provides methods to add and delete an exception, as well as query information about an exception list. For additional information, see the Exception, UserException, and Request class descriptions.

Supported methods

ExceptionList::_duplicate
ExceptionList::_nil
ExceptionList::add
ExceptionList::add_consume
ExceptionList::count
ExceptionList::item
ExceptionList::remove

ExceptionList::_duplicate

Duplicates an ExceptionList object.

Original class

CORBA::ExceptionList

IDL syntax

```
static CORBA::ExceptionList_ptr _duplicate (CORBA::ExceptionList_ptr p);
```

Parameters

p The ExceptionList object to be duplicated. The reference can be nil, in which case the return value will also be nil.

Return value

CORBA::ExceptionList_ptr
The new ExceptionList object reference. This value should subsequently be released using CORBA::release(ExceptionList_ptr).

Remarks

This method is intended to be used by client and server applications to duplicate a reference to an ExceptionList object. Both the original and the duplicate reference should subsequently be released using CORBA::release(ExceptionList_ptr).

ExceptionList::_nil

Returns a nil CORBA::ExceptionList reference.

Original class

CORBA::ExceptionList

IDL syntax

```
static CORBA::ExceptionList_ptr _nil ();
```

Return value

CORBA::ExceptionList_ptr
A nil ExceptionList reference.

Remarks

This method is intended to be used by client and server applications to create a nil ExceptionList reference.

ExceptionList::add

Adds a single user-defined exception to an exception list.

Original class

CORBA::ExceptionList

IDL syntax

```
void add(CORBA::TypeCode_ptr tc);
```

Parameters

tc A pointer to the TypeCode for the user-defined exception. A system exception is raised if the input pointer is null.

Exceptions

CORBA::SystemException

Remarks

The add method is used by a client program to populate the ExceptionList associated with a DII request. The add method adds a single user-defined exception to an exception list. The add and add_consume methods perform the same task but differ in memory management. The add method does not assume ownership of the input TypeCode; the add_consume method does.

ExceptionList::add_consume

Adds a single user-defined exception to an exception list.

Original class

CORBA::ExceptionList

IDL syntax

```
void add_consume(CORBA::TypeCode_ptr tc);
```

Parameters

tc A pointer to the TypeCode for the user-defined exception. The input TypeCode must either be retrieved from the Interface Repository or allocated using the ORB::create_exception_tc method. Ownership of this parameter transfers to the ExceptionList. A system exception is raised if the input pointer to the TypeCode is null.

Exceptions

CORBA::SystemException

Remarks

The add_consume method is used by a client program to populate the ExceptionList associated with a DII request. The add_consume method adds a single user-defined exception to an exception list. The add and add_consume methods perform the same task but differ in memory management. The add_consume method assumes ownership of the input TypeCode; the add method does not. The caller must not access the object referred to by the input parameter after it has been passed in.

ExceptionList::count

Retrieves the number of elements in an exception list.

Original class

CORBA::ExceptionList

IDL syntax

```
CORBA::ULong count();
```

Return value

CORBA::ULong
The number of elements in the exception list.

Remarks

The count method is used by a client program when querying the ExceptionList associated with a DII request. The count method returns the number of elements in an exception list.

ExceptionList::item

Retrieves the user-defined exception associated with an input index.

Original class

CORBA::ExceptionList

IDL syntax

```
CORBA::TypeCode_ptr item(CORBA::ULong index)
```

Parameters

index The index of the desired user-defined exception, starting at zero. A system exception is raised if the input index is greater than or equal to the number of elements in the exception list.

Return value

CORBA::TypeCode_ptr
A pointer to the TypeCode for the user-defined exception. Ownership of the return value is maintained by the ExceptionList; the return value must not be freed by the caller.

Exceptions

CORBA::SystemException

Remarks

The item method is used by a client program when querying the ExceptionList associated with a DII request. The item method returns the user-defined exception associated with an input index.

ExceptionList::remove

Deletes the user-defined exception associated with an input index.

Original class

CORBA::ExceptionList

IDL syntax

```
CORBA::Status remove(CORBA::ULong index);
```

Parameters

index The index of the user-defined exception to be deleted, starting at zero. A system exception is raised if the input index is greater than or equal to the number of elements in the exception list.

Return value

CORBA::Status

A zero return code indicates the user-defined exception was successfully deleted.

Exceptions

CORBA::SystemException

Remarks

The remove method is used by a client program when deleting an element of the ExceptionList associated with a DII request. The remove method deletes the user-defined exception associated with an input index. The remaining exceptions are re-indexed.

IDLType Interface

The IDLType interface is an abstract interface inherited by all Interface Repository objects that represent OMG IDL types. It provides access to the TypeCode that describes the type. The IDL Type is used in defining other interfaces whenever definitions of IDL types must be referenced.

File name

somir.idl

Intended usage

The IDLType interface is not itself instantiated as a means of accessing the Interface Repository. As an ancestor to Interface Repository objects that represent OMG IDL types, it provides a specific operation as noted below. Those Interface Repository objects that inherit (directly or indirectly) the operation defined in IDLType include: StructDef, UnionDef, EnumDef, AliasDef, PrimitiveDef, StringDef, WstringDef, SequenceDef, ArrayDef, and InterfaceDef.

Local-only

True

Ancestor interfaces

IRObject Interface

Exceptions

CORBA::SystemException

IDL syntax

```
module CORBA
   {
      interface IDLType:IRObject
      {
         readonly attribute TypeCode type;
      };
   };
```

Supported operations

IDLType::type

IDLType::type

The type operation retrieves a TypeCode pointer representative of specific Interface Repository objects.

Original interface

IDLType Interface

IDL syntax

```
readonly attribute TypeCode type;
```

Parameters

None.

Return value

TypeCode_ptr

The return value is a pointer to a TypeCode that describes the object. The memory associated with the returned TypeCode pointer is owned by the caller and can be released by calling CORBA::release.

Exceptions

CORBA::SystemException

Remarks

The type attribute (a TypeCode *) describes all objects derived from IDLType. The type read operation retrieves a pointer to a copy of the type attribute. Object types that inherit from IDLType and therefore support the type read operation are ArrayDef, SequenceDef, StringDef, WstringDef, PrimitiveDef, UnionDef, StructDef, AliasDef, EnumDef, and InterfaceDef.

There are other Interface Repository objects that do not inherit from IDLType that also have a type method that returns a TypeCode * representative of the specific object. The Interface Repository interfaces that have their own type method include: ConstantDef, ExceptionDef, and AttributeDef.

Example

```
// C++
   // assume that 'union_1' has already been initialized
   CORBA::UnionDef * union_1;

   // retrieve the TypeCode information which represents 'union_1' . . .
   CORBA::TypeCode * typecode_ptr;
   typecode_ptr = union_1-> type();
```

ImplRepository Class

Represents a persistent data store of ImplementationDef objects, each representing a logical server that has been registered.

File name

implrep

Intended usage

The ImplRepository class represents the Implementation Repository, a persistent data store of ImplementationDef objects, each representing a logical server that has been registered.

The ImplRepository class is intended to be used by server applications, using the find_impldef or find_impldef_by_alias method, to retrieve the ImplementationDef representing that server application. Each server must retrieve its own ImplementationDef object from the Implementation Repository (using the ImplRepository class), because the ImplementationDef is a parameter required by the BOA::impl_is_ready method.

The other methods of the ImplRepository class are not typically used by client or server applications, but can be used by an application to programmatically administer the Implementation Repository. (Typically servers are registered and unregistered from the Implementation Repository using the product tools, rather than programmatically by an application.)

Supported methods

ImplRepository::find_impldef
ImplRepository::find_impldef_by_alias

ImplRepository::find_impldef

Retrieves an ImplementationDef from the Implementation Repository, by server id.

This is an IBM-defined method, which extends the CORBA 2.0 specifications provided by the OMG.

Original class

CORBA::ImplRepository

IDL syntax

```
CORBA::ImplementationDef* find_impldef(const char * ImplId);
```

Parameters

ImplId The server id of the ImplementationDef to be retrieved. A CORBA::SystemException will be thrown if this string is NULL or is not in the proper format of an Implementation Repository UUID (as created by the ImplementationDef constructor).

Return value

CORBA::ImplementationDef*

The CORBA::ImplementationDef from the Implementation Repository whose id matches the input. The caller assumes ownership of this object and must subsequently delete it. (If passed to BOA::impl_is_ready, however, the ImplementationDef object should not be deleted until after BOA::deactivate_impl has subsequently been called and the server has quiesced, to insure that the BOA does not subsequently refer to that object.) Each call to find_impldef returns a different ImplementationDef object (although the different objects will be equivalent for equivalent input to find_impldef).

Exceptions

CORBA::SystemException

Remarks

This method is used to retrieve an ImplementationDef object from the Implementation Repository, using the server id as a key. This method (or the find_impldef_by_alias method) is intended to be used by a server application to retrieve its own ImplementationDef object from the Implementation Repository; a server application then passes this ImplementationDef object to the BOA::impl_is_ready and BOA::deactivate_impl methods.

A CORBA::SystemException is thrown if the Implementation Repository cannot be accessed (for instance, due to a configuration error), if the input server id is NULL or is not in the correct format, or if the specified server id cannot be found in the Implementation Repository.

ImplRepository::find_impldef_by_alias

Retrieves an ImplementationDef from the Implementation Repository, by server alias.

This is an IBM-defined method, which extends the CORBA 2.0 specifications provided by the OMG.

Original class

CORBA::ImplRepository

IDL syntax

```
CORBA::ImplementationDef* find_impldef(const char * ImplAlias);
```

Parameters

ImplAlias
The server alias of the ImplementationDef to be retrieved. A CORBA::SystemException will be thrown if this string is NULL.

Return value

CORBA::ImplementationDef*
The CORBA::ImplementationDef from the Implementation Repository whose alias matches the input. The caller assumes ownership of this object and must subsequently delete it. (If passed to BOA::impl_is_ready, however, the ImplementationDef object should not be deleted until after BOA::deactivate_impl has subsequently been called and the server has quiesced, to insure that the BOA does not subsequently refer to that object.) Each call to find_impldef_by_alias returns a different ImplementationDef object (although the different objects will be equivalent for equivalent input to find_impldef_by_alias).

Exceptions

CORBA::SystemException

Remarks

This method is used to retrieve an ImplementationDef object from the Implementation Repository, using the server alias as a key. This method (or the find_impldef method) is intended to be used by a server application to retrieve its own ImplementationDef object from the Implementation Repository; a server application then passes this ImplementationDef object to the BOA::impl_is_ready and BOA::deactivate_impl methods.

A CORBA::SystemException is thrown if the Implementation Repository cannot be accessed (for instance, due to a configuration error), if the input server alias is NULL, or if the specified server alias cannot be found in the Implementation Repository.

ImplementationDef Interface

Describes a logical server as registered in the Implementation Repository.

File name

impldef.h

Intended usage

CORBA::ImplementationDef objects represent logical server applications. They are stored persistently in the Implementation Repository, represented programmatically by the CORBA::ImplRepository class. ImplementationDef objects are stored and updated in the Implementation Repository as servers are registered, unregistered, or changed. Typically this administration of the Implementation Repository is done using the product tools, but it can also be done programmatically (using the ImplementationDef and ImplRepository classes).

The CORBA::ImplementationDef class is used in the following ways:

- By the somorbd daemon, to find and activate servers;
- By server applications, to initialize themselves (the BOA::impl_is_ready and BOA::deactivate_impl methods require an ImplementationDef parameter);
- By applications written to programmatically query or update the contents of the Implementation Repository (typically this is done using the product tools);
- By client applications, to discover information about servers with which they are communicating, using the CORBA::Object::_get_implementation method.

ImplementationDef objects contain the following data to describe registered servers:

- Server ID (a UUID that uniquely identifies the server throughout a network and is used as a key into the Implementation Repository),
- Server alias (a user-friendly, administrator-defined name that uniquely identifies the server on a given machine, but not necessarily throughout the network, and can be used as a key into the Implementation Repository),
- Server program name (the executable that implements the logical server, which the somorbd daemon starts to activate the server on demand; this need not be a fully-qualified pathname, and needn't be unique to a particular server),
- The communication protocol(s) that the server supports (e.g., SOMD_TCPIP, SOMD_IPC),
- The key to the server's configuration data, if different from the somorbd daemon's (set by the somorbd daemon before starting the server),
- Flag bits, defined and used by the ORB.

In addition, applications can store arbitrary name/value pairs in ImplementationDef objects; these values can be used by the application to control server behavior.

Types

```
typedef sequence<nameValue> seq_nameValue;
typedef sequence<string> seq_string;
struct nameValue {
  string name;
  string value;
};
```

Supported operations

ImplementationDef::get_alias
ImplementationDef::get_id

ImplementationDef::get_alias

Retrieves the user-defined alias of the logical server represented by an ImplementationDef.

This is an IBM-defined operation, which extends the CORBA 2.0 specifications provided by the OMG.

Original interface

CORBA::ImplementationDef

IDL syntax

```
string get_id();
```

Return value

string The alias of the logical server represented by the ImplementationDef. The caller assumes ownership of the returned string, and should subsequently deallocate it using the CORBA::string_free function.

Exceptions

CORBA::SystemException

Remarks

This method is intended to be used by an application to retrieve the user-friendly, administrator-assigned name (alias) of a logical server registered in the Implementation Repository.

ImplementationDef::get_id

Retrieves the ORB-assigned UUID of the logical server represented by an ImplementationDef.

This is an IBM-defined operation, which extends the CORBA 2.0 specifications provided by the OMG.

Original interface

CORBA::ImplementationDef

IDL syntax

```
string get_id();
```

Return value

string The UUID (id) of the logical server represented by the ImplementationDef. The caller assumes ownership of the returned string, and should subsequently deallocate it using the CORBA::string_free function.

Exceptions

CORBA::SystemException

Remarks

This method is intended to be used by an application to retrieve the ORB-assigned UUID of a logical server registered in the Implementation Repository. This id is unique throughout the network and can be used as a key into the Implementation Repository.

InterfaceDef Interface

The InterfaceDef object represents an interface definition in the Interface Repository.

File name

somir.idl

Intended usage

The InterfaceDef object is used to represent an interface definition. An InterfaceDef object may be created in the Interface Repository database and an associated memory image of the object by calling the create_interface operation of the Container interface. The create_interface parameters include the unique RepositoryId (CORBA::RepositoryId), the name (CORBA::Identifier), the version (CORBA::VersionSpec), and a sequence indicating the base interfaces from which the interface inherits.

Local-only

True

Ancestor interfaces

Contained Interface
Container Interface
IDLType Interface

IDL syntax

```
module CORBA
   {
      interface InterfaceDef;
      typedef sequence  InterfaceDefSeq;
      typedef sequence  RepositoryIdSeq;
      typedef sequence  OpDescriptionSeq;
      typedef sequence  AttrDescriptionSeq;
      Interface InterfaceDef: Container, Contained, IDLType
         {
            //read/write interface
            attribute InterfaceDefSeqbase_interfaces;
            //read interface
            boolean is_a(in RepositoryId interface_id);
         struct FullInterfaceDescription
         {
             Identifier name;
             Repository Id id;
             RepositoryId defined_in;
             VersionSpec version;
             OpDescriptionSeq operations;
             AttrDescriptionSeq attributes;
             RepositoryIdSeq base_interfaces;
             TypeCode type;
          };
         FullInterfaceDescription describe_interface();
         // write interface
         AttributeDef create_attribute (in RepositoryId id,
                                        in Identifier name,
                                        in VersionSpec version,
                                        in IDLType type,
                                        in AttributeMode mode);
         OperationDef create_operation (in RepositoryId id,
                                        in Identifier name,
                                        in VersionSpec version,
                                        in IDLType result,
                                        in OperationMode mode,
                                        In ParDescriptionSeq params,
                                        In ExceptionDefSeq exceptions,
                                        in ContextIdSeq contexts);
         };
         struct InterfaceDescription
         {
             Identifier name;
             RepositoryId id;
             RepositoryID defined_in;
             VersionSpec version;
             RepositoryId Seqbase_interfaces;
         };
```

Supported operations

InterfaceDef::base_interfaces
InterfaceDef::create_attribute
InterfaceDef::create_operation
InterfaceDef::describe
InterfaceDef::describe_interface
InterfaceDef::is_a

InterfaceDef::base_interfaces

The base_interface attribute is a list of all the interfaces from which the current interface directly inherits. The base_interface operations read/write the base_interface attribute of the target interface.

Original interface

InterfaceDef Interface

IDL syntax

```
attribute InterfaceDefSeq base_interfaces;
```

Parameters

InterfaceDefSeq

In Read Operation, no input parameters are required.

In Write operation, CORBA::InterfaceDefSeq. The sequence defines the new list of InterfaceDefs from which the target interface will be changed to inherit.

Return value

InterfaceDefSeq *

In Read operation, CORBA::InterfaceDefSeq *. A pointer is returned to a copy of the base_interfaces attribute. The memory associated with the returned value is owned by the caller and may be released by invoking CORBA::delete.

In Write operation, no value is returned.

Exceptions

CORBA::SystemException

Remarks

Both Read and Write methods are supported, with parameters given above.

This method is implemented in the OS/390 IR for use by IDL compiler emitted IR install client program. Use of this method in a generalized programming environment is

strongly discouraged since it could jeopardize coherency of the IR. In general, the IR install client programs should be the only means for updating the contents of the Interface Repository.

Example

```
// C++
   // assume 'interface_1', 'interface_99', and 'interface_100'
   // have already been initialized . . .
   CORBA::InterfaceDef * interface_1;
   CORBA::InterfaceDef * interface_99;
   CORBA::InterfaceDef * interface_100;

   // establish a new list for the interface_1 inheritance
   CORBA::InterfaceDefSeq new_base_interfaces;
   new_base_interfaces.length (2);
   new_base_interfaces[0] = CORBA::InterfaceDef::_duplicate (interface_99);
   new_base_interfaces[1] = CORBA::InterfaceDef::_duplicate (interface_100);

   // change the base interfaces for interface_1
   interface_1-> base_interfaces (new_base_interfaces);

   // retrieve the 'base interfaces' attribute
   CORBA::InterfaceDefSeq * returned_base_interfaces;
   returned_base_interfaces = interface_1-> base_interfaces ();
```

InterfaceDef::create_attribute

The create_attribute operation adds a new attribute definition to an interface definition on which it is invoked.

Original interface

InterfaceDef Interface

IDL syntax

```
AttributeDef create_attribute (in RepositoryId id,
                               in Identifier name,
                               in VersionSpec version,
                               in IDLType type,
                               in AttributeMode mode);
```

Parameters

mode Valid attribute mode values are CORBA::ATTR_NORMAL (read/write access) and CORBA::ATTR_READONLY (read only access).

name The name that will be associated with this CORBA::AttributeDef object in the Interface Repository.

type_def
The type_def parameter is a CORBA::IDLType pointer that specifies the type of the CORBA::AttributeDef.

id The id represents the CORBA::RepositoryId that will uniquely identify this CORBA::AttributeDef within the Interface Repository.

version
The version number that will be associated with this CORBA::AttributeDef object in the Interface Repository.

Return value

AttributeDef *
The returned value is a pointer to the created CORBA::AttributeDef object. The memory is owned by the caller and may be released using CORBA::release.

Exceptions

CORBA::SystemException

Remarks

The create_attribute operation adds a new CORBA::AttributeDef contained in the CORBA::InterfaceDef on which it is invoked. A representation of the new CORBA::AttributeDef object is created in the Interface Repository persistent database and a pointer to the memory representation of the CORBA::AttributeDef object is returned to the caller.

The id, name, version, typedef, and mode attributes are set as specified. The type attribute is also set. The defined_in attribute is initialized to identify the containing CORBA::InterfaceDef.

An error is returned if an object with the specified id already exists within this object's CORBA::RepositoryId, or if an object with the specified name already exists within this CORBA::InterfaceDef.

Example

```
// C++
   // assume 'this_interface' and 'pk_long_ptr'
   // have already been initialized
   CORBA::InterfaceDef * this_interface;
   CORBA::PrimitiveDef * pk_long_ptr;

   // establish the 'create_attribute' parameters
   CORBA::RepositoryId rep_id = CORBA::string_dup ("UniqueRepositoryId");
   CORBA::Identifier name = CORBA::string_dup ("this_attribute");
   CORBA::VersionSpec version = CORBA::string_dup ("1.0");
   CORBA::AttributeMode mode = CORBA::ATTR_READONLY;

   // create the new attribute definition contained in the interface
```

```
CORBA::AttributeDef * this_attribute;
this_attribute = this_interface->create_attribute(rep_id, name, version,
                                                  pk_long_ptr, mode);
```

InterfaceDef::create_operation

The create_operation operation returns a new operation definition (CORBA::OperationDef) contained in the interface definition (CORBA::InterfaceDef) on which it is invoked.

Original interface

InterfaceDef Interface

IDL syntax

```
OperationDef create_operation (in RepositoryId id,
                               in Identifier name,
                               in VersionSpec version,
                               in IDLType result,
                               in OperationMode mode,
                               In ParDescriptionSeq params,
                               In ExceptionDefSeq exceptions,
                               in ContextIdSeq contexts);
```

Parameters

id The id represents the CORBA::RepositoryId that will uniquely identify this CORBA::OperationDef within the Interface Repository.

name The name that will be associated with this CORBA::OperationDef object in the Interface Repository.

version
The version number that will be associated with this CORBA::OperationDef object in the Interface Repository.

result_def
The result_def parameter is a CORBA::IDLType * that specifies the type of the return value for the CORBA::OperationDef.

mode Valid operation mode values include CORBA::OP_NORMAL (normal operation) and CORBA::OP_ONEWAY (one way operation).

params
The sequence defines the list of parameters that are associated with the interface.

exceptions
The sequence defined the list of exceptions that are associated with the interface.

contexts
The sequence defines the list of contexts that are associated with the interface.

Return value

OperationDef *
The returned value is a pointer to the created CORBA::OperationDef object. The memory is owned by the caller and can be released using CORBA::release.

Exceptions

CORBA::SystemException

Remarks

The create_operation operation creates a new OperationDef in the Interface Repository and returns a pointer to the memory representation of the CORBA::OperationDef object. The id, name, version, result_def, mode, params, exceptions, and contexts attributes are set as specified. The result attribute is also set based on result_def. The defined_in attribute is initialized to identify the containing CORBA::InterfaceDef.

An error is returned if an object with the specified id already exists within the Interface Repository or if an object with the specified name already exists within this CORBA::InterfaceDef.

Example

```
// C++
   // assume 'this_interface', 'this_struct', 'this_exception', and
   // assume 'this_interface', 'this_struct', 'this_exception', and
   // 'pk_long_ptr' have already been defined
   CORBA::InterfaceDef * this_interface;
   CORBA::StructDef * this_struct;
   CORBA::ExceptionDef * this_exception;
   CORBA::PrimitiveDef * pk_long_ptr;

   // establish the 'create_operation' parameters
   CORBA::RepositoryId rep_id = CORBA::string_dup ("UniqueRepositoryId");
   CORBA::Identifier name = CORBA::string_dup ("this_operation");
   CORBA::VersionSpec version = CORBA::string_dup ("1.0");
   CORBA::IDLType * result_def = this_struct;
   CORBA::OperationMode mode = CORBA::OP_NORMAL;

   CORBA::ParDescriptionSeq params;
   params.length (1);
   params[0].name = CORBA::string_dup ("parameter_0");
   params[0].type = CORBA::_tc_long;
   params[0].type_def = pk_long_ptr;
   params[0].mode = CORBA::PARAM_IN;
```

```
CORBA::ExceptionDefSeq exceptions;
exceptions.length (1);
exceptions[0] = this_exception;

CORBA::ContextIdSeq contexts;
contexts.length (1);
contexts[0] = CORBA::string_dup ("CONTEXTS_0=value_0");

// create the operation . . .
CORBA::OperationDef * this_operation;
this_operation = this_interface-> create_operation
(rep_id, name, version, result_def, mode, params, exceptions, contexts);
```

InterfaceDef::describe

The describe operation returns a structure containing information about a CORBA::InterfaceDef Interface Repository object.

Original interface

InterfaceDef Interface

IDL syntax

```
struct InterfaceDescription
   {
      Identifier name;
      RepositoryId id;
      RepositoryId defined_in;
      VersionSpec version;
      RepositoryIdSeq base_interfaces;
   };
struct Description
   {
      DefinitionKind kind;
      any value;
   };
Description describe ();
```

Parameters

None.

Return value

Description *

The returned value is a pointer to a CORBA::Contained::Description structure. The memory is owned by the caller and can be removed by invoking delete.

Exceptions

CORBA::SystemException

Remarks

The inherited describe operation returns a structure (CORBA::Contained::Description) that contains information about a CORBA::InterfaceDef Interface Repository object. The CORBA::Contained::Description structure has two fields: kind (CORBA::DefinitionKind data type), and value (CORBA::Any data type).

The kind of definition described by the returned structure is provided by invoking the kind field, and the value field is a CORBA::Any containing the description specific to the kind of object described. When the describe operation is invoked on an interface (CORBA::InterfaceDef) object, the kind field is equal to CORBA::dk_Interface and the value field contains the CORBA::InterfaceDescription structure.

Example

```
// C++
   // assume that 'this_interface' has already been initialized
   CORBA::InterfaceDef * this_interface;

   // retrieve a description of the interface
   CORBA::Contained::Description * returned_description;
   returned_description = this_interface-> describe ();

   // retrieve the interface description from the returned description
   // structure
   CORBA::InterfaceDescription * interface_description;
   interface_description = (CORBA::InterfaceDescription *)
      returned_description value.value ();
```

InterfaceDef::describe_interface

The describe_interface operation returns a CORBA::InterfaceDef::FullInterfaceDescription describing the interface (CORBA::InterfaceDef), including its operations and attributes.

Original interface

InterfaceDef Interface

IDL syntax

```
struct FullInterfaceDescription
   {
      Identifier name;
      RepositoryId id;
      RepositoryId defined_in;
      VersionSpec version;
      OpDescriptionSeq operations;
      AttrDescriptionSeq attributes;
      RepositoryIdSeq base_interfaces;
```

```
      TypeCode type;
    };
  FullInterfaceDescription describe_interface();
```

Parameters

None.

Return value

FullInterfaceDescription *

The returned value is owned by the caller and can be removed by involking delete.

Exceptions

CORBA::SystemException

Remarks

The data fields of the CORBA::InterfaceDef::FullInterfaceDescription include: name (the interface name), id (the unique CORBA::RepositoryId that identifies the interface), defined_in (the unique CORBA::RepositoryId that identifies the defined_in attribute), version (the version number), operations (a sequence listing the operations for this interface), attributes (a sequence listing the attributes of this interface), base_interfaces (a sequence of CORBA::RepositoryIds that represent the base_interfaces attribute of the interface), and type (the CORBA::TypeCode representation of the interface).

The CORBA specification contains amibiguities with relation to the describe_interface. This method returns the FullInterfaceDescription structure that contains the interface's attributes and operations. It does not state whether that includes or excludes the inherited attributes and operations. The IBM implementation excludes the inherited attributes and operations.

Example

```
// C++
   // assume that 'this_interface' has already been initialized
   CORBA::InterfaceDef * this_interface;

   // retrieve a full description of the interface
   CORBA::InterfaceDef::FullInterfaceDescription *
      returned_full_interface_description;
   returned_full_interface_description = this_interface->
      describe_interface ();
```

InterfaceDef::is_a

The is_a operation is used to determine if the target interface is identical to or inherits from another interface referenced by its unique CORBA::RepositoryId.

Original interface

InterfaceDef Interface

IDL syntax

```
boolean is_a(in RepositoryId interface_id);
```

Parameters

interface_id
The ID attribute that globally identifies a Contained object.

Return value

Boolean
The return value is the result of the evaluation of the target object and the referenced object as in the Intended Usage section.

Exceptions

CORBA::SystemException

Remarks

The is_a operation returns TRUE if the interface on which it is invoked either is identical to or inherits, directly or indirectly, from the interface identified by its interface_id parameter. Otherwise it returns FALSE. The is_a read operation parameter and result description is provided below.

Example

```
// C++
   // assume 'this_interface' and "other_interfaces_rep_id'
   // have already been initialized
   CORBA::InterfaceDef * this_interface;
   CORBA::RepositoryId other_interfaces_rep_id;

   // determine if the two objects are related
   CORBA::Boolean returned_boolean;
   returned_boolean = this_interface-> is_a (other_interfaces_rep_id);
```

IRObject Interface

The IRObject interface represents the most generic interface from which all other Interface Repository interfaces are derived, including the Repository itself.

File name

somir.idl

Intended usage

The IRObject is not itself instantiated as a means of accessing the Interface Repository. As an ancestor of all Interface Repository objects, it defines a specific operation noted below.

Local-only

True

Types

```
enum DefinitionKind
  {dk_none, dk_all, dk_Attribute, dk_Constant, dk_Exception, dk_Interface,

   dk_Module, dk_Operation, dk_Typedef, dk_Alias, dk_Struct, dk_Union,
   dk_Enum, dk_Primitive, dk_String, dk_Sequence, dk_Array, dk_Repository
  };
```

Exceptions

CORBA::SystemException

IDL syntax

```
module CORBA {
   interface IRObject {

   //read Interface
   read only attribute DefinitionKind def_kind;

   //write interface
   void destroy ();
   }
   }
```

Supported operations

IRObject::def_kind
IRObject::destroy

IRObject::def_kind

The def_kind operation returns the kind of the Interface Repository definition.

Original interface

IRObject Interface

IDL syntax

```
readonly attribute DefinitionKind def_kind;
```

Parameters

No input parameters are defined.

Return value

DefinitionKind

The returned value indicates the definition kind of the Interface Repository object. Valid values that are returned by the def_kind read operation include: CORBA::dk_Attribute, CORBA::dk_Constant, CORBA::dk_Exception, CORBA::dk_Interface, CORBA::dk_Module, CORBA::dk_Operation, CORBA::dk_Alias, CORBA::dk_Struct, CORBA::dk_Union, CORBA::dk_Enum, CORBA::dk_Primitive, CORBA::dk_String, CORBA::dk_Sequence, CORBA::dk_Array, and CORBA::dk_Repository.

Exceptions

CORBA::SystemException

Remarks

The def_kind attribute identifies the kind of the Interface Repository definition. The def_kind operation returns the value in this attribute that identifies the definition kind of the object.

Example

```
// C++
   // assume 'ir_object_ptr' has already been initialized . . .
   CORBA::IRObject * ir_object_ptr;

   // query the object to determine the definition kind . . .
   CORBA::DefinitionKind this_objects_kind;
   this_objects_kind = ir_object_ptr-> def_kind();
```

IRObject::destroy

The destroy operation causes the object to cease to exist within the Interface Repository database.

Original interface

IRObject Interface

IDL syntax

```
void destroy ();
```

Parameters

No input parameters are defined.

Return value

Void No value is returned.

Exceptions

CORBA::SystemException

Remarks

The destroy operation causes the object to cease to exist. If the object is a Container, destroy is applied to all of its contents. If the object contains an IDLType attribute for an anonymous type, that IDLType is destroyed. If the object is currently contained in some other object, it is removed from that container object. Invoking destroy on a Repository object or on a PrimitiveDef is an error.

The destroy operation causes the object to cease to exist. If the object is a Container, destroy is applied to all of its contents. If the object contains an IDLType attribute for an anonymous type, that IDLType is destroyed. If the object is currently contained in some other object, it is removed from that container object. Invoking destroy on a Repository object or on a PrimitiveDef is an error. CORBA 2.0 requires that the IR not be left in an incoherent state. After a destroy there cannot be any dangling references. The IBM implementation of destroy ensures this by deleting all objects that refer to the destroy target. When destroying an interface this will include all of it's children. Use caution.

Example

```
// C++
   // assume that 'this_module' has already been initialized
   CORBA::ModuleDef * this_module;

   // destroy the module and all that it contains
   this_module-> destroy ();
```

ModuleDef Interface

A ModuleDef can contain constants, typedefs, exceptions, interfaces, and other module objects.

File name

somir.idl

Intended usage

The ModuleDef interface is used within the Interface Repository to represent an OMG IDL module. A ModuleDef object can be created using the create_module operation defined for the Container interface.

Local-only

True

Ancestor interfaces

Contained Interface
Container Interface

Exceptions

CORBA::SystemException

IDL syntax

```
module CORBA {
   interface ModuleDef:Container,Contained {
   };
   struct ModuleDescription {
   Identifier name;
   RepositoryId id;
   RepositoryId defined_in;
   VersionSpec version;
   }
   }
```

Supported operations

ModuleDef::describe

ModuleDef::describe

The describe operation returns a structure containing information about a CORBA::ModuleDef Interface Repository object.

Original interface

ModuleDef Interface

IDL syntax

```
struct ModuleDescription {
   Identifier name;
   RepositoryId id;
   RepositoryId defined_in;
   VersionSpec version;
   };
   struct Description {
   DefinitionKind kind;
   any value;
   };
   Description describe ();
```

Parameters

No input parameters are defined.

Return value

Description *

The returned value is a pointer to a CORBA::Contained::Description structure. The memory is owned by the caller and can be removed using delete.

Exceptions

CORBA::SystemException

Remarks

The inherited describe operation returns a structure (CORBA::Contained::Description) that contains information about a CORBA::ModuleDef Interface Repository object. The CORBA::Contained::Description structure has two fields: kind (CORBA::DefinitionKind data type), and value (CORBA::Any data type).

The kind of definition described by the returned structure is provided using the kind field, and the value field is a CORBA::Any that contains the description that is specific to the kind of object described. When the describe operation is invoked on a module (CORBA::ModuleDef) object, the kind field is equal to CORBA::dk_Module and the value field contains the CORBA::ModuleDescription structure.

Example

```
// C++
   // assume that 'this_module' has already been initialized
   CORBA::ModuleDef * this_module;

   // retrieve a description of the module
   CORBA::Contained::Description * returned_description;
   returned_description = this_module-> describe ();
   // retrieve the module description from the returned description structure
   CORBA::ModuleDescription * module_description;
   module_description = (CORBA::ModuleDescription * ) returned_description
                        value.value ();
```

NamedValue Class

Represents a request parameter, request return value, or Context property.

File name

nvlist.h

Intended usage

A Dynamic Invocation Interface request is comprised of an object reference, an operation, a list of arguments for the operation, and a return value. A NamedValue is used to represent each element of the argument list and the return value. A NamedValue is also used to represent each element of the property list associated with a Context. The ORB::create_named_value method is used to create an empty NamedValue. The NVList class provides methods to manage a list of named values. For additional information, see the NVList, Request, Context, and ORB class descriptions.

Supported methods

NamedValue::_duplicate
NamedValue::_nil
NamedValue::flags
NamedValue::name
NamedValue::value

NamedValue::_duplicate

Duplicates a NamedValue object.

Original class

CORBA::NamedValue

IDL syntax

```
static CORBA::NamedValue_ptr _duplicate (CORBA::NamedValue_ptr p);
```

Parameters

p The NamedValue object to be duplicated. The reference can be nil, in which case the return value will also be nil.

Return value

CORBA::NamedValue_ptr
The new NamedValue object reference. This value should subsequently be released using CORBA::release(NamedValue_ptr).

Remarks

This method is intended to be used by client and server applications to duplicate a reference to a NamedValue object. Both the original and the duplicate reference should subsequently be released using CORBA::release(NamedValue_ptr).

NamedValue::_nil

Returns a nil CORBA::NamedValue reference.

Original class

CORBA::NamedValue

IDL syntax

```
static CORBA::NamedValue_ptr _nil ();
```

Return value

CORBA::NamedValue_ptr
A nil NamedValue reference.

Remarks

This method is intended to be used by client and server applications to create a nil NamedValue reference.

NamedValue::flags

Returns a bitmask that identifies the argument passing mode.

Original class

CORBA::NamedValue

IDL syntax

```
CORBA::Flags flags() const;
```

Return value

CORBA::Flags
The argument passing mode. If the flags method is called on a NamedValue which does not represent a request parameter, an empty bitmask is returned.

Remarks

The flags method is used by a client program when querying a NamedValue representing a parameter of a DII request. The flags method returns a bitmask that identifies the argument passing mode. The following flag values are defined:

CORBA::ARG_IN
The associated value is an input only argument.

CORBA::ARG_OUT
The associated value is an output only argument.

CORBA::ARG_INOUT
The associated value is an in/out argument.

NamedValue::name

Returns the argument name.

Original class

CORBA::NamedValue

IDL syntax

```
const char *name() const;
```

Return value

const char *
The argument name, if any, or a null pointer. Ownership of the return value is maintained by the NamedValue; the return value must not be freed by the caller.

Remarks

The name method is used by a client program when querying a NamedValue associated with a DII request. The name method returns the argument name, which is optional. The argument name in a NamedValue, if present, matches the argument name specified in the IDL definition of the operation.

NamedValue::value

Returns the argument value.

Original class

CORBA::NamedValue

IDL syntax

```
CORBA::Any *value() const;
```

Return value

CORBA::Any *

A pointer to the argument value, if any, or a null pointer. Ownership of the return value is maintained by the NamedValue; the return value must not be freed by the caller.

Remarks

The value method is used by a client program when querying a NamedValue associated with a DII request. The value method returns the argument value, which is accessed using standard operations on the Any class.

NVList Class

Specifies a list of arguments: parameters associated with a request or properties associated with a Context.

File name

nvlist

Intended usage

A Dynamic Invocation Interface request is comprised of an object reference, an operation, a list of arguments for the operation, and a return value. An NVList is used to specify the list of arguments for the operation. An NVList is also used to specify the list of properties associated with a Context. The ORB::create_list method is called to create an empty named value list. The ORB::create_operation_list method is called to create a named value list for a specific operation. The NVList class provides methods to add and delete a named value, as well as query information about a named value list. For additional information, see the NamedValue, Request, Context, and ORB class descriptions.

Supported methods

NVList::_duplicate
NVList::_nil
NVList::add
NVList::add_item
NVList::add_item_consume
NVList::add_value
NVList::add_value_consume
NVList::count
NVList::get_item_index
NVList::item
NVList::remove

NVList::_duplicate

Duplicates an NVList object.

Original class

CORBA::NVList

IDL syntax

```
static CORBA::NVList_ptr _duplicate (CORBA::NVList_ptr p);
```

Parameters

p The NVList object to be duplicated. The reference can be nil, in which case the return value will also be nil.

Return value

CORBA::NVList_ptr
The new NVList object reference. This value should subsequently be released using CORBA::release(NVList_ptr).

Remarks

This method is intended to be used by client and server applications to duplicate a reference to an NVList object. Both the original and the duplicate reference should subsequently be released using CORBA::release(NVList_ptr).

NVList::_nil

Returns a nil CORBA::NVList reference.

Original class

CORBA::NVList

IDL syntax

```
static CORBA::NVList_ptr _nil ();
```

Return value

CORBA::NVList_ptr
A nil NVList reference.

Remarks

This method is intended to be used by client and server applications to create a nil NVList reference.

NVList::add

Adds an element to the end of a named value list.

Original class

CORBA::NVList

IDL syntax

```
CORBA::NamedValue_ptr add(CORBA::Flags flags);
```

Parameters

flags A bitmask describing the argument. The following standard flag values identify the argument passing mode:

CORBA::ARG_IN
The associated value is an input-only argument.

CORBA::ARG_OUT
The associated value is an output-only argument.

CORBA::ARG_INOUT
The associated value is an in/out argument.

Return value

CORBA::NamedValue_ptr
A pointer to the newly created named value. Ownership of the return value is maintained by the NVList; the return value must not be freed by the caller.

Remarks

The add method is used by a client program to populate the NVList associated with a DII request. The add method adds an element to the end of a named value list. The newly created named value is empty, except for the flags. See also the add_item, add_item_consume, add_value, and add_value_consume methods, which perform the same task but differ in memory management and how the newly created named value is initialized.

NVList::add_item

Adds an element to the end of a named value list.

Original class

CORBA::NVList

IDL syntax

```
CORBA::NamedValue_ptr add_item(const char *id, CORBA::Flags flags);
```

Parameters

flags A bitmask describing the argument. The following standard flag values identify the argument passing mode:

CORBA::ARG_IN
The associated value is an input-only argument.

CORBA::ARG_OUT
The associated value is an output-only argument.

CORBA::ARG_INOUT
The associated value is an in/out argument.

Return value

CORBA::NamedValue_ptr
A pointer to the newly created named value. Ownership of the return value is maintained by the NVList; the return value must not be freed by the caller.

Remarks

The add_item method is used by a client program to populate the NVList associated with a DII request. The add_item method adds an element to the end of a named value list. The newly created named value is initialized using the input argument name and flags. The difference between the add_item and add_item_consume methods is that the former does not assume ownership of the input argument name, while the latter does. See also the add, add_value, and add_value_consume methods.

NVList::add_item_consume

Adds an element to the end of a named value list

Original class

CORBA::NVList

IDL syntax

```
CORBA::NamedValue_ptr add_item_consume(char *id, CORBA::Flags flags);
```

Parameters

id The name of the argument to be added. It is legal to pass a null pointer. If

specified, the input name should match the argument name specified in the IDL definition for the operation. The argument name must be allocated using the CORBA::string_alloc method. Ownership of this parameter transfers to the NVList.

flags A bitmask describing the argument. The following standard flag values identify the argument passing mode:

CORBA::ARG_IN
The associated value is an input only argument.

CORBA::ARG_OUT
The associated value is an output only argument.

CORBA::ARG_INOUT
The associated value is an in/out argument.

Return value

CORBA::NamedValue_ptr
A pointer to the newly created named value. Ownership of the return value is maintained by the NVList; the return value must not be freed by the caller.

Remarks

The add_item_consume method is used by a client program to populate the NVList object associated with a DII request. The add_item_consume method adds an element to the end of a named value list. The newly created named value is initialized using the input argument name and flags. The difference between the add_item and add_item_consume methods is that the former does not assume ownership of the input argument name, while the latter does. The caller may not access the memory referred to by the input parameter after it has been passed in. See also the add, add_value, and add_value_consume methods.

NVList::add_value

Adds an element to the end of a named value list.

Original class

CORBA::NVList

IDL syntax

```
CORBA::NamedValue_ptr add_value(const char *id,
                                const CORBA::Any &any,
                                CORBA::Flags flags);
```

Parameters

id The name of the argument to be added. It is legal to pass a null pointer. If specified, the input name should match the argument name specified in the IDL definition for the operation.

any The address of the value of the argument. It is legal to pass a null pointer.

flags A bitmask describing the argument. The following standard flag values identify the argument passing mode:

CORBA::ARG_IN
The associated value is an input only argument.

CORBA::ARG_OUT
The associated value is an output only argument.

CORBA::ARG_INOUT
The associated value is an in/out argument.

Return value

CORBA::NamedValue_ptr
A pointer to the newly created named value. Ownership of the return value is maintained by the NVList; the return value must not be freed by the caller.

Remarks

The add_value method is used by a client program to populate the NVList associated with a DII request. The add_value method adds an element to the end of a named value list. The newly created named value is initialized using the input argument name, value, and flags. The difference between the add_value and add_value_consume methods is that the former does not assume ownership of the input argument name and value, while the latter does.

See also NVList::add, NVList::add_item, and NVList::add_item_consume.

NVList::add_value_consume

Adds an element to the end of a named value list.

Original class

CORBA::NVList

IDL syntax

```
CORBA::NamedValue_ptr add_value_consume(char *id,
                                        CORBA::Any_ptr any,
                                        CORBA::Flags flags);
```

Parameters

id The name of the argument to be added. It is legal to pass a null pointer. If specified, the input name should match the argument name specified in the IDL definition for the operation. Ownership of this parameter transfers to the NVList.

any The address of the value of the argument. It is legal to pass a null pointer. Ownership of this parameter transfers to the NVList.

flags A bitmask describing the argument. The following standard flag values identify the argument passing mode:

CORBA::ARG_IN
The associated value is an input only argument.

CORBA::ARG_OUT
The associated value is an output only argument.

CORBA::ARG_INOUT
The associated value is an in/out argument.

Return value

CORBA::NamedValue_ptr
A pointer to the newly created named value. Ownership of the return value is maintained by the NVList; the return value must not be freed by the caller.

Remarks

The add_value_consume method is used by a client program to populate the NVList associated with a DII request. The add_value_consume method adds an element to the end of a named value list. The newly created named value is initialized using the input argument name, value, and flags. The difference between the add_value and add_value_consume methods is that the former does not assume ownership of the input argument name and value, while the latter does. The caller may not access the memory referred to by the input parameters after they have been passed in.

See also NVList::add, NVList::add_item, and NVList::add_item_consume.

NVList::count

Returns the number of elements in a named value list.

Original class

CORBA::NVList

IDL syntax

```
CORBA::ULong count() const;
```

Return value

CORBA::ULong
The number of elements in the named value list.

Remarks

The count method is used by a client program when querying the NVList associated with a DII request. The count method reports the number of elements in a named value list.

NVList::get_item_index

Returns the index of the specified named value.

Original class

CORBA::NVList

IDL syntax

```
CORBA::Long get_item_index(const char *id);
```

Parameters

id The argument name of the desired named value. A system exception is raised if a null pointer is passed for this parameter.

Return value

CORBA::Long
The index of the specified named value. If the named value list is empty or the input argument name is not found, -1 is returned.

Exceptions

CORBA::SystemException

Remarks

The get_item_index method is used by a client program when querying the NVList associated with a DII requst. The get_item_index method returns the index of the specified named value. The argument name comparison is case insensitive. This method is an IBM extension to the CORBA 2.0 specification.

NVList::item

Returns the named value associated with the input index.

Original class

CORBA::NVList

IDL syntax

```
CORBA::NamedValue_ptr item(CORBA::ULong index);
```

Parameters

index The index of the desired named value, starting at zero. A system exception is raised if the input index is greater than or equal to the number of elements in the named value list.

Return value

CORBA::NamedValue_ptr
A pointer to the named value associated with the input index. Ownership of the return value is maintained by the NamedValue; the return value must not be freed by the caller.

Exceptions

CORBA::SystemException

Remarks

The item method is used by a client program when querying the NVList associated with a DII request. The item method returns the named value associated with the input index.

NVList::remove

Deletes the named value associated with the input index.

Original class

CORBA::NVList

IDL syntax

```
CORBA::Status remove(CORBA::Ulong index);
```

Parameters

index The index of the named value to be deleted, starting at zero. A system exception is raised if the input index is greater than or equal to the number of elements in the named value list.

Return value

CORBA::Status

A zero return code indicates the named value was successfully deleted.

Exceptions

CORBA::SystemException

Remarks

The remove method is used by a client program when freeing an element of the NVList associated with a DII request. The remove method deletes the named value associated with the input index. CORBA::release(NamedValue_ptr) is called on the named value element. The remaining named value elements are re-indexed.

Object Class

Provides behavior common to all object references (both local objects and proxies to remote objects).

File name

object.h

Intended usage

The CORBA::Object class is the abstract base class for all object references. This includes all proxy classes (objects, residing in client processes, that refer to remote objects residing in a server) and all classes that implement IDL interfaces to be exported from a server. As such, the CORBA::Object class provides methods that are meaningful to both local objects and (references to) remote objects.

Constants

```
static const char* Object_CN;
```

Supported methods

Object::_create_request
Object::_duplicate
Object::_get_implementation
Object::_get_interface
Object::_hash
Object::_is_a
Object::_is_equivalent
Object::_narrow
Object::_nil
Object::_non_existent
Object::_request
Object::_this

Object::_create_request

Creates a Request object suitable for invoking a specific operation using the Dynamic Invocation Interface (DII).

Original class

CORBA::Object

IDL syntax

```
virtual CORBA::Status _create_request (CORBA::Context_ptr ctx,
                                       const char* operation,
                                       CORBA::NVList_ptr arg_list,
                                       CORBA::NamedValue_ptr result,
                                       CORBA::Request_ptr &request,
                                       CORBA::Flags reg_flags) = 0;
virtual CORBA::Status _create_request (CORBA::Context_ptr ctx,
                                       const char* operation,
                                       CORBA::NVList_ptr arg_list,
                                       CORBA::NamedValue_ptr result,
                                       CORBA::ExceptionList_ptr exc_list,
                                       CORBA::ContextList_ptr ctx_list,
                                       CORBA::Request_ptr &request,
                                       CORBA::Flags reg_flags) = 0;
```

Parameters

ctx A pointer to the CORBA::Context object to be passed when the DII request is invoked. For operations having no `context` clause in their IDL specification, this can be NULL. The CORBA::Request object assumes ownership of this parameter.

operation
The unscoped name of the IDL operation to be invoked using the Request. This must be an operation that is implemented or inherited by the CORBA::Object on which CORBA::Object::_create_request() is invoked. The caller retains ownership of this parameter (the CORBA::Request object makes a copy).

arg_list
A pointer to a CORBA::NVList object that describes;
- The types of all the operation parameters of the IDL.
- The values of the `in` and `inout` parameters of the operation.
- The variables in which the `out` parameter values will be stored after the DII request is invoked.

result A pointer to a CORBA::NamedValue object that will hold the result of the DII request after it is invoked. The CORBA::Request object assumes ownership of this parameter.

request
A CORBA::Request_ptr variable, passed by reference, to be initialized by

CORBA::Object::_create_request() to point to the newly-created CORBA::Request object. The caller retains ownership of this parameter.

req_flags
A bit-vector describing how the DII request will be invoked. Oneway methods (methods that do not require a response) should be created using a *req_flags* value of `CORBA::INV_NO_RESPONSE`. No other *req_flags* values are currently used.

exc_list
A pointer to a CORBA::ExceptionList object that describes the user-defined exceptions that the DII operation can throw (according to the operation declaration in the IDL specification). This parameter is essentially a list of TypeCodes for UserException subclasses. The CORBA::Request object assumes ownership of this parameter. This parameter is optional and NULL can be passed (even for methods that raise user-defined exceptions).

ctx_list
A pointer to a CORBA::ContextList object that lists the Context strings that must be sent with the DII operation (according to the operation declarations in the IDL specification). This parameter differs from the *ctx* parameter in that this parameter supplies only the context strings whose values are to be transmitted with the DII request, while the *ctx* parameter is the object from which those context string values are obtained. The CORBA::Request object assumes ownership of this parameter. This parameter is optional and NULL can be passed (even for methods that pass Context parameters).

Return value

CORBA::Status
A return value of zero indicates success.

Exceptions

CORBA::SystemException

Remarks

The _create_request method (two forms) are used to create CORBA::Request objects tailored to a specific IDL operation. The CORBA::Request object can then be used to invoke requests using the DII. After invoking the method, an application can obtain the return result, output parameter values, and exceptions using methods on CORBA::Request.

The target of CORBA::Object::_create_request() is typically a proxy object, rather than a local object. When invoked on a proxy object, this method operates locally; the remote object to which the proxy refers is unaffected until the DII request that is created by CORBA::Object::_create_request() is invoked.

The two forms of CORBA::Object::_create_request() differ in whether a CORBA::ExceptionList_ptr and a CORBA::ContextList_ptr are provided as input. These input parameters are not needed for operations that have no `raises` or `context` clause

in the IDL specification. For IDL operations that do have a `raises` or `context` clause, the second form of CORBA::Object::_create_request() can be used to avoid (potentially time-consuming) Interface Repository lookups by the ORB when the DII request is invoked.

See also, Object::_request, which creates a CORBA::Request without providing the parameters for the operation.

Example

```
/* The following IDL signature is used:
   interface testObject
   {
      string testMethod(in long input_value, out float outvalue);
   };
 */
...
/* Get the OperationDef that describes testMethod */
CORBA::ORB_var myorb =
   CORBA::ORB_init(argc, argv, "DSOM");  /* argc, argv: input arguments */
CORBA::Repository_var my_IR = CORBA::Repository::_narrow(generic_IR);
CORBA::Contained_var generic_opdef =
   my_IR -> lookup("testObject::testMethod");
CORBA::OperationDef_var my_opdef =
   CORBA::OperationDef::_narrow(generic_opdef);

/* Create the NVList and NamedValue for the request */
CORBA::NVList_ptr params = NULL;
myorb -> create_operation_list(my_opdef, params);
CORBA::NamedValue_ptr result = NULL;
myorb -> create_named_value(result);

/* Create the Request object */
CORBA::Object_var my_proxy =   /* Get a proxy somehow */
   my_proxy -> _create_request(NULL, "testMethod", params, result,
      my_request, 0);
```

Object::_duplicate

Duplicates an object reference

Original class

CORBA::Object

IDL syntax

```
static CORBA::Object_ptr _duplicate (CORBA::Object_ptr obj);
```

Parameters

obj The object reference to be duplicated., If this parameter is a nil object reference (NULL), no action is taken.

Return value

CORBA::Object_ptr

The duplicate of the input object reference. (Because CORBA::Object::_duplicate and CORBA::release are implemented using reference couting, this will be the same as the input value.) If the input value is a nil reference, the return value will likewis e be nil.

Remarks

This method is intended to be used by client and server applications, to duplicate object references (both pointers to local implementation objects and proxies to remote objects). For each duplication performed on an object reference, an equal number of alls to CORBA::release must also be made for the reference to be deleted.

When an application passes an object reference (either a local object or a proxy) on a method call, either as a parameter value or a return result, if the call transfers ownership of the object reference and the application needs to retain ownership of the reference as well, the application should first duplicate the reference before passing it. Each user of the reference should subsequently CORBA::release the reference so that its resources can be reclaimed.

When CORBA::Object::_duplicate is called on a proxy object, only the proxy is affected; no remote invocation is made to the remote object to which the proxy refers. Hence, CORBA::Object::_duplicate and CORBA::release are only used to manage the local resources associated with object references.

Example

```
/*The following example is written in C++*/
#include "corba.h"

/* this function returns duplicate of an object ref */
::CORBA::Object_ptr getObj(::CORBA::Object_ptr p)
{
  return CORBA::Object::_duplicate(p);
}
```

Object::_get_implementation

Returns a reference to the CORBA::ImplementationDef describing the server in which a remote object resides.

Original class

CORBA::Object

IDL syntax

```
virtual CORBA::ImplementationDef_ptr _get_implementation () = 0;
```

Return value

CORBA::ImplementationDef_ptr

A pointer to the ImplementationDef object that describes the server in which an object (refererred to by an object reference) resides. The caller assumes ownership of this object, and should subsequently CORBA::release (not delete) it.

Exceptions

CORBA::SystemException

Remarks

This method is intended to be used to obtain the CORBA::ImplementationDef object describing the server in which a remote object resides. When invoked on a proxy object, this method is forwarded to the remote object, and a proxy to a remote CORBA::ImplementatinDef object (residing in the same server as the remote object) is returned. When invoked on a local object residing in a server, the local CORBA::ImplementationDef object (the one originally passed to CORBA::BOA::impl_is_ready) is returned. When invoked on a local object in a client (that is not also a server), NULL is returned.

Example

```
/* The following is a C++ example */
#include "corba.h"
#include <string.h>
/* Assume p is a proxy object pointer derived from CORBA::Object class
   the following will get the impl def and interface def on remote objects
 */
CORBA::ImplementationDef_ptr impl;
CORBA::InterfaceDef_ptr intf;
string str;
impl = p->_get_implementation();
if(impl)
{
  str = impl->get_alias();            /* get implementation alias */
  /* ensure it's the right impl and work with the impl */
  ...
}
else /* generate exception */ ...
  intf = p->_get_interface();
  if(intf)
  {
```

```
    str = intf->id();                /* get interface id */
    /* ensure it's the right interface and work with the intf */
     ...
  }
else /* generate exception */ ...
CORBA::release(p);
```

Object::_get_interface

Returns a reference to the CORBA::InterfaceDef describing the most specific interface supported by the target object.

Original class

CORBA::Object

IDL syntax

```
virtual CORBA::InterfaceDef_ptr _get_interface () = 0;
```

Return value

CORBA::InterfaceDef_ptr
A pointer to the InterfaceDef object that describes the most specific interface supported by the target object . The caller assumes ownership of this object.

Original class

CORBA::SystemException

Remarks

This method is intended to be used to obtain the CORBA::InterfaceDef object describing the most specific interface of the target object. When invoked on a proxy object, this method is forwarded to the remote object, and a proxy to a remote CORBA::InterfaceDef object (residing in the same server as the remote object) is returned. This InterfaceDef describes the interface of the remote object, which may be more specific than the interface of the proxy on which CORBA::_get_interface was invoked. (This can occur when the client does not have bindings for the most specific interface supported by the remote object.)

When invoked on a local object, a local CORBA::InterfaceDef object is retrieved from the local Interface Repository.

Example

See the CORBA::“Object::_get_implementation” on page 160.

Object::_hash

Maps object references into disjointed groups of potentially equilvalent references.

Original class

CORBA::Object

IDL syntax

```
virtual CORBA::ULong _hash (CORBA::ULong maximum) = 0;
```

Parameters

maximum
The upper bound on the return value.

Return value

CORBA::ULong
A hash value with a lower bound of zero and an upper bound as indicated by the maximum parameter.

Remarks

This method is intended to be used by applications that manipulate large numbers of object references, for mapping object references into disjoint groups of potentially equivalent references. The hash value of an object reference does not change during the lifetime of the reference. The hash value of an object reference is not necessarily unique (another reference may have the same hash value). Different object references to the same remote object do not necessarily hash to the same value.

When invoked on a proxy object, this method does not result in a remote request to the server; all processing is done locally.

Example

```
/*The following example is written in C++*/
#include "corba.h"
#define HASH_MAX 10000
/* assume p is CORBA::Object pointer */
::CORBA::ULong hash_ulong = 0;
hash_ulong = p->_hash(HASH_MAX);
...
```

Object::_is_a

Determines whether an object supports a given IDL interface.

Original class

CORBA::Object

IDL syntax

```
virtual CORBA::Boolean _is_a (const char* logical_type_id) = 0;
```

Parameters

logical_type_id

The Interface Repository type identifier of an IDL interface. This is not simply the interface name. For programmer convenience, type identifers are provided by the C++ bindings, as static consts of the C++ class corresponding to the interface, using the naming convention <interface-name>::<interface-name>_CN. If this parameter value is NULL or not a valid Interface Repository type identifier, zero is returned.

Return value

CORBA::Boolean

A zero return value indicates that the object referenced by the object reference does not support the specified IDL interface. A nonzero return value indicates that it does support it. An object is considered to support the interface if it either implements it or inherits it.

Exceptions

CORBA::SystemException

Remarks

This method is intended to be used by applications to determine whether an object reference refers to an object that supports a given IDL interface (and hence whether the reference can be successfully narrowed). When invoked on a proxy object, this call sometimes results in a remote invocation (if it cannot be determined locally).

Example

```
/* Assume the following idl interface: */
interface testObject {
string testMethod (in long input_value, out float out_value);
};

/* Here is the cpp code: */
CORBA::Object_ptr test_obj;
/* initialize test_obj somehow */
...
/* To find out if test_obj can be narrowed to testObject, use
   CORBA::Object::_is_a and the Repository ID for the testObject interface
```

```
   (defined in the emitted bindings as testObject::testObject_RID) */
if (test_obj->_is_a(testObject::testObject_RID))
testObject_ptr new_test_obj = testObject::_narrow(test_obj);
...
```

Object::_is_equivalent

Determines whether two object references refer to the same object.

Original class

CORBA::Object

IDL syntax

```
virtual CORBA::Boolean _is_equivalent (const
   CORBA::Object_ptr other_object) = 0;
```

Parameters

other_object
An object reference to be compared to the target object reference.

Return value

CORBA::Boolean
A zero return value indicates that the target object reference does not refer to the same object as the given object reference, as far as the ORB can easily determine. (It is still possible that the two object references are equivalent, however.) A non-zero return value indicates that the target object reference and the given object reference do refer to the same object.

Remarks

This method is intended to be used by applications to determine whether two object references refer to the same object, as far as the ORB can easily determine. In the case of proxies, this method attempts to determine whether two proxies refer to the same remote object. When invoked on proxy objects, this method operates locally and does not involve the remote object to which the proxy refers. For this reason, it is possible for this method to return zero, indicating that the two references do not appear to be equivalent, when in fact they are equivalent (but it cannot be determined without communicating with the remote server).

Example

```
/*The following example is written in C++*/
#include "corba.h"
CORBA::Object_ptr p1;
CORBA::Object_ptr p2;
/*construct p1 and p2 */
```

```
...
/* check to see if they are different objects */
CORBA::Boolean retval = p1-> _is_equivalent(p2);
```

Object::_narrow

Performs essentially the same function as CORBA::Object::_duplicate().

Original class

CORBA::Object

IDL syntax

```
static CORBA::Object_ptr _narrow (CORBA::Object_ptr obj);
```

Parameters

obj The CORBA::Object to be narrowed.The caller retains ownership of this object reference.

Return value

CORBA::Object_ptr
The narrowed (and duplicated) object reference. The caller assumes ownership of this object reference and should subsequently CORBA::release it.

Exceptions

CORBA::SystemException

Remarks

This method is provided for consistency with the _narrow methods provided by the C++ bindings for subclasses of CORBA::Object, which narrow a generic CORBA::Object to a more specific type. When narrowing from a CORBA::Object to a CORBA::Object, however, the method degenerates to a simple duplication. Hence, this method is equivalent to CORBA::Object::_duplicate.

Example

```
/* Assume the following idl interface: */
interface testObject
{
  string testMethod (in long input_value, out float out_value);
};
/* Here is the cpp code: */
CORBA::Object_ptr optr;
```

```
/* instantiate optr somehow */
...
testObject_ptr test_obj = testObject::_narrow(optr);
...
```

Object::_nil

Returns a nil CORBA::Object reference.

Original class

CORBA::Object

IDL syntax

```
static CORBA::Object_ptr _nil ();
```

Return value

CORBA::Object_ptr
A nil Object reference.

Remarks

This method is intended to be used by client and server applications to create a nil Object reference. Since a nil value proxy object may be generated and returned by this call (versus a NULL), nil references can and should be released when no longer required by the client application. Due to this ″variable″ returned value, client and server applications should be using the CORBA::is_nil() method for checking for nil references (instead of checking against NULL).

Example

```
/* Assume the following IDL interface */
interface testObject
{
  string testMethod ( in long input_value, out float out_value);
};
/* Here is the cpp code */
testObject_ptr test_obj = testObject::_nil();
...
```

Object::_non_existent

Determines whether an object reference regers to a valid object.

Original class

CORBA::Object

IDL syntax

```
virtual Boolean _non_existent () = 0;
```

Return value

CORBA::Boolean

A zero return value indicates that the target object reference refers to a valid object. A nonzero return value indicates that the target object reference refers to a non-existent object.

Remarks

This method is intended to be used to determine whether an object reference (either a proxy object or a local pointer) refers to a valid object. When invoked on a proxy object, this results in a remote call to the server, which may activate the remote object to determine its existence, but no method is invoked on the remote object itself (unless the server invokes some method on the object as part of activation).

Example

```
/* Assume the following IDL interface */
interface testObject
{
  string testMethod (in long input_value, out float out_value);
};
/* Here is the cpp code */
testObject_ptr test_obj;
/* instantiate test_obj and make it a proxy somehow */
...
CORBA::Boolean retval = test_obj->_non_existent();
...
```

Object::_request

Creates a Request object suitable for invoking a specific operation using the Dynamic Invocation Interface (DII).

Original class

CORBA::Object

IDL syntax

```
virtual CORBA::Request_ptr _request (const char* operation) = 0;
```

Parameters

operation

The unscoped name of the IDL operation to be invoked using the Request. This must be an operation that is implemented or inherited by the

CORBA::Object on which CORBA::Object::_create_request is invoked. The caller retains ownership of this parameter (the CORBA::Request object makes a copy).

Return value

CORBA::Request_ptr
A pointer to the newly-created CORBA::Request object. The caller assumes ownership of this object and should subsequently delete it.

Exceptions

CORBA::SystemException

Remarks

The _request method is used to create a CORBA::Request object tailored to a specific IDL operation. Arguments, the operation's return type, and context identifiers should be added after construction via methods on CORBA::Request. The CORBA::Request object can then be used to invoke requests using the Dynamic Invocation Interface (DII). After invoking the method, an application can obtain the return results, output parameter values, and exceptions via methods on CORBA::Request.

The target of the CORBA::Object::_request method is typically a proxy object, rather than a local object. When invoked on a proxy object, this method operates locally; the remote object to which the proxy refers is unaffected until the DII Request that is created by CORBA::Object::_request is invoked.

This mechanism for creating a CORBA::Request assumes that the operation to be invoked via the DII is not ″oneway″ (that a response is required).

See also CORBA::Object::"Object::_create_request" on page 157, which allows the CORBA::Request to be created and fully initialized at once.

Example

```
/* The following C++ fragment creates a request object assuming
   that p is a proxy object pointer already declared and defined
 */
#include "corba.h"
CORBA::Request_ptr req = p->_request("dii_string_tst";
CORBA::String str = CORBA::string_alloc(12+1);
strcpy( str, "Input String");
req->add_in_arg() <<= str;
req->set_return_type(CORBA::_tc_string);
req->invoke();
...
CORBA::string_free(str);
CORBA::release(req);
```

Object::_this

Returns a duplicated object reference for the object implementation on which this operation was invoked.

Original class

CORBA::Object

IDL syntax

```
CORBA::Object_ptr _this();
```

Return value

CORBA::Object_ptr
A duplicate of the object reference on which CORBA::Object::_this was invoked. The caller assumes ownership of this object reference and should subsequently either CORBA::release it or transfer ownership of it to another party.

Remarks

This method is intended to be used within an implementation of an IDL interface, to obtain a duplicate of the object on which an operation was invoked. Calling _this() within an IDL operation implementation is not equivalent to calling _duplicate(this), because an object reference is not necessarily the object itself. To be CORBA compliant, an implementation should use _this() instead of _duplicate(this). In addition, even though _this() is implemented today as a non-virtual method on CORBA::Object, and on all the C++ interface classes generated for each interface, an implementation may not assume that it will always be implemented in this way. It may only assume that ″_this()″ is available within the scope of the implementation, and will always return the correct object reference for the interface that corresponds to the implementation.

_this() may not be used by a client. A client who already holds an object reference may use 'InterfaceName'::_narrow(objref) to obtain an object reference to a more derived interface whose name is 'InterfaceName'. It can also rely on automatic C++ conversion to obtain an object reference to a parent interface.

Example

```
/* Assume the following IDL interface */
interface testObject
{
  testObject testMethod ( );
};

/* Here is the cpp code that might appear in
   an implementation of testObject::testMethod
 */
testObject_ptr MyImplementation::testMethod()
```

```
{
  return _this();      /* duplicates and returns self */
}
...
```

ORB Class

Provides basic Object Request Broker services.

File name

orb.h

Intended usage

The ORB class is intended to be used by client and server applications to access basic Object Request Broker (ORB) services, as described by the CORBA specification. One instance of the ORB class exists in each client or server process at all times. An application typically accesses the ORB object using the CORBA::ORB_init function. The ORB provides methods for converting between object references (e.g., proxies) and strings, methods used to support the Dynamic Invocation Interface (DII), and initialization methods that list and retrieve references to the Naming Service, the Interface Repository, and the Basic Object Adapter (BOA).

Nested classes

RequestSeq

Types

```
typedef char* OAid;
typedef char* ObjectId;
typedef _IDL_SEQUENCE_String ObjectIdList;
```

Constants

```
static const char* ex_InvalidName;
```

Exceptions

```
class InvalidName : public UserException
{
  public:
  static const char* exception_id;
  InvalidName () : UserException (ex_InvalidName) {}
  static InvalidName* _narrow (Exception *exception);
};
```

Supported methods

ORB::_duplicate
ORB::_nil
ORB::BOA_init
ORB::create_alias_tc

ORB::create_array_tc
ORB::create_context_list
ORB::create_enum_tc
ORB::create_environment
ORB::create_exception_list
ORB::create_exception_tc
ORB::create_interface_tc
ORB::create_list
ORB::create_named_value
ORB::create_operation_list
ORB::create_recursive_sequence_tc
ORB::create_sequence_tc
ORB::create_string_tc
ORB::create_struct_tc
ORB::create_union_tc
ORB::get_default_context
ORB::get_next_response
ORB::get_service_information
ORB::list_initial_services
ORB::object_to_string
ORB::poll_next_response
ORB::resolve_initial_references
ORB::resolve_initial_references_remote
ORB::send_multiple_requests_deferred
ORB::send_multiple_requests_oneway
ORB::string_to_object

ORB::_duplicate

Duplicates an ORB object.

Original class

CORBA::ORB

IDL syntax

```
static CORBA::ORB_ptr _duplicate (CORBA::ORB_ptr p);
```

Parameters

p The ORB object to be duplicated. The reference can be nil, in which case the return value will also be nil.

Return value

CORBA::ORB_ptr
The new ORB object reference. This value should subsequently be released using CORBA::release(ORB_ptr).

Remarks

This method is intended to be used by client and server applications to duplicate a reference to an ORB object. Both the original and the duplicate reference should subsequently be released using CORBA::release(ORB_ptr).

Example

```
/* For illustrative purposes, the following program
   duplicates the orb pointer */
#include "corba.h"
int main(int argc, char* argv[])
{
  int rc = 0;
  CORBA::ORB_ptr cop = CORBA::ORB_init(argc, argv, "DSOM");
  CORBA::ORB_ptr dup_cop = CORBA::ORB::_duplicate(cop);
  return rc;
}
```

ORB::_nil

Returns a nil CORBA::ORB reference.

Original class

CORBA::ORB

IDL syntax

```
static CORBA::ORB_ptr _nil ();
```

Return value

CORBA::ORB_ptr
A nil ORB reference.

Remarks

This method is intended to be used by client and server applications to create a nil ORB reference.

Example

See the example in the Object::_nil method

ORB::BOA_init

Initializes and returns a pointer to the Basic Object Adapter (BOA) in a server.

Original class

CORBA::ORB

IDL syntax

```
CORBA::BOA_ptr BOA_init (int& argc,
                         char** argv,
                         const CORBA::ORB::OAid boa_identifier);
```

Parameters

argc The number of strings in the argv array of strings. This is typically the *argc* parameter passed in to the server's main() function.

argv An array of strings, whose size is indicated by the argc parameter. This is typically the *argv* parameter passed in to the server's main() function. [**Workstation Implementation**: If one of the strings in *argv* matches `-OAid "DSOM_BOA"`, then BOA initialization is performed, the matching string is consumed, and argc is decremented. (The remaining strings in argv may be reordered as part of consuming the `-OAid "DSOM_BOA"` string.) If *argv* is NULL or contains no string that matches `-OAid "DSOM_BOA"`, then the BOA is initialized only if the *boa_identifier* parameter is `"DSOM_BOA"`.]

boa_identifier
A string that indicates which BOA to initialize. [**Workstation Implementation**: If no string in the *argv* parameter matches `-OAid "DSOM_BOA"`, then the BOA is initialized only if the *boa_identifier* parameter is `"DSOM_BOA"`.]

Return value

CORBA::BOA_ptr
A pointer to the BOA object. The return result should be released using CORBA::release(ORB_ptr).

Exceptions

CORBA::SystemException

Remarks

This method is intended to be used by all server applications to both initialize the BOA and obtain a pointer to it. This method can be called multiple times without adverse effect (the BOA is only initialized once, regardless of how many times BOA_init is called). The return value should be released using CORBA::release(BOA_ptr).

Note: All Component Broker server programs implicitly provide the necessary BOA_init invocation suitable for most applications. Thus, under normal conditions, the use of BOA_init() is not required or recommended.

Example

```
/* This is a minimal, dummy server program. It does not create any
   object to export. It assumes that the server with the name
   "dummyServer" is already registered in the implementation
   repository.
 */

#include #include "corba.h"
void main(int argc, char* argv[])
{
  try
  {
    /* Initialize the server's ImplementationDef, ORB, and BOA */
    CORBA::ImplRepository_ptr implrep = new CORBA::ImplRepository;
    CORBA::ImplementationDef_ptr imp =
       implrep->find_impldef_by_alias ("dummyServer");
    /* Assume op-parm is initialized. For workstation initialize
       to "DSOM" */
    char * op_parm;
    /* Assume bp_parm is initialized. For workstation initialize
       to "DSOM_BOA" */
    char * bp-parm;
    static CORBA::ORB_ptr op = CORBA::ORB_init(argc, argv, op_parm);
    static CORBA::BOA_ptr bp = op->BOA_init(argc, argv, bp_parm);
    bp->impl_is_ready(imp);
    /* To customize, fill in : create objects to export, and so on */
    cout << "server listening ...." << endl;
    cout.flush();
    bp->execute_request_loop(CORBA::BOA::SOMD_WAIT);
  }
  catch (CORBA::SystemException &sysex)
  {
    cout << "caught a system exception, terminating." << endl;
    cout.flush();
  }
}
```

ORB::create_alias_tc

Creates a tk_alias TypeCode.

Original class

CORBA::ORB

IDL syntax

```
CORBA::TypeCode_ptr create_alias_tc (
            CORBA::RepositoryId rep_id,
            CORBA::Identifier   name,
            ORBA::TypeCode_ptr original_type);
```

Parameters

rep_id The Interface Repository identifier for the alias. The caller retains ownership of this string.

name The simple name of the alias. The caller retains ownership of this string.

original_type
The non-NULL TypeCode of the type being aliased. The caller retains ownership of this TypeCode.

Return value

CORBA::TypeCode_ptr
The newly-created TypeCode. The caller assumes ownership of this TypeCode, and should subsequently release it using CORBA::release(TypeCode_ptr).

Exceptions

CORBA::SystemException

Remarks

This method is intended to be used to create a TypeCode of kind tk_alias, representing an IDL typedef.

Example

```
/* Code to create a tk_alias TypeCode corresponding to this IDL
   definition: "typedef long my_long;"
 */
/* assume op initialized */
extern CORBA::ORB_ptr op;
CORBA::RepositoryId rep_id = CORBA::string_dup("RepositoryId_999");
CORBA::Identifier name = CORBA::string_dup("my_long");
CORBA::TypeCode_ptr tc = op->create_alias_tc (rep_id, name, CORBA::_tc_long);
```

ORB::create_array_tc

Creates a tk_array TypeCode.

Original class

CORBA::ORB

IDL syntax

```
CORBA::TypeCode_ptr create_array_tc (
           CORBA::ULong length,
           CORBA::TypeCode_ptr element_type_code);
```

Parameters

length The length of the IDL array.

element_type_code
A non-NULL TypeCode representing the type of the elements of the array. The caller retains ownership of this TypeCode.

Return value

CORBA::TypeCode_ptr
The newly-created TypeCode. The caller assumes ownership of this TypeCode, and should subsequently release it using CORBA::release(TypeCode_ptr).

Exceptions

CORBA::SystemException

Remarks

This method is intended to be used to create a TypeCode of kind tk_array, representing an IDL array.

Example

```
/* Code to create a tk_array TypeCode corresponding to this IDL
   definition: "typedef string my_string[1997];"
 */
/* assume op initialized */
extern CORBA::ORB_ptr op;
CORBA::TypeCode_ptr tc = op->create_array_tc(1997, CORBA::_tc_string);
```

ORB::create_context_list

Creates a CORBA::ContextList object.

Original class

CORBA::ORB

IDL syntax

```
CORBA::Status create_context_list (CORBA::ContextList_ptr& cntxt_list);
```

Parameters

cntxt_list
A pointer for a CORBA::ContextList object, passed by reference, to be initialized by the CORBA::ORB::create_context_list method. The caller assumes ownership of the new ContextList object, but if the caller passes the

ContextList to the CORBA::Object::create_request method, ownership of the ContextList is then transferred to the Request object.

Return value

CORBA::Status
A zero return code indicates success.

Exceptions

CORBA::SystemException

Remarks

The CORBA::ORB::create_context_list method is intended to be used by client applications using the Dynamic Invocation Interface (DII), to create a CORBA::ContextList object to be subsequently passed to the CORBA::Object::create_request method.

Example

```
/* The following program creates a CORBA::context list
   object and generates a system exception if appropriate
 */
#include "corba.h"
#include
int main(int argc, char* argv[])
{
  int rc = 0;
  CORBA::ContextList_ptr CLptr = CORBA::ContextList::_nil();
  /* assume op initialized */
  extern CORBA::ORB_ptr op;
  try
  {
    CORBA::Status st = orb->create_context_list(CLptr);
  }
  catch (CORBA::SystemException &se)
  {
    cout << "exception: " << se.id() << endl; rc="1;"
  } return rc;
}
```

ORB::create_enum_tc

Creates a tk_enum TypeCode.

Original class

CORBA::ORB

IDL syntax

```
CORBA::TypeCode_ptr create_enum_tc (
    CORBA::RepositoryId rep_id,
    CORBA::Identifier name,
    CORBA::EnumMemberSeq & members);
```

Parameters

rep_id The non-NULL Interface Repository identifier of the IDL enum. The caller retains ownership of this string.

name The non-NULL simple name of the IDL enum. The caller retains ownership of this string.

members
A CORBA::EnumMemberSeq object (essentially a sequence of strings) listing the members of the IDL enum. The caller retains ownership of this object. The length of this sequence cannot be zero, and the contained strings must not be NULL.

Return value

CORBA::TypeCode_ptr
The newly-created TypeCode. The caller assumes ownership of this TypeCode, and should subsequently release it using CORBA::release(TypeCode_ptr).

Exceptions

CORBA::SystemException

Remarks

This method is intended to be used to create a TypeCode of kind tk_enum, representing an IDL enum.

Example

```
/* Code to create a tk_enum TypeCode corresponding to this
   IDL definition: enum color { red, green, blue };
 */
/* assume op initialized */
extern CORBA::ORB_ptr op;
CORBA::Identifier identenum = CORBA::string_dup ("color");
CORBA::EnumMemberSeq enum_seq;
enum_seq.length(3);
enum_seq[0].type = CORBA::_tc_string;
enum_seq[0].name = CORBA::string_dup("red");
enum_seq[1].type = CORBA::_tc_string;
enum_seq[1].name = CORBA::string_dup("green");
enum_seq[2].type = CORBA::_tc_string;
enum_seq[2].name = CORBA::string_dup("blue");
```

```
CORBA::RepositoryId rep_id = CORBA::string_dup ("RepositoryId_999");
CORBA::TypeCode_ptr tc= op->create_enum_tc (rep_id, identenum, enum_seq);
...
```

ORB::create_environment

Creates an Environment object.

Original class

CORBA::ORB

IDL syntax

```
CORBA::Status create_environment (CORBA::Environment_ptr& envptr);
```

Parameters

envptr A pointer for a CORBA::Environment object, passed by reference, to be initialized by the CORBA::ORB::create_environment method. The caller assumes ownership of the new Environment object.

Return value

CORBA::Status
A zero return value indicates success.

Exceptions

CORBA::SystemException

Remarks

This method is intended to be used to create an Environment object.

Example

```
#include "corba.h"
int main(int argc, char* argv[])
{
  /* assume cop initialized */
  extern CORBA::ORB_ptr cop;
  CORBA::Environment_ptr envptr = CORBA::Environment::_nil();
  CORBA::Status status = cop->create_environment(envptr);
  return status;
}
```

ORB::create_exception_list

Creates a CORBA::ExceptionList object.

Original class

CORBA::ORB

IDL syntax

```
CORBA::Status create_exception_list (CORBA::ExceptionList_ptr & excp_list);
```

Parameters

excp_list
A pointer for a CORBA::ExceptionList object, passed by reference, to be initialized by the CORBA::ORB::create_exception_list method. The caller assumes ownership of the new ExceptionList object, but if the caller passed the ExceptionList to the CORBA::Object::create_request method, ownership of the ExceptionList is then transferred to the Request object.

Return value

CORBA::Status
A zero return code indicates success.

Exceptions

CORBA::SystemException

Remarks

The CORBA::ORB::create_exception_list method is intended to be used by client applications using the Dynamic Invocation Interface (DII), to create a CORBA::ExceptionList object to be subsequently passed to the CORBA::Object::create_request method.

Example

```
#include "corba.h"
#include
int main(int argc, char* argv[])
{
  int rc = 0;
  CORBA::ExceptionList_ptr ELptr = CORBA::ExceptionList::_nil();
  /* assume orb initialized */
  extern CORBA::ORB_ptr orb;
  try
  {
    CORBA::Status st = orb->create_exception_list(ELptr);
  }
  catch(CORBA::SystemException &se)
  {
    cout << "exception: " << se.id() << endl; rc="1;"
  }
  return rc;
}
```

ORB::create_exception_tc

Creates a tk_except TypeCode.

Original class

CORBA::ORB

IDL syntax

```
CORBA::TypeCode_ptr create_exception_tc (
    CORBA::RepositoryId rep_id,
    CORBA::Identifier name,
    CORBA::StructMemberSeq & members);
```

Parameters

rep_id The non-NULL Interface Repository identifier of the IDL exception. The caller retains ownership of this string.

name The non-NULL simple name of the IDL exception. The caller retains ownership of this string.

members
A CORBA::StructMemberSeq object (a sequence of structs of type CORBA::StructMember) listing the members of the IDL exception. Each CORBA::StructMember in the sequence specifies the name and type of the corresponding exception member; only the type member is used, and the type_def member should be set to NULL. The caller retains ownership of this object.

Return value

CORBA::TypeCode_ptr
The newly-created TypeCode. The caller assumes ownership of this TypeCode, and should subsequently release it using CORBA::release(TypeCode_ptr).

Exceptions

CORBA::SystemException

Remarks

This method is intended to be used to create a TypeCode of kind tk_except, representing an IDL exception.

Example

```
/* Code to create a tk_except TypeCode corresponding to this IDL definition:
     exception my_exception { string my_string; }'
 */
/* assume op initialized */
```

```
extern CORBA::ORB_ptr op;
CORBA::RepositoryId rep_id = CORBA::string_dup("RepositoryId_999");
CORBA::Identifier name = CORBA::string_dup("my_exception");
CORBA::StructMemberSeq st_seq;
st_seq.length(1);
st_seq[0].type = CORBA::_tc_string;
st_seq[0].name = CORBA::string_dup("my_string");
CORBA::TypeCode_ptr tc = op->create_exception_tc (rep_id, name, st_seq);
```

ORB::create_interface_tc

Creates a tk_objref TypeCode.

Original class

CORBA::ORB

IDL syntax

```
CORBA::TypeCode_ptr create_interface_tc (
    CORBA::RepositoryId rep_id,
    CORBA::Identifier name);
```

Parameters

rep_id The non-NULL Interface Repository identifier of the IDL interface. The caller retains ownership of this string.

name The non-NULL simple name of the IDL interface. The caller retains ownership of this string.

Return value

CORBA::TypeCode_ptr
The newly-created TypeCode. The caller assumes ownership of this TypeCode, and should subsequently release it using CORBA::release(TypeCode_ptr).

Exceptions

CORBA::SystemException

Remarks

This method is intended to be used to create a TypeCode of kind tk_objref, representing an IDL interface.

Example

```
/* Code to create a tk_objref TypeCode corresponding to this
   IDL definition: interface my_interface;
 */
/* assume op initialized */
```

```
extern CORBA::ORB_ptr op;
CORBA::RepositoryId rep_id = CORBA::string_dup("RepositoryId_999");
CORBA::Identifier name = CORBA::string_dup("my_interface");
CORBA::TypeCode_ptr tc = op->create_interface_tc (rep_id, name);
```

ORB::create_list

Creates a CORBA::NVList object.

Original class

CORBA::ORB

IDL syntax

```
CORBA::Status create_list (CORBA::Long count,
                           CORBA::NVList_ptr& nvlist);
```

Parameters

count The number of elements in the CORBA::NVList to be created. A zero value is valid.

nvlist A pointer for a CORBA::NVList, passed by reference, to be initialized by the CORBA::ORB::create_list method. The caller assumes ownership of the NVList object, but if the same object is passed to the CORBA::Object::create_request method, the Request object assumes ownership of the NVList.

Return value

CORBA::Status
A zero return value indicates success.

Exceptions

CORBA::SystemException

Remarks

This method is intended to be used to create a CORBA::NVList object, when using the Dynamic Invocation Interface (DII), to be passed to the CORBA::Object::create_request method. The caller specifies the length of the NVLIst to be created; upon return, the new NVList contains the specified number of (uninitialized) items. The caller must initialize the items in the NVList (or add new items) prior to passing it to the CORBA::Object::create_request method. Note, however, that since there is no mechanism for updating the flags of a NamedValue already contained by an NVList, it is advisable to create the list initially empty (that is, pass zero as the count), and then initialize the list by adding (initialized) CORBA::NamedValue objects to it, using the method s on CORBA::NVList.

See also ORB::create_operation_list and Object::_request.

Example

```
/* The following program creates a CORBA::create_list object and
   generates a system exception if appropriate
 */
#include "corba.h"
#include   CORBA::Long NUMITEMS = 3;
int main(int argc, char* argv[])
{
  int rc = 0;
  CORBA::NVList_ptr NVLptr = CORBA::NVList::_nil();
  /* assume orb initialized */
  extern CORBA::ORB_ptr orb;

  try
  {
    CORBA::Status st = orb->create_list(NUMITEMS, NVLptr);
  }
  catch (CORBA::SystemException &se)
  {
    cout << "exception: " << se.id() << endl; rc="1;"
  }
  return rc;
}
```

ORB::create_named_value

Creates a CORBA::NamedValue object.

Original class

CORBA::ORB

IDL syntax

```
CORBA::Status create_named_value (CORBA::NamedValue_ptr& nv)
```

Parameters

nv A pointer for a CORBA::NamedValue object, passed by reference, to be initialized by the CORBA::ORB::create_named_value method. The caller assumes ownership of the new NamedValue object, but if the caller passes the NamedValue to the CORBA::Object::create_request method, ownership of the NamedValue is then transferred to the Request object.

Return value

CORBA::Status
A zero return code indicates success.

Exceptions

CORBA::SystemException

Remarks

The CORBA::ORB::create_named_value method is intended to be used by client applications using the Dynamic Invocation Interface (DII), to create a CORBA::NamedValue object to be subsequently passed to the CORBA::Object::create_request method.

Example

```
/* The following program creates a CORBA::NamedValue object and
   generates a system exception if appropriate
 */
#include "corba.h"
#include   int main(int argc, char* argv[])
{
  int rc = 0;
  CORBA::NamedValue_ptr NVptr = CORBA::NamedValue::_nil();
  /* assume orb initialized */
  extern CORBA::ORB_ptr orb;

  try
  {
    CORBA::Status st = orb->create_named_value(NVptr);
  }
  catch (CORBA::SystemException &se)
  {
    cout << "exception: " << se.id() << endl; rc="1;"
  }
  return rc;
}
```

ORB::create_operation_list

Creates a CORBA::NVList for a particular IDL operation.

Original class

CORBA::ORB

IDL syntax

```
CORBA::Status create_operation_list (
  CORBA::OperationDef_ptr operdf,
  CORBA::NVList_ptr& nvlist);
```

Parameters

operdf A non-NULL CORBA::OperationDef object, obtained from the Interface Repository, that describes the operation that the new NVList will describe. The caller retains ownership of this object.

nvlist A pointer for a CORBA::NVList object, passed by reference, to be initialized by the CORBA::ORB::create_operation_list method. The caller assumes

ownership of the new NVList object, but if the caller subsequently passes the NVList to the CORBA::Object::create_request method, the Request object then assumes ownership of the NVList.

Return value

CORBA::Status
A zero return value indicates success.

Exceptions

CORBA::SystemException

Remarks

The CORBA::ORB::create_operation_list method is intended to be used by client applications that are using the Dynamic Invocation Interface (DII), to create a CORBA::NVList object to be passed to the CORBA::Object::create_request method. The new NVList contains an item describing the name, type, and mode of each parameter of the IDL operation described by the input CORBA::OperationDef object. The application must update the NVList with the values of any in and inout parameters before invoking the corresponding DII request.

See also ORB::create_list.

Example

See example in CORBA::"Object::_create_request" on page 157.

ORB::create_recursive_sequence_tc

Creates a tk_recursive_sequence TypeCode.

Original class

CORBA::ORB

IDL syntax

```
CORBA::TypeCode_ptr create_recursive_sequence_tc (
    CORBA::ULong bound,
    CORBA::ULong offset);
```

Parameters

bound The bound of the IDL sequence. Zero designates an unbounded sequence.

offset Indicates which enclosing TypeCode describes the elements of the recursive

sequence. It is the level of nesting of the sequence in the type that matches the sequence's elements. For example, the sequences in the following examples all have an offset of one:

```
struct foo1 {
  long value;
  sequence <foo1> chain;
  };
struct foo2 {
  long value1;
  long value2;
  sequence <foo2> chain;
  };
struct foo3 {
  struct foo4 {
    sequence <foo4> chain;
  };
};
```

while the sequences in the following example has an offset of two:

```
struct foo4 {
  struct foo5 {
    sequence <foo4> chain;
  };
};
```

Return value

CORBA::TypeCode_ptr
The newly-created TypeCode. The caller assumes ownership of this TypeCode., and should subsequently release it using CORBA::release(TypeCode_ptr)

Exceptions

CORBA::SystemException

Remarks

This method is intended to be used to create a TypeCode of kind tk_recursive_sequence, representing a recursive IDL sequence. (A recursive sequence is one whose element type matches a type in which the recursive sequence is nested. For example, if IDL struct A contains a sequence of A, then the sequence is a recursive sequence.) The result of this method is used to construct other TypeCodes.

See also the CORBA::ORB::create_sequence_tc method, for creating TypeCodes describing non-recursive IDL sequences.

Example

```
/* Code to create a tk_recursive_sequence TypeCode corresponding to
   this IDL definition:
       struct my_struct { long my_long;
                          sequence my_seq; };
```

```
 */
/* assume op initialized */
extern CORBA::ORB_ptr op;
CORBA::TypeCode_ptr tc = op->create_recursive_sequence_tc (3, 1);
```

ORB::create_sequence_tc

Creates a tk_sequence TypeCode.

Original class

CORBA::ORB

IDL syntax

```
CORBA::TypeCode_ptr create_sequence_tc (
                      CORBA::ULong bound,
                      CORBA::TypeCode_ptr element_type);
```

Parameters

bound The bound of the IDL sequence. Zero designates an unbounded sequence.

element_type
A non-NULL CORBA::TypeCode describing the type of the sequence elements. The caller retains ownership of this TypeCode.

Return value

CORBA::TypeCode_ptr
The newly-created TypeCode. The caller assumes ownership of this TypeCode, and should subsequently release it using CORBA::release(TypeCode_ptr).

Exceptions

CORBA::SystemException

Remarks

This method is intended to be used to create a TypeCode of kind tk_sequence, representing an IDL sequence.

See also the CORBA::ORB::create_recursive_sequence_tc method, for creating TypeCodes describing recursive IDL sequences.

Example

```
/* Code to create a tk_sequence TypeCode corresponding to this
   IDL definition:
       sequence  my_seq;
 */
```

```
/* assume op initialized */
extern CORBA::ORB_ptr op;
CORBA::TypeCode_ptr tc = op->create_sequence_tc (12, CORBA::_tc_short);
```

ORB::create_string_tc

Creates a tk_string TypeCode.

Original class

CORBA::ORB

IDL syntax

```
CORBA::TypeCode_ptr create_string_tc (CORBA::ULong bound);
```

Parameters

bound The bound of the IDL string. Zero designates an unbounded string.

Return value

CORBA::TypeCode_ptr
The newly-created TypeCode. The caller assumes ownership of this TypeCode, and should subsequently release it using CORBA::release(TypeCode_ptr).

Exceptions

CORBA::SystemException

Remarks

This method is intended to be used to create a TypeCode of kind tk_string, representing an IDL string.

Example

```
/* Code to create a tk_string TypeCode corresponding to this
   IDL definition:
       string <123> my_string; (this is a bounded string)
 */
/* assume op initialized */
extern CORBA::ORB_ptr op;
CORBA::TypeCode_ptr tc = op->create_string_tc (123);
```

ORB::create_struct_tc

Creates a tk_struct TypeCode.

Original class

CORBA::ORB

IDL syntax

```
CORBA::TypeCode_ptr create_struct_tc (
    CORBA::RepositoryId rep_id,
    CORBA::Identifier name,
    CORBA::StructMemberSeq & members);
```

Parameters

rep_id The non-NULL Interface Repository identifier of the IDL struct. The caller retains ownership of this string.

name The non-NULL simple name of the IDL struct. The caller retains ownership of this string.

members
A CORBA::StructMemberSeq object (a sequence of structs of type CORBA::StructMember) listing the members of the IDL struct. Each CORBA::StructMember in the sequence specifies the name and type of the corresponding struct member; only the type member is used, and the type_def member should be set to NULL. The sequence must contain at least one CORBA::StructMember, and each CORBA::StructMember in the sequence must have a non-NULL TypeCode. The caller retains ownership of this object.

Return value

CORBA::TypeCode_ptr
The newly-created TypeCode. The caller assumes ownership of this TypeCode, and should subsequently release it using CORBA::release(TypeCode_ptr).

Exceptions

CORBA::SystemException

Remarks

This method is intended to be used to create a TypeCode of kind tk_struct, representing an IDL struct.

Example

```
/* Code to create a tk_struct TypeCode corresponding to this
   IDL definition:
       struct my_struct
       {
         long my_long;
         char my_char;
       };
 */
```

```
/* assume op initialized */
extern CORBA::ORB_ptr op;
CORBA::_IDL_SEQUENCE_StructMember stm_seq;
stm_seq.length(2);
stm_seq [0].type = CORBA::_tc_long;
stm_seq [0].name = CORBA::string_dup ("my_long");
stm_seq [1].type = CORBA::_tc_char;
stm_seq [1].name = CORBA::string_dup ("my_char");

CORBA::RepositoryId rep_id = CORBA::string_dup("RepositoryId_999");
CORBA::Identifier name = CORBA::string_dup("my_struct");
CORBA::TypeCode_ptr tc = op->create_struct_tc (rep_id, name, stm_seq);
```

ORB::create_union_tc

Creates a tk_union TypeCode.

Original class

CORBA::ORB

IDL syntax

```
CORBA::TypeCode_ptr create_union_tc (
    CORBA::RepositoryId rep_id,
    CORBA::Identifier name,
    CORBA::TypeCode_ptr discriminator_type,
    CORBA::UnionMemberSeq & members);
```

Parameters

rep_id The non-NULL Interface Repository identifier of the IDL union. The caller retains ownership of this string.

name The non-NULL simple name of the IDL union. The caller retains ownership of this string.

discriminator_type
A non-NULL CORBA::TypeCode describing the type of the union's discriminator. The caller retains ownership of this TypeCode.

members
A CORBA::UnionMemberSeq object (a sequence of unions of type CORBA::UnionMember) listing the members of the IDL union. Each CORBA::UnionMember in the sequence specifies the name, type, and member label of the corresponding union member. The type member is used, but the type_def member should be set to NULL. A union-member label of the zero octet is used to indicate the default union member. The sequence must contain at least one CORBA::UnionMember, and the TypeCode of each CORBA::UnionMember in the sequence must be non-NULL. The caller retains ownership of this object.

Return value

CORBA::TypeCode_ptr

The newly-created TypeCode. The caller assumes ownership of this TypeCode, and should subsequently release it using CORBA::release(TypeCode_ptr).

Exceptions

CORBA::SystemException

Remarks

This method is intended to be used to create a TypeCode of kind tk_union, representing an IDL union.

Example

```
/* Code to create a tk_union TypeCode corresponding to this
   IDL definition:
       union my_union switch (long)
       {
          case1: ulong my_ulong;
          case2: float my_float;
       };
 */
/* assume op initialized */
extern CORBA::ORB_ptr op;

CORBA::_IDL_SEQUENCE_UnionMember unm_seq;
unm_seq.length(2);

/* Set the member typecode and the member name for the first UnionMember */
unm_seq [0].type = CORBA::_tc_ulong;
unm_seq [0].name = CORBA::string_dup("my_ulong");

/* Set the member typecode and the member name for the second UnionMember */
unm_seq [1].type = CORBA::_tc_float;
unm_seq [1].name = CORBA::string_dup("my_float");

/* Create the Any that define the two member labels */
unm_seq [0].label <<= (corba::long) 1;
unm_seq [1].label <<="(CORBA::Long)" 2;
corba::repositoryid rep_id="CORBA::string_dup("repositoryid_999");
" corba::identifier name="CORBA::string_dup("my_union"); "
corba::typecode_ptr discriminator_type = "CORBA::_t _long;"
corba::typecode_ptr tc="op-">create_union_tc (rep_id, name,
  discriminator_type, unm_seq);
```

ORB::get_current

Returns the CORBA::Current object of the calling thread.

Note: This method has been deprecated in Version 3.0 of Component Broker and will be removed altogether in a future version.

AIX 390 WIN The ORB::resolve_initial_references() method should be used instead on AIX, OS/390, and Windows NT.

Original class

CORBA::ORB

IDL syntax

```
CORBA::Current_ptr get_current(const char * current_classname);
```

Parameters

current_classname
The classname of a subclass of current.

Return value

CORBA::Current_ptr
The Current object that reflects the current execution context of the calling thread. The caller assumes ownership of this object and should subsequently release it using CORBA::release.

Exceptions

CORBA::SystemException

Remarks

The CORBA::ORB::get_current method is intended to be used by both client and server applications to obtain a pointer to the CORBA::Current object that reflects the execution context of the calling thread. The caller must narrow the returned reference to an instance of some subclass of CORBA::Current. (The Security Service and the Transaction Service define subclasses of CORBA::Current.)

Example

AIX 390 WIN For AIX, OS/390, and Windows NT:

```
#include "corba.h"
int main(int argc, char* argv[])
{
int rc = 0;
/* assume cop initialized */
extern CORBA::ORB_ptr cop;

/*This example obtains a Transactions Current as opposed to a Security
Current - make sure that the parameter you use reflects the specific
```

```
specialization of Current you want to use*/
CORBA::Current_ptr current = cop->get_current("TransactionCurrent");

/* narrow the current object ...*/
CORBA::release(current);
return rc;
}
```

SOLARIS For Solaris:

```
#include "corba.h"
int main(int argc, char* argv[])
{
int rc = 0;
/* assume cop initialized */
extern CORBA::ORB_ptr cop;

/*This example obtains a Transactions Current as opposed to a Security
Current - make sure that the parameter you use reflects the specific
specialization of Current you want to use*/
CORBA::Current_ptr current = cop->get_current("CosTransactions::Current");

/* narrow the current object ...*/
CORBA::release(current);
return rc;
}
```

ORB::get_default_context

Returns the default CORBA::Context object.

Original class

CORBA::ORB

IDL syntax

```
CORBA::Status get_default_context (CORBA::Context_ptr& ctx);
```

Parameters

ctx A pointer for a CORBA::Context object, passed by reference, to be initialized by the CORBA::ORB::get_default_context method. The caller assumes ownership of the CORBA::Context object.

Return value

CORBA::Status
A zero return value indicates success.

Exceptions

CORBA::SystemException

Remarks

This method is intended to be used by client applications to obtain a default CORBA::Context object, which can be passed to IDL operations that require a Context parameter. The default CORBA::Context object contains a name/value pair for each environment variable set in the calling process's environment.

Example

```
/* The following program creates a CORBA::Context object and generates
   system exception if appropriate
 */
#include "corba.h"
#include   int main(int argc, char* argv[])
{
  int rc = 0;
  CORBA::Context_ptr Ctxtptr = CORBA::Context::_nil();
  /* assume orb initialized */
  extern CORBA::ORB_ptr orb;

  try
  {
    CORBA::Status st = orb->get_default_context( Ctxtptr);
  }
  catch (CORBA::SystemException &se)
  {
    cout << "exception: " << se.id() << endl; rc="1;"
  }
  return rc;
}
```

ORB::get_next_response

Returns the next available response, after issuing multiple deferred requests in parallel.

Original class

CORBA::ORB

IDL syntax

```
CORBA::Status get_next_response (CORBA::Request_ptr& req);
```

Parameters

req A pointer for a CORBA::Request object, passed by reference, to be initialized by the CORBA::ORB::get_next_response method to point to the CORBA::Request object whose response was received. The CORBA::Request object is owned by the client that originally issued the Request.

Return value

CORBA::Status
A zero return value indicates success.

Exceptions

CORBA::SystemException

Remarks

The CORBA::ORB::get_next_response method is intended to be used by client applications that are using the Dynamic Invocation Interface (DII), to obtain the next available response after sending multiple deferred requests in parallel (for example, using CORBA::ORB::send_multiple_requests_deferred or CORBA::Request::send_deferred). The order in which responses are received does not necessarily match the order in which requests were sent. If no response is currently available, this method will block until a response is available. To avoid blocking, use the CORBA::ORB::poll_next_reponse method.

Example

```
/* Assume the following IDL interface:
       interface testObject
       {
         string testMethod (in long input_value, out float out_value);
       };
 */
#include "corba.h"
...
/* assume cop initialized */
extern CORBA::ORB_ptr cop;
/* Create the Request object */
CORBA::Object_var my_proxy =                          /* get a proxy somehow */
CORBA::Request_ptr req = my_proxy->_request ("testMethod");
req->add_in_arg() <<= (corba::long) 12345;          /* sets type and value */
  corba::float out_float; req -> add_out_arg() <<= out_float; /* sets type */
  req -> set_return_type (CORBA::_tc_string);
...
while (!cop->poll_next_response())
{
  /* Wait */ ...
};           /* determine if a response to a deferred request is available */
cop->get_next_response(req);         /* return the next available response */
...
```

ORB::get_service_information

Describes what services of a particular type are available.

Not supported by OS/390 Component Broker

Original class

CORBA::ORB

IDL syntax

```
CORBA::Boolean get_service_information (CORBA::ServiceType service_type,
  CORBA::ServiceInformation& service_information);
```

Parameters

service_type
The identifier of the service for which information is needed. For example, use CORBA::Security to obtain information about what security services are available in the calling process.

service_information
A CORBA::ServiceInformation variable, passed by reference, to be initialized by the CORBA::ORB::get_service_information method.

Return value

CORBA::Boolean
Zero indicates that the requested service is not available, and hence that the service_information parameter has not been updated. A nonzero return value indicates that the service_information parameter has been initialized.

Exceptions

CORBA::SystemException

Remarks

This method is intended to be used by client and server applications to determine what services of a particular type (such as Security services) are available. The result of CORBA::ORB::get_service_information does not vary during the lifetime of a single process.

Example

```
#include "corba.h"
int main(int argc, char* argv[])
{
  int rc = 0;
  /* assume cop initialized */
  extern CORBA::ORB_ptr cop;
  CORBA::ServiceInformation si ;
  /* request service information for CORBA::Security */
```

```
  CORBA::Boolean retval =
    cop->get_service_information(CORBA::Security, si);
  return rc;
}
```

ORB::list_initial_services

Lists the runtime objects available by calling the CORBA::ORB::resolve_initial_references method.

Original class

CORBA::ORB

IDL syntax

```
CORBA::ObjectIdList* list_initial_services ();
```

Return value

CORBA::ObjectIdList *

A pointer to a sequence of strings, where each string is an identifier that can be passed to CORBA::ORB::resolve_initial_references. The caller assumes ownership of the returned result and should subsequently delete it.

Exceptions

CORBA::SystemException

Remarks

The CORBA::ORB::list_initial_services method is intended to be used by client and server applications to determine what object references are available from the CORBA::ORB::resolve_initial_references method.

Example

```
/* This program lists the runtime objects available into a
   CORBA::ORB::ObjectIdList obj
 */
#include "corba.h"
#include   int main(int argc, char* argv[])
{
  int rc = 0;
  CORBA::ORB::ObjectIdList *idlist = NULL;
  /* assume orb initialized */
  extern CORBA::ORB_ptr orb;
  try
  {
    idlist = orb->list_initial_services();
  }
  catch (CORBA::SystemException &se)
```

```
  {
    cout << "exception : " << se.id() << endl; rc="1;"
  }
  if (idlist)
  {
    /* use idlist such as idlist->length(), (*idlist)[i] where i is
       index ... */
  }
  return rc;
}
```

ORB::object_to_string

Converts an object reference to an external form that can be stored outside the ORB or exchanged between processes.

Original class

CORBA::ORB

IDL syntax

```
char * object_to_string (CORBA::Object_ptr obj);
```

Parameters

obj The object reference to be converted to string form. Nil object references are valid.

Return value

char * The string form of the input object reference (either a proxy object or a local object). The caller assumes ownership of this string and should subsequently free it using CORBA::string_free.

Exceptions

CORBA::SystemException

Remarks

The CORBA::ORB::object_to_string method is intended to be used by client or server applications to convert object references (either proxy objects or local objects) to a string form that has meaning outside the process. The CORBA::ORB::string_to_object can then be used to reconstitute the object reference (either in the same process or a different process). The output string is compliant with the CORBA 2.0 specification for Interoperable Object References.

If the caller is a server (that it, the caller has already invoked CORBA::BOA::impl_is_ready), and the input object reference is a local object, then the

resulting string can be passed to CORBA::ORB::string_to_object to construct a proxy in another process, or to obtain the original object pointer in the same process.

If the caller is not a server and the input object reference is a local object (rather than a proxy), then the result of CORBA::ORB::object_to_string is valid only within the calling process for the lifetime of the process and as long as the input object resides in the process.

Example

```
/* convert local object to string representation. Assume that p is
   a local object pointer already declared and defined of a class,
   say Foo
 */
#include "corba.h"
#include   ...
/* assume op initialized */
extern CORBA::ORB_ptr op;
CORBA::string str = op->object_to_string(p);
CORBA::Object_ptr objPtr = op->string_to_object(str);
/* narrow down objPtr by calling Foo::_narrow(objPtr),
   get back a local obj same as p ...
 */
CORBA::string_free(str);
CORBA::release(objPtr);
...
```

ORB::poll_next_response

Determines whether a response to a deferred request is available.

Original class

CORBA::ORB

IDL syntax

```
CORBA::Boolean poll_next_response ();
```

Return value

CORBA::Boolean
A non-zero return value indicates that a response is available.

Exceptions

CORBA::SystemException

Remarks

The CORBA::ORB::poll_next_response method is intended to be used by client applications that are using the Dynamic Invocation Interface (DII), to determine whether

a response is available, after sending one or more deferred requests (for example, using CORBA::ORB::send_multiple_requests_deferred or CORBA::Request::send). This method can be called prior to calling ORB::get_next_response, to avoid blocking.

Example

See example in “ORB::get_next_response” on page 196.

ORB::resolve_initial_references

Obtains an object reference to a key service, such as the Naming Service or the Interface Repository.

Original class

CORBA::ORB

IDL syntax

```
CORBA::Object_ptr resolve_initial_references (const char* identifier);
```

Parameters

identifier
The non-NULL identifier of the object reference to be obtained. Valid identifiers are those returned by CORBA::ORB::list_initial_services. The caller retains ownership of this string.

Return value

CORBA::Object_ptr
A reference to the requested services. The caller assumes ownership of the returned object reference, and should subsequently release it using CORBA::release.

Exceptions

- If the input identifier is not valid, a CORBA::ORB::InvalidName exception is thrown.
- If another error occurs, a CORBA::SystemException is thrown.

Remarks

The CORBA::ORB::resolve_initial_references method is intended to be used by client and server applications to obtain initial object references for accessing key services, such as the Interface Repository or the Naming Service. The caller specifies the identifier of the service for which a reference is needed, then narrows the return result to the proper type. For example, when the input is ″InterfaceRepository″, the return result should be narrowed to CORBA::Repository. When the input is ″NameService″, the return result is the root name context of the local naming tree., and should be

narrowed to CosNaming::NamingContext (or some class derived from it). Typically an application uses CORBA::ORB::resolve_initial_references to obtain a reference to the root name context, then invokes operations on that reference to obtain all other object references.

Example

```
#include "corba.h"
...
/* assume op initialized */
extern CORBA::ORB_ptr op;
CORBA::ORB::ObjectIdList *oil = op->list_initial_services();
/* pass in the first element of oil as identifier to obtain object reference */
CORBA::Object_ptr optr ;
optr = op->resolve_initial_references((*oil)[0]);
/* narrow optr appropriately ... */
CORBA::release(optr);
...
```

ORB::resolve_initial_references_remote

Obtains an object reference to the Naming Service.

Supported on AIX and Windows NT only

Original class

CORBA::ORB

IDL syntax

```
CORBA::Object_ptr resolve_initial_references_remote
   (const char * identifier,
    const CORBA::ORB::remote_modifier  host_port_list );
```

Parameters

identifier

The non-NULL identifier of the Naming Service object reference to be obtained. This string must be “NameService”.

host_port_list

This is a list of host names and associated port numbers on which the resolve_initial_references_remote operation will attempt to locate a NameService object. The operation will return the first NameService object located from the host and port combinations provided in the list.

Each string representing a hostname and port combination must be of the following syntax:

```
iiop://HostName:PortNumber
```

Return value

CORBA::Object_ptr

A reference to the requested Naming Service is returned. The caller assumes ownership of the returned object reference, and should subsequently release it using CORBA::release.

Exceptions

- If the input identifier is not valid, a CORBA::ORB::InvalidName exception is thrown.
- If another error occurs, a CORBA::SystemException is thrown.

Remarks

The CORBA::ORB::resolve_initial_references_remote method is intended to be used by client and server applications to obtain a reference to a NameService object from an input list of host names and associated port numbers.

The return result is the first root name context of a naming tree located from the input list of hosts. The returned object should be narrowed to CosNaming::NamingContext (or some class derived from it).

Example

```
#include "corba.h"
...
//-------------------------------------
// - assume the ORB object pointer
//   has already been initialized . . .
//-------------------------------------
extern CORBA::ORB_ptr op;

CORBA::Object_ptr     optr = NULL;
CORBA::String_var     naming_objectid = CORBA::string_dup ("NameService");

//---------------------------------
// - create a host port list and
//   provide room for three entries
//---------------------------------
CORBA::ORB::remote_modifier     host_port_list;
host_port_list.length (3);

//-------------------------------------
// - initialize the host port list
//   with three host name and port number
//   combinations . . .
//-------------------------------------
host_port_list [0] = CORBA::string_dup ("iiop://hostName1:900");
host_port_list [1] = CORBA::string_dup ("iiop://hostName2:3003");
host_port_list [2] = CORBA::string_dup ("iiop://hostName3:900);

optr = op-> resolve_initial_references_remote (naming_objectid,
  host_port_list);
```

```
if (optr != NULL)
{
   //-----------------------------------------------------------------
   // - narrow the object to the appropriate object type
   // - release the object (_narrow performs a _duplicate)
   //-----------------------------------------------
   optr = CORBA::CosNaming::NamingContext::_narrow (optr);
   CORBA::release (optr);
}
...
```

ORB::send_multiple_requests_deferred

Issues multiple deferred requests in parallel.

Original class

CORBA::ORB

IDL syntax

```
CORBA::Status send_multiple_requests_deferred (
  const CORBA::RequestSeq& req_seq);
```

Parameters

req_seq
A sequence of CORBA::Request_ptr objects to be invoked. An empty sequence is valid. However, each CORBA::Request_ptr object in the sequence must be non-NULL. The caller retains ownership of this sequence and of the CORBA::Request objects it contains.

Return value

CORBA::Status
A zero return value indicates that all Requests were issued.

Exceptions

CORBA::SystemException

Remarks

The CORBA::ORB::send_multiple_requests_deferred method is intended to be used by client applications that are using the Dynamic Invocation Interface (DII), to issue multiple deferred requests in parallel. The results of these requests can later be obtained using CORBA::ORB::get_next_response. For each CORBA::Request in the input sequence, CORBA::Request::send_deferred is invoked.

Example

```
#include "corba.h"
...
/* assume op initialized */
extern CORBA::ORB_ptr op;
CORBA::ORB::RequestSeq reqSeq = CORBA::ORB::RequestSeq(1024);
... /* prepare each request in reqSeq */
/* issue multiple deferred requests */
CORBA::Status rc = op->send_multiple_requests_deferred(reqSeq);
...
```

ORB::send_multiple_requests_oneway

Issues multiple oneway requests in parallel.

Original class

CORBA::ORB

IDL syntax

```
CORBA::Status send_multiple_requests_oneway (
  const CORBA::RequestSeq& req_seq);
```

Parameters

req_seq
A sequence of CORBA::Request_ptr objects to be invoked. An empty sequence is valid. However, each CORBA::Request_ptr object in the sequence must be non-NULL. The caller retains ownership of this sequence and of the CORBA::Request objects it contains.

Return value

CORBA::Status
A zero return value indicates that all Requests were issued.

Exceptions

CORBA::SystemException

Remarks

The CORBA::ORB::send_multiple_requests_oneway method is intended to be used by client applications that are using the Dynamic Invocation Interface (DII), to issue multiple oneway requests in parallel. For each CORBA::Request in the input sequence, CORBA:: Request::send_oneway is invoked.

Example

```
#include "corba.h"
...
/* assume op initialized */
extern CORBA::ORB_ptr op;
CORBA::ORB::RequestSeq reqSeq = CORBA::ORB::RequestSeq(1024);
... /* prepare each request in reqSeq */
/* issue multiple oneway requests */
CORBA::Status rc = op->send_multiple_requests_oneway(reqSeq);
...
```

ORB::string_to_object

Converts a string (produced by CORBA::ORB::object_to_string) into an object reference.

Original class

CORBA::ORB

IDL syntax

```
CORBA::Object_ptr string_to_object (const char* str);
```

Parameters

str A string form of an object reference. This string must have been originally generated using CORBA::ORB::object_to_string (although not necessarily by the process). The caller retains ownership of this string.

Return value

CORBA::Object_ptr
The object reference encoded by the input string. The caller assumes ownership of this object reference and should subsequently release it using CORBA::release.

Exceptions

If the input string is not valid, or refers to a local object that is no longer valid (insofar as the ORB can determine), a CORBA::SystemException is thrown.

Remarks

The CORBA::ORB::string_to_object method is intended to be used by client or server applications to convert a string form of an object reference (originally generated using CORBA::ORB::object_to_string) back into an object reference. If the input string refers to a local object residing in a server process (a process that has called CORBA::BOA::impl_is_ready), then the result is the same local object originally passed to CORBA::ORB::object_to_string. If the input string refers to a local object residing in a non-server process, then the result is the same local object originally passed to

CORBA::ORB::object_to_string provided that both calls were made from the same process instance. If the input string refers to an object in another process, then CORBA::ORB::string_to_object always constructs a new proxy object. The validity of the object/server to which the proxy refers is not checked until the application invokes an application operation on the proxy.

Example

See example in “ORB::object_to_string” on page 200.

OperationDef Interface

An OperationDef is used within the Interface Repository to represent the information needed to define an operation of an interface.

File name

somir.idl

Intended usage

The OperationDef object is used to represent the information that defines an operation of an interface. An OperationDef may be created by calling the create_operation operation of the InterfaceDef interface . The create_operation parameters include the unique RepositoryId (CORBA::RepositoryId), the name (CORBA::Identifier), the version (CORBA::VersionSpec), the result (CORBA::IDLType*) to indicate the type of the returned operation result, the mode of the operation (CORBA::OP_NORMAL or CORBA::OP_ONEWAY), a sequence (CORBA::ParDescriptionSeq) defining the parameters of the operation, a sequence (CORBA::ExceptionDefSeq) defining the exceptions of the operation, and a sequence (CORBA::ContextIdSeq) defining the contexts of the operation.

Local-only

True

Ancestor interfaces

Contained Interface

Exceptions

CORBA::SystemException

IDL syntax

```
module CORBA {
   enum OperationMode {OP_NORMAL, OP_ONEWAY};
   enum ParameterMode {PARAM_IN, PARAM_OUT, PARAM_INOUT};
   struct Parameter Description {
```

```
Identifier name;
TypeCode type;
IDLType type_def;
ParamterMode mode;
};
typedef sequence  ParDescriptionSeq;
typedef identifier ContextIdentifier;
typedef sequence  contextIdSeq;
typedef sequence  ExceptionDefSeq;
typedef sequence  ExcDescriptionSeq;
interface OperationDef:Contained {
readonlyattribute TypeCode result;
attribute IDLType result_def;
attribute ParDescriptionSeq params;
attribute OperationMode mode;
attribute ContextIdSeq contexts;
attribute ExceptionDefSeq exceptions;
};
struct OperationDescription {
Identifier name;
RepositoryId id;
RepositoryId defined_in;
VersionSpec version;
TypeCode result;
OperationMode mode;
ContextIdSeq contexts;
ParDescriptionSeq parameters;
ExcDescriptionSeq exceptions;
};
};
```

Supported operations

OperationDef::contexts
OperationDef::describe
OperationDef::exceptions
OperationDef::mode
OperationDef::params
OperationDef::result
OperationDef::result_def

OperationDef::contexts

The context read and write operations allow the access and update of the list of context identifiers that apply to an operation (CORBA::OperationDef object) in the Interface Repository.

Original interface

OperationDef Interface

IDL syntax

```
attribute ContextIdSeq contexts;
```

Parameters

ContextIdSeq & contexts

In Read operation, no input parameters are defined.

In Write operation, CORBA::ContextIdSeq & contexts. The contexts parameter is the new list of contexts with which to update the operation definition (the length of the sequence may be set to zero to indicate no contexts).

Return value

ContextIdSeq *

In Read operation, CORBA::ContextIdSeq *. The returned value is a pointer to a copy of the contexts attribute of the operation definition. The memory is owned by the caller and can be removed by invoking delete.

Exceptions

CORBA::SystemException
CORBA::NO_IMPLEMENT thrown by Write operation on OS/390

Remarks

The context attribute specifies the list of context identifiers that apply to an operation definition. The context read and write operations are supported with parameter and result definitions as described below.

Example

```
// C++
   // assume that 'this_operation' has already been initialized
   CORBA::OperationDef * this_operation;

   // establish the sequence of contexts for updating the operation
   // definition
   CORBA::ContextIdSeq seq_update;
   seq_update.length (2);
   seq_update[0] = CORBA::string_dup ("CONTEXT_0=value_0");
   seq_update[1] = CORBA::string_dup ("CONTEXT_1= value_1");

   // update the operation with the new contexts list
   this_operation-> contexts (seq_update);

   // retrieve the contexts list from the operation definition
   CORBA::ContextIdSeq * returned_context_list;
   returned_context_list = this_operation-> contexts ();
```

OperationDef::describe

The describe operation returns a structure containing information about a CORBA::OperationDef Interface Repository object.

Original interface

OperationDef Interface

IDL syntax

```
struct OperationDescription {
   Identifier name;
   RepositoryId id;
   RepositoryId defined_in;
   VersionSpec version;
   TypeCode result;
   OperationMode mode;
   ContextIdSeq contexts;
   ParDescriptionSeq parameters;
   ExcDescriptionSeq exceptions;
   };
   struct Description {
   DefinitionKind kind;
   any value;
   };
   Description describe ();
```

Parameters

No input parameters are defined.

Return value

Description *

The returned value is a pointer to a CORBA::Contained::Description structure. The memory is owned by the caller and can be removed by invoking delete.

Exceptions

CORBA::SystemException

Remarks

The inherited describe operation returns a structure (CORBA::Contained::Description) that contains information about a CORBA::OperationDef Interface Repository object. The CORBA::Contained::Description structure has two fields: kind (CORBA::DefinitionKind data type), and value (CORBA::Any data type).

The kind of definition described by the returned structure is provided using the kind field, and the value field is a CORBA::Any that contains the description that is specific to the kind of object described. When the describe operation is invoked on an operation

(CORBA::OperationDef) object, the kind field is equal to CORBA::dk_Operation and the value field contains the CORBA::OperationDescription structure.

Example

```
// C++
   // assume that 'this_operation' has already been initialized
   CORBA::OperationDef * this_operation;

   // retrieve a description of the operation
   CORBA::OperationDef::Description * returned_description;
   returned_description = this_operation-> describe ();

   // retrieve the operation description from the returned description
   // structure
   CORBA::OperationDescription * operation_description;
   operation_description = (CORBA::OperationDescription *)
      returned_description value.value ();
```

OperationDef::exceptions

The exceptions read and write operations allow access and update of the list of exceptions associated with an operation definition (CORBA::OperationDef) within the Interface Repository.

Original interface

OperationDef Interface

IDL syntax

```
attribute ExceptionDefSeq exceptions;
```

Parameters

ExceptionDefSeq & exceptions

In Read operation, no input parameters are defined.

In Write operation, CORBA::ExceptionDefSeq & exceptions. The exceptions parameter is the sequence of exceptions with which to update the exceptions attribute of the operation definition (the sequence length may be set to zero to indicate no exceptions for the operation).

Return value

ExceptionDefSeq *

In Read operation, CORBA::ExceptionDefSeq *. The returned value is a pointer to a copy of the exceptions attribute of the operation definition. The memory is owned by the caller and can be removed by invoking delete.

Exceptions

CORBA::SystemException
CORBA::NO_IMPLEMENT thrown by Write operation on OS/390

Remarks

The exceptions attribute specifies the list of exception types that can be raised by the operation. The exceptions read and write operations are supported with parameter and result descriptions as defined below.

Example

```
// C++
   // assume that 'this_operation' and 'this_exception'
   // have already been defined
   CORBA::OperationDef * this_operation;
   CORBA::ExceptionDef * this_exception;

   // establish the exception definition sequence to update the operation
   CORBA::ExceptionDefSeq new_exceptions;
   new_exceptions.length (1);
   new_exceptions[0] = this_exception;
   this_operation-> exceptions (new_exceptions);

   // retrieve the exception list from the operation
   CORBA::ExceptionDefSeq * returned_exception_list;
   returned_exception_list = this_operation-> exceptions ();
```

OperationDef::mode

The mode read and write operations allow the access and update of the mode attribute of an operation definition (CORBA::OperationDef) within the Interface Repository.

Original interface

OperationDef Interface

IDL syntax

```
OperationMode mode;
```

Parameters

mode In Read operation, no input parameters are defined.

In Write operation, The mode parameter is the new value to which the mode attribute of the CORBA::OperationDef object will be set. Valid mode values include CORBA::OP_ONEWAY and CORBA::OP_NORMAL.

Return value

OperationMode

In Read operation, CORBA::OperationMode mode. The returned value is the current value of the mode attribute of the operation definition (CORBA::OperationDef) object.

In Write operation, no value is returned.

Exceptions

CORBA::SystemException
CORBA::NO_IMPLEMENT thrown by Write operation on OS/390

Remarks

The operation's mode attribute can be one of two values. If no output is returned by the operation, the operation is oneway (the mode attribute is CORBA::OP_ONEWAY), otherwise the operation is normal (the mode attribute is CORBA::OP_NORMAL).

The mode attribute can only be set to CORBA::OP_ONEWAY if the result is void and all of the operation parameters (the params attribute) are input only (have a mode of CORBA::PARAM_IN).

Example

```
// C++
   // assume that 'this_operation' has already been initialized
   CORBA::OperationDef * this_operation;

   // set the new mode in the operation definition
   CORBA::OperationMode new_mode = CORBA::OP_NORMAL;
   this_operation-> mode (new_mode);

   // retrieve the mode from the operation definition
   CORBA::OperationMode returned_mode;
   returned_mode = this_operation-> mode ();
```

OperationDef::params

The params read and write operations allow the access and update of the parameter descriptions of an operation definition object (CORBA::OperationDef) in the Interface Repository.

Original interface

OperationDef Interface

IDL syntax

```
attribute ParDescriptionSeq params;
```

Parameters

ParDescriptionSeq & params

In Read operation, no input parameters are defined.

In Write operation, CORBA::ParDescriptionSeq & params. The params parameter defines the new list of parameters that will comprise the parameters for the operation.

Return value

ParDescriptionSeq *

In Read operation, CORBA::ParDescriptionSeq*. The returned sequence of CORBA::ParameterDescriptions is a copy of the params attribute of the CORBA::OperationDef object. The memory is owned by the caller and can be removed by calling delete.

In Write operation, no values are returned.

Exceptions

CORBA::SystemException
CORBA::NO_IMPLEMENT thrown by Write operation on OS/390

Remarks

The params attribute describes the parameters of the operation.

The params attribute is a CORBA::ParDescriptionSeq data type, each element of which has 4 fields. The name (CORBA::Identifier) is the name of the parameter. The type field references a CORBA::TypeCode that represents the parameter type. The type_def field references a CORBA::IDLType that represents the parameter type definition. The mode field defines the parameter as an input parameter, an output parameter, or as used for both input and output (CORBA::PARAM_IN, CORBA::PARAM_OUT, and CORBA::PARAM_INOUT, respectively). The order of the elements in the sequence is important and should reflect the actual order of the parameters in the operation signature.

The params read and write operations are supported with the parameters and return values as defined below.

Example

```
// C++
   // assume that 'this_operation' and 'pk_long_ptr'
   // have already been initialized
   CORBA::OperationDef * this_operation;
   CORBA::PrimitiveDef * pk_long_ptr;

   // establish the CORBA::ParDescriptionSeq
   CORBA::ParDescriptionSeq seq_update;
   seq_update.length (1);
   seq_update[0].name = CORBA::string_dup ("parameter_0");
   seq_update[0].type = CORBA::_tc_long;
```

```
seq_update[0].type_def = CORBA::IDLType::_duplicate (pk_long_ptr);
seq_update[0].mode = CORBA::PARAM_IN;

// update the params attribute in the OperationDef
this_operation-> params (seq_update);

// retrieve the params attribute from the OperationDef
CORBA::ParDescriptionSeq * returned_parm_list;
returned_parm_list = this_operation-> params ();
```

OperationDef::result

The result read operation returns a type (CORBA::TypeCode *) representative of the value returned by the operation.

Original interface

OperationDef Interface

IDL syntax

```
readonly attribute TypeCode result;
```

Parameters

No input parameters are defined.

Return value

TypeCode *

The returned value is a pointer to a copy of the CORBA::TypeCode referenced by the result attribute. The memory is owned by the caller and can be returned using CORBA::release.

Exceptions

CORBA::SystemException

Remarks

The result attribute of a CORBA::OperationDef object references a CORBA::TypeCode * that describes the type of the value returned by the operation. The result read operation can be used to retrieve a pointer to a copy of the CORBA::TypeCode referenced by the result attribute.

Example

```
// C++
   // assume that 'this_operation' has already been initialized
   CORBA::OperationDef * this_operation;

   // retrieve the TypeCode which represents the type of result
```

```
                of the operation
    CORBA::TypeCode * operations_result_tc;
    operations_result_tc = this_operation-> result ();
```

OperationDef::result_def

The result_def read and write operations allow the access and update of the result type definition of an operation definition (CORBA::OperationDef) in the Interface Repository.

Original interface

OperationDef Interface

IDL syntax

```
attribute IDL/Type result_def;
```

Parameters

result_def

In Read operation, no input parameters are defined.

In Write operation, CORBA::IDLType * result_def. The result_def parameter represents the new result definition for the CORBA::OperationDef. Setting the result_def attribute also updates the result attribute.

Return value

IDLType *

In Read operation, CORBA::IDLType *. The returned object is a pointer to a copy of the CORBA::IDLType referenced by the result_def attribute of the CORBA::OperationDef object. The returned object is owned by the caller and can be released by invoking CORBA::release.

In Write operation, no value is returned.

Exceptions

CORBA::SystemException
CORBA::NO_IMPLEMENT thrown by Write operation on OS/390

Remarks

The type of the result of an operation definition is identified by the result_def attribute (a reference to a CORBA::IDLType *). Read and write result_def operations are supported, the parameter and return value definitions of which are defined below.

Example

```
// C++
   // assume that 'this_operation' and 'this_struct' have already been
   // initialized
```

```
CORBA::OperationDef * this_operation;
CORBA::StructDef * this_struct;

// change the operation result type definition to 'this_struct'
this_operation-> result_def (this_struct);

// read the operation's result type definition from 'this_operation'
CORBA::IDLType * returned_result_def;
returned_result_def = this_operation-> result_def ();
```

Policy Interface

This interface is not part of the programming model and should not be directly invoked or overridden.

PrimitiveDef Interface

The PrimitiveDef interface is used by the Interface Repository to represent one of the OMG IDL primitive data types.

File name

somir.idl

Intended usage

An instance of a PrimitiveDef object is used by the Interface Repository to represent an OMG IDL primitive data type. The OMG IDL primitive data types include CORBA::Null, CORBA::Void, CORBA::Short, CORBA::Long, CORBA::UShort, CORBA::ULong, CORBA::Float, CORBA::Double, CORBA::Boolean, CORBA::Char, CORBA::Octet, CORBA::Any, CORBA::TypeCode, CORBA::Principal, CORBA::String, CORBA::LongLong, CORBA::ULongLong,CORBA::Wstring, CORBA::Wchar, and CORBA::Objref.

The OMG IDL primitive data types CORBA::LongLong and CORBA::ULongLong are supported on AIX and Windows NT only.

PrimitiveDef objects are not named Interface Repository objects, and as such do not reside as named objects in the Interface Repository database. PrimitiveDef objects are used to create other Interface Repository objects (both named and un-named). An instance of an PrimitiveDef object can be created using the get_primitive operation of the Repository interface.

Local-only

True

Ancestor interfaces

IDLType Interface

Exceptions

CORBA::SystemException

IDL syntax

```
module CORBA
   {
      enum PrimitiveKind
      {
         pk_null, pk_void, pk_short, pk_long, pk_ushort, pk_ulong,
         pk_float, pk_double, pk_boolean, pk_char, pk_octet, pk_any,
         pk_TypeCode, pk_Principal, pk_string, pk_objref,
         pk_longlong, pk_ulonglong,     //supported on AIX and Windows NT
         pk_wchar, pk_wstring,
         pk_longdouble                  //not supported
      };
      interface PrimitiveDef:IDLType
      {
         readonly attribute PrimitiveKind kind;
      };
   };
```

Supported operations

PrimitiveDef::kind
IDLType::type

PrimitiveDef::kind

The kind read operation retrieves the kind of a primitive definition (CORBA::PrimitiveDef).

Original interface

PrimitiveDef Interface

IDL syntax

```
readonly attribute PrimitiveKind kind;
```

Parameters

No input parameters are defined.

Return value

PrimitiveKind

The returned value is the value of the kind attribute (CORBA::PrimitiveKind) of the CORBA::PrimitiveDef.

Exceptions

CORBA::SystemException

Remarks

The kind attribute indicates which primitive type is represented by a PrimitiveDef object. The valid values for the kind attribute that may be retrieved using the kind operation include CORBA::pk_short, CORBA::pk_long, CORBA::pk_ushort, CORBA::pk_ulong, CORBA::pk_float, CORBA::pk_double, CORBA::pk_boolean, CORBA::pk_char, CORBA::pk_wchar, and CORBA::pk_octet, that represent the basic kinds implied by the names.

Other kind values include: CORBA::pk_any (CORBA::Any data type), CORBA::pk_TypeCode (CORBA::TypeCode data type), CORBA::pk_Principal (CORBA::Principal data type), CORBA::pk_string (an unbounded string), CORBA::pk_wstring, and CORBA::pk_objref (CORBA::Object data type).

Example

```
// C++
   // assume that 'this_primitive' has already been initialized
   CORBA::PrimitiveDef * this_primitive;

   // retrieve the 'kind' of the PrimitiveDef
   CORBA::PrimitiveKind returned_kind;
   returned_kind = this_primitive-> kind ();
```

Principal Interface

This interface is not part of the programming model and should not be directly invoked or overridden.

Repository Interface

The Repository interface provides global access to the Interface Repository. As it inherits from Container, it can be used to look up any definition either by the name or by id (RepositoryId).

File name

somir.idl

Intended usage

The Repository object is a single instance object used to access member objects of the Interface Repository. The Repository object can directly contain constants (ConstantDef objects), type definitions (TypedefDef objects, including StructDef objects, UnionDef

objects, EnumDef objects, and AliasDef objects), exceptions (ExceptionDef objects), interfaces (InterfaceDef objects), and modules (ModuleDef objects).

Access to the Repository object is achieved by invoking the ORB operation resolve_initial_references.

Local-only

True

Ancestor interfaces

Container Interface

Exceptions

CORBA::SystemException

IDL syntax

```
module CORBA {
   interface Repository:Container {
   // read interface
   Contained lookup_id (in RepositoryId search_id);
   PrimitiveDef get_primitive (in PrimitiveKind kind);
   // write interface
   StringDef create_string (in unsigned long bound);
   WstringDef create_wstring (in unsigned long bound);
   SequenceDef create_sequence(
   in unsigned long bound,
   in IDLType element_type
   );
   ArrayDef create_array (
   in unsigned long length,
   in IDLType element_type
   );
   };
   };
```

Supported operations

Repository::create_array
Repository::create_sequence
Repository::create_string
Repository::create_wstring
Repository::get_primitive
Repository::lookup_id

Repository::create_array

The create_array operation is used to create a new array definition (ArrayDef).

Original interface

Repository Interface

IDL syntax

```
ArrayDef create_array (
   in unsigned long length,
   in IDLType element_type
   );
```

Parameters

length The length value specifies the length of the new ArrayDef.

element_type
The element_type is the IDLType representing each element of the ArrayDef.

Return value

ArrayDef_ptr
The return value is a pointer to the newly created ArrayDef object. The memory associated with the object is owned by the caller and can be released by invoking CORBA::release.

Exceptions

CORBA::SystemException

Remarks

The create_array operation returns a new ArrayDef with the specified length and element_type.

Example

```
// C++
   // create_array
   // assume that 'repository_ptr' and 'struct_1'
   // have already been initialized
   CORBA::Repository * repository_ptr;
   CORBA::StructDef * struct_1;

   // create an array definition with a bound of 409
   // and array element type of 'struct_1"
   CORBA::ArrayDef * array_def_ptr;
   CORBA::ULong array_length = 409;
   array_def_ptr = repository_ptr-> create_array (array_length, struct_1);
```

Repository::create_sequence

The create_sequence operation is used to create a new sequence definition (SequenceDef).

Original interface

Repository Interface

IDL syntax

```
SequenceDef create_sequence(
   in unsigned long bound,
   in IDLType element_type
   );
```

Parameters

bound The bound value represents the bound of the sequence definition. The bound value can be zero.

element_type
The element_type is the IDLType of the elements in the sequence.

Return value

SequenceDef_ptr
The return value is a pointer to the SequenceDef of the specified bound and element_type.

Exceptions

CORBA::SystemException

Remarks

The create_sequence operation returns a new SequenceDef with the specified bound and element_type.

Example

```
// C++
   // assume that 'repository_ptr' and 'struct_1' have already been
   // initialized
   CORBA::Repository * repository_ptr;
   CORBA::StructDef * struct_1;

   // create a sequence of 45 'struct_1' elements . . .
   CORBA::ULong bound_of_sequence = 45;
   CORBA::SequenceDef * sequence_def_ptr;
   sequence_def_ptr = repository_ptr-> create_sequence
                      (bound_of_sequence, struct_1);
```

Repository::create_string

The create_string operation is used to create a new StringDef to represent a bounded string.

Original interface

Repository Interface

IDL syntax

```
StringDef create_string (in unsigned long bound);
```

Parameters

bound The bound parameter represents the bound (the maximum number of characters in the string) of the bounded string. The value must be greater than zero.

Return value

StringDef_ptr
The returned value is a pointer to a CORBA::StringDef object with the specified bound. The memory associated with the object is owned by the caller and can be released by invoking CORBA::release.

Exceptions

CORBA::SystemException

Remarks

The create_string operation returns a new StringDef with the specified bound, that must be non-zero.

(Unbounded strings are represented by using the get_primitive operation to create a PrimitiveDef with a kind of CORBA::pk_string).

Example

```
// C++
   // assume that 'repository_ptr' has already been initialized
   CORBA::Repository * repository_ptr;

   // create a bounded string with a bound of 51
   CORBA::ULong bound_of_string = 51;
   CORBA::StringDef * string_def_ptr;
   string_def_ptr = repository_ptr-> create_string (bound_of_string);
```

Repository::create_wstring

The create_wstring operation is used to create a new WstringDef to represent a bounded wide string.

Not supported by OS/390 Component Broker

Original interface

Repository Interface

IDL syntax

```
StringDef create_wstring (in unsigned long bound);
```

Parameters

bound The bound parameter represents the bound (the maximum number of wide characters in the string) of the bounded wide string. The value must be greater than zero.

Return value

StringDef_ptr
The returned value is a pointer to a CORBA::WstringDef object with the specified bound. The memory associated with the object is owned by the caller and can be released by invoking CORBA::release.

Exceptions

CORBA::SystemException

Remarks

The create_wstring operation returns a new WstringDef with the specified bound, that must be non-zero.

(Unbounded wide strings are represented by using the get_primitive operation to create a PrimitiveDef with a kind of CORBA::pk_wstring).

Example

```
// C++
   // assume that 'repository_ptr' has already been initialized
   CORBA::Repository * repository_ptr;

   // create a bounded wide string with a bound of 51
   CORBA::ULong bound_of_wstring = 51;
   CORBA::WstringDef * wstring_def_ptr;
   wstring_def_ptr = repository_ptr-> create_wstring (bound_of_wstring);
```

Repository::get_primitive

The get_primitive operation is used to get a PrimitiveDef object with the specified kind attribute.

Original interface

Repository Interface

IDL syntax

```
PrimitiveDef get_primitive (in PrimitiveKind kind);
```

Parameters

kind The kind parameter indicates the kind of PrimitiveDef that is to be created. The valid values for kind include CORBA::pk_null, CORBA::pk_void, CORBA::pk_short, CORBA::pk_long, CORBA::pk_ushort, CORBA::pk_ulong, CORBA::pk_float, CORBA::pk_double, CORBA::pk_longlong, CORBA::pk_ulonglong,CORBA::pk_boolean, CORBA::pk_char, CORBA::pk_wchar, CORBA::pk_octet, CORBA::pk_any, CORBA::pk_TypeCode, CORBA::pk_Principal, CORBA::pk_string, CORBA::pk_wstring, and CORBA::pk_objref.

Return value

PrimitiveDef_ptr
The return value is a pointer to the new PrimitiveDef.

Exceptions

CORBA::SystemException

Remarks

The get_primitive operation returns a reference to a PrimitiveDef with the specified kind attribute. All PrimitiveDefs are immutable and owned by the Repository.

Example

```
// C++
   // assume 'repository_ptr' has already been initialized
   CORBA::Repository * repository_ptr;

   // create a PrimitiveDef to represent a CORBA::Long data type
   CORBA::PrimitiveDef * pk_long_def;
   pk_long_def = repository_ptr-> get_primitive (CORBA::pk_long);
```

Repository::lookup_id

The lookup_id operation is used to look up an object in a Repository given its RepositoryId.

Original interface

Repository Interface

IDL syntax

```
Contained lookup_id (in RepositoryId search_id);
```

Parameters

search_id
The search_id parameter is the unique CORBA::RepositoryId value of the Interface Repository object that is sought.

Return value

Contained *
The returned value is a pointer to a CORBA::Contained object that was retrieved from the Interface Repository. A nil object reference is returned if no object in the Interface Repository has the specified CORBA::RepositoryId.

Exceptions

CORBA::SystemException

Remarks

The lookup_id operation is used to retrieve an object from the Interface Repository based upon its unique CORBA::RepositoryId. If the Repository does not contain a definition for the search CORBA::RepositoryId, a nil object reference is returned.

Example

```
// C++
   // assume that 'interface_1' and 'repository_ptr' have already
      been initialized;
   CORBA::InterfaceDef * interface_1;
   CORBA::Repository * repository_ptr;

   // obtain the CORBA::RepositoryId for 'interface_1'
   CORBA::RepositoryId rep_id;
   rep_id = interface_1-> id();
   //  . . . .
   // retrieve the object from the Interface Repository database
   // using the CORBA::RepositoryId as the search key
```

```
CORBA::Contained * contained_ptr;
contained_ptr = repository_ptr-> lookup_id (rep_id);
```

Request Class

Represents a DII request.

File name

request.h

Intended usage

The Request class provides the primary support for the Dynamic Invocation Interface (DII), which allows client applications to dynamically build and invoke requests on objects. A Request object contains the following attributes:

target A CORBA::Object object that is the target of the request.

operation
The unscoped name of the IDL operation that is executed by the request.

arguments
A CORBA::NVList object that describes the types of all the IDL operation's parameters, the values of the operation's in and inout parameters, and the variables in which the out parameter values are stored after the request is invoked.

result A CORBA::NamedValue object that holds the result of the request after it is invoked.

env A CORBA::Environment object that describes the client environment associated with the request. If an exception is raised by the request, it is reported in the client environment.

exceptions
An optional CORBA::ExceptionList object that describes the user-defined exceptions that the DII operation can throw. This object is essentially a list of TypeCodes for UserException subclasses.

contexts
An optional CORBA::ContextList object that lists the context strings that are sent with the DII operation. A ContextList object differs from a Context object in that a ContextList supplies only the context strings whose values are transmitted with the request, while Context is the object from which those context string values are obtained.

ctx A CORBA::Context object that is passed when the request is invoked. For operations having no "context" clause in their IDL specification, this attribute is NULL.

The CORBA::Object::_create_request and CORBA::Object::_request methods can be used to create a Request object tailored to a specific IDL operation. These methods are invoked on the target object of the DII request. The _create_request method allows the Request object to be created and fully initialized at once. The _request method requires additional initialization after construction, using methods provided by the Request class. The Request class provides methods to get and set attributes, send a synchronous or asynchronous DII request, and receive the result of an asynchronous request. For additional information, see the CORBA::Object::_create_request and CORBA::Object::_request method descriptions.

Supported methods

Request::_duplicate
Request::_nil
Request::add_in_arg
Request::add_inout_arg
Request::add_out_arg
Request::arguments
Request::contexts
Request::ctx
Request::env
Request::exceptions
Request::get_response
Request::invoke
Request::operation
Request::poll_response
Request::result
Request::return_value
Request::send_deferred
Request::send_oneway
Request::set_return_type
Request::target

Request::_duplicate

Duplicates a Request object.

Original class

CORBA::Request

IDL syntax

```
static CORBA::Request_ptr _duplicate (CORBA::Request_ptr p);
```

Parameters

p The Request object to be duplicated. The reference can be nil, in which case the return value will also be nil.

Return value

CORBA::Request_ptr
The new Request object reference. This value should subsequently be released using CORBA::release(Request_ptr).

Remarks

This method is intended to be used by client and server applications to duplicate a reference to a Request object. Both the original and the duplicate reference should subsequently be released using CORBA::release(Request_ptr).

Request::_nil

Returns a nil CORBA::Request reference.

Original class

CORBA::Request

IDL syntax

```
static CORBA::Request_ptr _nil();
```

Return value

CORBA::Request_ptr
A nil Request reference.

Remarks

This method is intended to be used by client and server applications to create a nil Request reference.

Request::add_in_arg

Adds an input argument to the named value list for a DII request.

Original class

CORBA::Request

IDL syntax

```
CORBA::Any &add_in_arg();
CORBA::Any &add_in_arg(const char *name);
```

Parameters

name The name of the argument to be added. It is legal to pass a null pointer. If specified, the input name should match the argument name specified in the IDL definition for the operation.

Return value

CORBA::Any &

The value associated with the newly created named value, to be set by the caller with the value of the input argument.

Remarks

The add_in_arg method is used by a client program to populate the named value list associated with a CORBA::Request, which was created by calling CORBA::Object::_request. When called without a parameter, the add_in_arg method adds an element to the end of a CORBA::NVList by calling CORBA::NVList::add with argument passing mode CORBA::ARG_IN. When passed a string, the add_in_arg method adds an element to the end of a CORBA::NVList by calling CORBA::NVList::add_item with the input argument name and argument passing mode CORBA::ARG_IN.

Request::add_inout_arg

Adds an in/out argument to the named value list of a DII request.

Original class

CORBA::Request

IDL syntax

```
CORBA::Any &add_inout_arg();
CORBA::Any &add_inout_arg(const char *name);
```

Parameters

name The name of the argument to be added. It is legal to pass a null pointer. If specified, the input name should match the argument name specified in the IDL definition for the operation.

Return value

CORBA::Any &

The value associated with the newly created named value, to be set by the caller with the value of the input/output argument.

Remarks

The add_inout_arg method is used by a client program to populate the named value list associated with a CORBA::Request, which was created by calling CORBA::Object::_request. When called without a parameter, the add_in_arg method adds an element to the end of a CORBA::NVList by calling CORBA::NVList::add with argument passing mode CORBA::ARG_INOUT. When passed a string, the add_in_arg method adds an element to the end of a CORBA::NVList by calling CORBA::NVList::add_item with the input argument name and argument passing mode CORBA::ARG_INOUT.

Request::add_out_arg

Adds an output argument to the named value list of a DII request.

Original class

CORBA::Request

IDL syntax

```
CORBA::Any &add_out_arg();
CORBA::Any &add_out_arg(const char *name);
```

Parameters

name The name of the argument to be added. It is legal to pass a null pointer. If specified, the input name should match the argument name specified in the IDL definition for the operation.

Return value

CORBA::Any &
The value associated with the newly created named value, to be set by the caller with the value of the output argument.

Remarks

The add_out_arg method is used by a client program to populate the named value list associated with a CORBA::Request, which was created by calling CORBA::Object::_request. When called without a parameter, the add_out_arg method adds an element to the end of a CORBA::NVList by calling CORBA::NVList::add with argument passing mode CORBA::ARG_OUT. When passed a string, the add_out_arg method adds an element to the end of a CORBA::NVList by calling CORBA::NVList::add_item with the input argument name and argument passing mode CORBA::ARG_OUT.

Request::arguments

Retrieves the argument list of a DII request.

Original class

CORBA::Request

IDL syntax

```
CORBA::NVList_ptr arguments();
```

Return value

CORBA::NVList_ptr

A pointer to the argument list of the DII request, if any, or a null pointer. Ownership of the return value is maintained by the Request object; the return value must not be freed by the caller. If the DII request raises an exception, the values of the operation's out parameters are unpredictable.

Remarks

The arguments method is used by a client program to access the argument list of a Dynamic Invocation Interface (DII) request. The argument list is specified using a CORBA::NVList object and describes the types of all the IDL operation's parameters, the values of the operation's in and inout parameters, and the variables in which the out parmeter values are stored after the DII request is invoked. For additional information, see the NVList class description. The argument list of a Request object is set by the CORBA::Object::_create_request method.

Request::contexts

CORBA::NVList_ptr arguments();

Original class

CORBA::Request

IDL syntax

```
CORBA::ContextList_ptr contexts();
```

Return value

CORBA::ContextList_ptr

A pointer to the list of context strings that are sent with the DII request, as input to CORBA::Object::_create_request, or a null pointer. Ownership of the return result is maintained by the Request object; the return value must not be freed by the caller.

Remarks

The contexts method is used by a client application to access the list of context strings that are sent with a Dynamic Invocation Interface (DII) request, as optionally input to the CORBA::Object::create_request method. The context list is specified using a CORBA::ContextList object and is used to improve performance. When invoking a request without an associated ContextList object, the ORB looks up context infomation in the Interface Repository. For additional information, see the ContextList class description.

Request::ctx

Gets and sets the Context object associated with a DII request.

Original class

CORBA::Request

IDL syntax

```
void ctx(CORBA::Context_ptr p);
CORBA::Context_ptr ctx() const;
```

Parameters

new_context
A pointer to the new Context object to be associated with the DII request. If a Context object is already associated with the request, it is released. The caller retains ownership of this parameter (the Request object makes a duplicate). It is valid to pass a null pointer.

Return value

CORBA::Context_ptr
A pointer to the Context currently associated with the DII request, if any, or a null pointer. Ownership of the return value is maintained by the Request object; the return value must not be freed by the caller.

Remarks

The ctx method is used by a client application to get and set the list of properties that are sent with a Dynamic Invocation Interface (DII) request. The list of properties is specified using a CORBA::Context object and is used to pass information from the client environment to the server environment. For additional information, see the Context class description. The Context object associated with a DII request can also be set by the CORBA::Object::_create_request method.

Request::env

Retrieves the client environment associated with a DII request.

Original class

CORBA::Request

IDL syntax

```
CORBA::Environment_ptr env();
```

Return value

CORBA::Environment_ptr
A pointer to the client environment associated with a DII request. Ownership of the return value is maintained by the Request object; the return value must not be freed by the caller.

Remarks

The env method is used by a client application to access an exception raised by a Dynamic Invocation Interface (DII) request. The exception is held in a CORBA::Environment object, which is used for error handling in those cases where catch/throw exception handling cannot be used (such as DII). For additional information, see the Environment class description. The Environment object is automatically created by the CORBA::Object::_create_request or CORBA::Object::_request method used to construct the Request object.

Request::exceptions

Retrieves the list of user-defined exceptions that can be thrown by a DII request.

Original class

CORBA::Request

IDL syntax

```
CORBA::ExceptionList_ptr exceptions();
```

Return value

CORBA::ExceptionList_ptr
A pointer to the list of user-defined exceptions that can be thrown by the DII request, as input to CORBA::Object::_create_request, or a null pointer. Ownership of the return result is maintained by the Request object; the return value must not be freed by the caller.

Remarks

The exceptions method is used by a client application to access the list of user-defined exceptions that can be thrown by a Dynamic Invocation Interface (DII) request, as optionally input to CORBA::Object::_create_request. The exception list is specified using a CORBA::ExceptionList object and is used to improve performance. When invoking a request without an associated ExceptionList object, the ORB looks up user-defined exception information in the Interface Repository. For additional information, see the ExceptionList class description.

Request::get_response

Get the response from the request that is expected after a send_deferred has been issued.

Original class

CORBA::Request

IDL syntax

```
CORBA::Status get_response();
```

Return value

CORBA::Status
A zero return code indicates the response was successfuly received. A non-zero return code indicates failure.

Exceptions

CORBA::SystemException

Remarks

Use of the get_response method will result in a blocking call if the response hasn't been received yet. For client DII applications, use of the ORB::get_next_response method is recommended.

Request::invoke

Sends a synchronous DII request.

Original class

CORBA::Request

IDL syntax

```
CORBA::Status invoke();
```

Return value

CORBA::Status
A zero return code indicates the DII request was successfully sent and the response received. A non-zero return code indicates failure.

Remarks

The invoke method is used by a client application that is using the Dynamic Invocation Interface (DII), to issue a request. The invoke method blocks until a response is received. A Request object is constructed using the CORBA::ORB::_create_request or CORBA::ORB::_request method.

Request::operation

Retrieves the unscoped operation name of a DII request.

Original class

CORBA::Request

IDL syntax

```
const char * operation();
```

Return value

const char *
The unscoped operation name of the DII request. Ownership of the return value is maintained by the Request object; the return value must not be freed by the caller.

Remarks

The operation method is used by a client application to access the unscoped operation name of a Dynamic Invocation Interface (DII) request. The operation name of a Request object is set by either the CORBA::Object::_create_request or CORBA::Object::_request method.

Request::poll_response

Determines whether a response to an asynchronous request is available.

Original class

CORBA::Request

IDL syntax

```
CORBA::Boolean poll_response();
```

Return value

CORBA::Boolean
A zero return value indicates that a response is not available. A non-zero return value indicates that a response has been received.

Remarks

The poll_response method is used by a client application that is using the Dynamic Invocation Interface (DII), to determine whether a response is available, after sending one or more deferred requests (for example, using CORBA::Request::send_deferred). This method can be called prior to calling CORBA::Request::get_response, to avoid blocking.

Request::result

Retrieves the return value of a DII request.

Original class

CORBA::Request

IDL syntax

```
CORBA::NamedValue_ptr result();
```

Return value

CORBA::NamedValue_ptr
A pointer to the return value of the DII request. Ownership of the return value is maintained by the Request object; the return value must not be freed by the caller. If the DII request raises an exception, the return value is unpredictable.

Remarks

The result method is used by a client application to access the return value of a Dynamic Invocation Interface (DII) request. The return value is specified using a CORBA::NamedValue object and must not be accessed until after the request has been invoked. For additional information, see the NamedValue class description. The return value of a Request object is set by the CORBA::Request::invoke or CORBA::Request::get_response method, depending whether the request is synchronous or asynchronous, respectively.

Request::return_value

Retrurns the value of the return type of a DII request.

Original class

CORBA::Request

IDL syntax

```
CORBA::Any &return_value();
```

Return value

CORBA::Any &
The value of the return type of the DII request.

Remarks

The return_value method is used by a client program to access the value of the return type of a DII request. Specifically, the return_value method returns the CORBA::Any contained in the CORBA::NamedValue holding the return value of a DII request.

Request::send_deferred

Sends an asynchronous DII request.

Original class

CORBA::Request

IDL syntax

```
CORBA::Status send_deferred();
```

Return value

CORBA::Status
A zero return code indicates the DII request was successfully sent. A non-zero return code indicates failure.

Remarks

The send_deferred method is used by a client application that is using the Dynamic Invocation Interface (DII), to issue an asynchronous request. The results of an asynchronous request are later obtained using CORBA::Request::get_next_response. The CORBA:: Request::poll_response method is used to determine whether a response is available. A Request object is constructed using the CORBA::ORB::_create_request or CORBA::ORB::_request method.

Request::send_oneway

Sends a oneway DII request.

Original class

CORBA::Request

IDL syntax

```
CORBA::Status send_oneway();
```

Return value

CORBA::Status
A zero return code indicates the DII request was successfully sent. A non-zero return code indicates failure.

Remarks

The send_oneway method is used by a client application that is using the Dynamic Invocation Interface (DII), to issue a oneway request. A oneway request does not have any output parameters or return value. No response is sent back, so the client application must not call CORBA::Request::get_response. A Request object is constructed using the CORBA::ORB::_create_request or CORBA::ORB::_request method.

Request::set_return_type

Sets the return type for a DII request.

Original class

CORBA::Request

IDL syntax

```
CORBA::Void set_return_type(CORBA::TypeCode_ptr tc);
```

Parameters

tc A pointer to a CORBA::TypeCode representing the type of the operation return value. The caller retains ownership of the this parameter (the CORBA::Request makes its own copy).

Remarks

The set_return_type method is used by a client program to set the return type for a CORBA::Request, which was created by calling CORBA::Object::_request.

Request::target

Retrieves a pointer to the target object of a DII request.

Original class

CORBA::Request

IDL syntax

```
CORBA::Object_ptr target() const;
```

Return value

CORBA::Object_ptr
A pointer to the target object of the request. Ownership of the return value is maintained by the Request; the return value must not be freed by the caller.

Remarks

The target method is used by a client program to access the target object of a DII request. The target object of a Request object is automatically set by the Object::_create_request or Object::_request method. These methods are invoked on the target object of the DII request.

RequestSeq Class

Specifies a list of Requests.

File name

orb.h

Intended usage

Specifies a list of Requests to be sent in parallel using the Dynamic Invocation Interface. RequestSeq is used to specify the collection of requests sent by the ORB methods send_multiple_requests_oneway and send_multiple_requests_deferred. For additional information, see the Request and ORB class descriptions.

Supported methods

RequestSeq::allocbuf
RequestSeq::freebuf
RequestSeq::length
RequestSeq::maximum
RequestSeq::operator[]

RequestSeq::allocbuf

Allocates a sequence of pointers to Request elements.

Original class

CORBA::RequestSeq

IDL syntax

```
CORBA::Request_ptr allocbuf(CORBA::ULong nelems);
```

Parameters

nelems
The number of pointers to Request elements to be allocated. The requested number of elements must be greater than zero.

Return value

CORBA::Request_ptr *
A pointer to the address of the newly allocated sequence of pointers to Request elements. Ownership of the return value transfers to the caller. If the allocbuf method fails, a null pointer is returned.

Remarks

The allocbuf method is used by a client program to allocate a sequence of pointers to Request elements. The pointers are initialized to NULL. The newly allocated buffer can be passed to the "CORBA::Request_ptr *" constructor. Memory allocated using the allocbuf method must be freed using the freebuf method or by transferring ownership to a RequestSeq object.

RequestSeq::freebuf

Frees a sequence of Request elements.

Original class

CORBA::RequestSeq

IDL syntax

```
void freebuf(CORBA::Request_ptr *data);
```

Parameters

data A pointer to the sequence of Request elements to be freed. The freebuf method ignores null pointers passed to it.

Remarks

The freebuf method is used by a client program to free a sequence of Request elements. The release method is called on each Request element.

RequestSeq::length

Gets and sets the number of Request elements in a sequence.

Original class

CORBA::RequestSeq

IDL syntax

```
CORBA::ULong length() const;
void length(CORBA::ULong len);
```

Parameters

len The desired number of Request elements in the sequence.

Return value

CORBA::ULong
The current number of Request elements in the sequence.

Remarks

The length method is used by a client program to get and set the number of Requests in a collection of DII requests. Increasing the number of Request elements causes the sequence buffer to be reallocated. Decreasing the number of Request elements causes the orphaned Requests to be released.

RequestSeq::maximum

Retrieves the maximum number of Request elements in a sequence.

Original class

CORBA::RequestSeq

IDL syntax

```
CORBA::ULong maximum() const;
```

Return value

CORBA::ULong
The maximum number of Request elements in the sequence.

Remarks

The maximum method is used by a client program when querying the RequestSeq object associated with a collection of DII requests. For an unbounded sequence, the maximum method returns the number of Request elements currently allocated in a sequence. This tells applications the total amount of buffer space available. The application can then determine how many additional Request elements can be inserted before the buffer is reallocated. For a bounded sequence, the maximum method returns the maximum number of Request elements in a sequence as specified in the IDL type declaration. By definition, the maximum number of Request elements in a bounded sequence cannot be changed.

RequestSeq::operator[]

Retrieves the Request element at the specified index.

Original class

CORBA::RequestSeq

IDL syntax

```
CORBA::Request_SeqElem operator[] (CORBA::ULong index);
CORBA::Request_SeqElem operator[] (CORBA::ULong index) const;
```

Parameters

index The index corresponding to the desired Request element, starting at zero. A system exception is raised if the input index is greater than or equal to the number of elements in the sequence.

Return value

CORBA::Request_SeqElem
The Request element at the specified index. Ownership of the return value does not transfer to the caller. However, the sequence buffer may be owned by the caller prior to the method invocation.

Exceptions

CORBA::SystemException

Remarks

The subscript operator is called by a client program when querying the RequestSeq object associated with a collection of DII requests. The subscript operator returns the Request element at the specified index. Const and non-const versions of this operator are provided.

SequenceDef Interface

A SequenceDef object represents an OMG IDL sequence definition in the Interface Repository.

File name

somir.idl

Intended usage

An instance of a SequenceDef object is used by the Interface Repository to represent an OMG IDL bounded sequence data type.

SequenceDef objects are not named Interface Repository objects, and as such do not reside as named objects in the Interface Repository database (they are in a group of interfaces known as Anonymous types). An instance of a SequenceDef object can be created using the create_sequence operation of the Repository interface.

Local-only

True

Ancestor interfaces

IDLType Interface

Exceptions

CORBA::SystemException

IDL syntax

```
module CORBA {
   interface SequenceDef:IDLType {
   attribute unsigned longbound;
   readonlyattribute TypeCode element_type;
   attribute IDLType element_type_def;
   };
   };
```

Supported operations

SequenceDef::bound
SequenceDef::element_type

SequenceDef::element_type_def
IDLType::type

SequenceDef::bound

The bound read and write operations allow the access and update of the bound attribute of a sequence definition (CORBA::SequenceDef) within the Interface Repository.

Original interface

SequenceDef Interface

IDL syntax

```
attribute unsigned longbound;
```

Parameters

bound In Read operation, no input parameters are defined.

In Write operation, CORBA::ULong bound. The bound parameter is the new value to which the bound attribute of the CORBA::SequenceDef object is set.

Return value

ULong In Read operation, CORBA::ULong. The returned value is the current value of the bound attribute of the sequence definition (CORBA::SequenceDef) object.

In Write operation, no value is returned.

Exceptions

CORBA::SystemException
CORBA::NO_IMPLEMENT thrown by Write operation on OS/390

Remarks

The bound attribute specifies the maximum number of elements in the sequence. A bound of zero indicates an unbounded sequence. Read and write bound operations are supported with parameters as defined below.

Example

```
// C++
   // assume that 'this_sequence' has already been initialized
   CORBA::SequenceDef * this_sequence;

   // change the bound attribute of the sequence definition
   CORBA::ULong new_bound = 409;
   this_sequence-> bound (new_bound);
```

```
// obtain the bound of a sequence definition
CORBA::ULong returned_bound;
returned_bound = this_sequence-> bound ();
```

SequenceDef::element_type

The element_type operation returns a type (CORBA::TypeCode *) representative of the sequence element of a SequenceDef.

Original interface

SequenceDef Interface

IDL syntax

```
readonlyattribute TypeCode element_type;
```

Parameters

No input parameters are defined.

Return value

TypeCode *

The returned value is a pointer to a copy of the CORBA::TypeCode referenced by the element_type attribute. The memory is owned by the caller and can be returned by invoking CORBA::release.

Exceptions

CORBA::SystemException

Remarks

The element_type attribute of a SequenceDef object references a CORBA::TypeCode * that represents the type of the sequence element. The element_type read operation returns a copy of the CORBA::TypeCode referenced by the element_type attribute.

Example

```
// C++
   // assume that 'this_sequence' has already been initialized
   CORBA::SequenceDef * this_sequence;

   // retrieve the TypeCode which represents the type of the sequence
   // elements
   CORBA::TypeCode * sequence_element_type;
   sequence_element_type = this_sequence-> element_type
```

SequenceDef::element_type_def

The element_type_def read and write operation allow the access and update of the element type definition of a sequence definition (SequenceDef) in the Interface Repository.

Original interface

SequenceDef Interface

IDL syntax

```
attribute IDL/Type element_type_def;
```

Parameters

element_type_def

In Read operation, no input parameters are defined.

In Write operation, CORBA::IDLType_ptr element_type_def. The element_type_def parameter represents the new sequence element definitionfor the SequenceDef.

Return value

IDLType_ptr

In Read operation, CORBA::IDLType_ptr. The returned object is a pointer to a copy of the IDLType referenced by the element_type_def attribute of the SequenceDef object. The returned object is owned by the caller and can be released using CORBA::release.

In Write operation, no value is returned.

Exceptions

CORBA::SystemException
CORBA::NO_IMPLEMENT thrown by Write operation on OS/390

Remarks

The type of the elements within a sequence definition is identified by the element_type_def attribute (a reference to a CORBA::IDLType *). Setting the element_type_def attribute also updates the element_type attribute as well as the inherited type attribute.

Example

```
// C++
   // assume that 'this_sequence' and 'this_union' have already been
   // initialized
   CORBA::SequenceDef * this_sequence;
   CORBA::UnionDef * this_union;
```

```
// change the sequence element type definition to 'this_union'
this_sequence-> element_type_def (this_union);

// read the element type definition from 'this_sequence'
CORBA::IDLType * returned_element_type_def;
returned_element_type_def = this_sequence-> element_type_def ();
```

ServerRequest Class

Provides information about a request to be dispatched by a DynamicImplementation.

File name

request.h

Intended usage

The ServerRequest class is intended to be used by an implementation of a subclass of CORBA::BOA::DynamicImplementation, within the invoke method. The ServerRequest class is part of the DynamicSkeleton Interface (DSI), used primarily to create inter-ORB bridges or gateway servers.

The ServerRequest object provides information to the CORBA::BOA::DynamicImplementation::invoke method about the operation to be invoked and the in and inout parameter values. It also provides methods for recording the output and return values after the operation has been dispatched, so that the response can be sent back to the calling client.

Supported methods

ServerRequest::_duplicate
ServerRequest::_nil
ServerRequest::ctx
ServerRequest::exception
ServerRequest::op_def
ServerRequest::op_name
ServerRequest::params
ServerRequest::result

ServerRequest::_duplicate

Duplicates a ServerRequest object.

Original class

CORBA::ServerRequest

IDL syntax

```
static CORBA::ServerRequest_ptr _duplicate (CORBA::ServerRequest_ptr p);
```

Parameters

p The ServerRequest object to be duplicated. The reference can be nil, in which case the return value will also be nil.

Return value

CORBA::ServerRequest_ptr
The new ServerRequest object reference. This value should subsequently be released using CORBA::release(ServerRequest_ptr).

Remarks

This method is intended to be used by client and server applications to duplicate a reference to a ServerRequest object. The duplicate reference should subsequently be released using CORBA::release(ServerRequest_ptr).

ServerRequest::_nil

Returns a nil CORBA::ServerRequest reference.

Original class

CORBA::ServerRequest

IDL syntax

```
static CORBA::ServerRequest_ptr _nil ();
```

Return value

CORBA::ServerRequest_ptr
A nil ServerRequest reference.

Remarks

This method is intended to be used by client and server applications to create a nil ServerRequest reference.

ServerRequest::ctx

Provides the Context of an operation being invoked on a DynamicImplementation.

Original class

CORBA::ServerRequest

IDL syntax

```
CORBA::Context_ptr ctx() throw (CORBA::SystemException);
```

Return value

CORBA::Context_ptr
The Context of the operation that an implementation of CORBA::BOA::DynamicImplementation::invoke is dispatching. The ServerRequest retains ownership of the Context and the caller must not modify or free it.

Exceptions

CORBA::SystemException

Remarks

This method is intended to be used by an implementation of CORBA::BOA::DynamicImplementation::invoke (in a subclass of DynamicImplementation) to discover the Context of the request (if any). The IDL specification (for the operation being dispatched by CORBA::BOA::DynamicImplementation::invoke) indicates what Context identifiers are transmitted on each invocation of that operation.

ServerRequest::exception

Stores an exception in a ServerRequest.

Original class

CORBA::ServerRequest

IDL syntax

```
void exception (CORBA::Any *value)
     throw (CORBA::SystemException);
```

Parameters

value A CORBA::Any containing the exception to be stored in the ServerRequest. This exception is sent back to the client that originated the request. The ServerRequest assumes ownership of this Any.

Exceptions

CORBA::SystemException

Remarks

This method is intended to be called from an implementation of CORBA::BOA::DynamicImplementation::invoke (in a subclass of

DynamicImplementation) when an exception has been thrown by the operation being dispatched by the CORBA::BOA::DynamicImplementation::invoke method. This method can be called at most once by an execution of CORBA::BOA::DynamicImplementation::invoke, and only after CORBA::ServerRequest::params has been called. ServerRequest::exception may not be called if CORBA::ServerRequest::result has already been called. The ServerRequest object assumes ownership of the input Any object.

ServerRequest::op_def

Describes the signature of an operation being invoked on a DynamicImplementation.

Original class

CORBA::ServerRequest

IDL syntax

```
CORBA::OperationDef_ptr op_def()
       throw (CORBA::SystemException);
```

Return value

CORBA::OperationDef_ptr
A pointer to the OperationDef object from the Interface Repository that describes the operation being dispatched. The caller assumes ownership of this object.

Exceptions

CORBA::SystemException

Remarks

This method is intended to be used by an implementation of CORBA::BOA::DynamicImplementation::invoke (in a subclass of DynamicImplementation), to discover the signature of an operation being dispatched.

ServerRequest::op_name

Indicates the name of an operation being invoked on a DynamicImplementation.

Original class

CORBA::ServerRequest

IDL syntax

```
CORBA::Identifier op_name()
       throw (CORBA::SystemException);
```

Return value

Identifier (char *)

The (unscoped) IDL name of the operation being dispatched by an implementation of CORBA::BOA::DynamicImplementation::invoke. The ServerRequest retains ownership of this string and the caller must not modify it. For attribute accessor methods, the operation names are _get_<attribute> and _set_<attribute>. For operations introduced in CORBA::Object, the operation names are _interface, _implementation, _is_a, and _non_existent.

Exceptions

CORBA::SystemException

Remarks

This method is intended to be used by an implementation of CORBA::BOA::DynamicImplementation::invoke (in a subclass of DynamicImplementation), to discover which operation needs to be dispatched.

ServerRequest::params

Retrieves the in and inout parameter values of a ServerRequest.

Original class

CORBA::ServerRequest

IDL syntax

```
void params (CORBA::NVList_ptr parameters)
     throw (CORBA::SystemException);
```

Parameters

parameters

An NVList containing the TypeCodes (but not the values) for the parameters of the method being dispatched. On output, the NVList additionally contains the values of any in and inout parameters for that operation. This same NVList should subsequently be modified by the invoke() method (after dispatching the target method) to record the output parameter values. The ServerRequest assumes ownership of the NVList. If the operation has no parameters, an empty NVList can be passed.

Exceptions

CORBA::SystemException

Remarks

This method is intended to be used by an implementation of CORBA::BOA::DynamicImplementation::invoke (in a subclass of DynamicImplementation), to discover the the in and inout parameter values for the operation being dispatched. The caller supplies the types of the parameters via an NVList, and receives the parameter values in the same NVList. An implementation of CORBA::BOA::DynamicImplementation::invoke must invoke CORBA::ServerRequest::params exactly once.

ServerRequest::result

Record the return value of an operation invoked on a DynamicImplementation.

Original class

CORBA::ServerRequest

IDL syntax

```
void result (CORBA::Any *value)
     throw (CORBA::SystemException);
```

Parameters

value A CORBA::Any containing the return result of the operation invoked by an implementation of CORBA::BOA::DynamicImplementation::invoke. The ServerRequest assumes ownership of the Any.

Exceptions

CORBA::SystemException

Remarks

This method is intended to be used by an implementation of CORBA::BOA::DynamicImplementation::invoke (in a subclass of DynamicImplementation), to record the return result of an operation that was dispatched. If there is no return value (the return type is void or an exception occurred), CORBA::ServerRequest::result should not be called. The CORBA::ServerRequest::result method can be called at most once by an execution of CORBA::BOA::DynamicImplementation::invoke, and only after calling CORBA::ServerRequest::params.

StringDef Interface

The StringDef interface is used to represent an OMG IDL bounded string type.

File name

somir.idl

Intended usage

An instance of a StringDef object is used by the Interface Repository to represent an OMG IDL bounded string data type.

The StringDef object is not a named Interface Repository object (it is in a group of interfaces known as Anonymous types), and as such does not reside as a named object in the Interface Repository database. An instance of a StringDef object can be created using the create_string operation of the Repository interface.

Local-only

True

Ancestor interfaces

IDLType Interface

Exceptions

CORBA::SystemException

IDL syntax

```
module CORBA
   {
      interface StringDef:IDLType
      {
         attribute unsigned longbound;
      };
   };
```

Supported operations

StringDef::bound

StringDef::bound

The bound read and write operations allow the access and update of the bound attribute of a bounded string definition (CORBA::StringDef) within the Interface Repository.

Original interface

StringDef Interface

IDL syntax

```
attribute unsigned longbound;
```

Parameters

bound In Read operation, no input parameters are defined.

In Write operation, CORBA::ULong bound. The bound parameter is the new value to which the bound attribute of the CORBA::StringDef object is set.

Return value

ULong In Read operation, CORBA::ULong. The returned value is the current value of the bound attribute of the string definition (CORBA::StringDef) object.

In Write operation, no value is returned.

Exceptions

CORBA::SystemException
CORBA::NO_IMPLEMENT thrown by Write operation on OS/390

Remarks

The bound attribute specifies the maximum number of characters in the string, and must not be zero.

Example

```
// C++
  // assume that 'this_string' has already been initialized
  CORBA::StringDef * this_string;

  // change the bound attribute of the string definition
  CORBA::ULong new_bound = 409;
  this_string-> bound (new_bound);

  // obtain the bound of a string definition
  CORBA::ULong returned_bound;
  returned_bound = this_string-> bound ();
```

StructDef Interface

The StructDef interface is used to represent and OMG IDL structure definition.

File name

somir.idl

Intended usage

An instance of a StructDef object is used within the Interface Repository to represent an OMG IDL structure definition. An instance of a StructDef can be created using the create_struct operation of the Container interface.

Local-only

True

Ancestor interfaces

TypedefDef Interface

Exceptions

CORBA::SystemException

IDL syntax

```
module CORBA
   {
      struct StructMember
      {
         Identifier name;
         TypeCode type;
         IDLType type_def;
      };
      typedef sequence  StructMemberSeq;
      interface StructDef: TypedefDef
      {
         attribute StructMemberSeq members;
      };
   };
```

Supported operations

StructDef::members
IDLType::type

StructDef::members

The members read and write operations provide for the access and update of the list of elements of an OMG IDL structure definition in the Interface Repository.

Original interface

StructDef Interface

IDL syntax

```
attribute StructMemberSeq members;
```

Parameters

StructMemberSeq & members

In Read operation, no input parameters are defined.

In Write operation, CORBA::StructMemberSeq & members. The members parameter provides the list of structure members with which to update the StructDef.

Return value

StructMemberSeq *

In Read operation, CORBA::StructMemberSeq *. The returned pointer references a sequence that is representative of the structure members. The memory is owned by the caller and can be released by invoking delete.

Exceptions

CORBA::SystemException

Remarks

The members attribute contains a description of each structure member. The members read and write operations allow the access and update of the members attribute.

This method is implemented in the OS/390 IR for use by IDL compiler emitted IR install client program. Use of this method in a generalized programming environment is strongly discouraged since it could jeopardize coherency of the IR. In general, the IR install client programs should be the only means for updating the contents of the Interface Repository.

Example

```
// C++
   /// assume 'this_struct_def', 'pk_long_ptr', and 'pk_double_ptr'
   // have already been initialized
   CORBA::StructDef * this_struct_def;
   CORBA::PrimitiveDef * pk_long_ptr;
   CORBA::PrimitiveDef * pk_double_ptr;

   // establish and initialize the StructMemberSeq . . .
   CORBA::StructMemberSeq seq_update;
   seq_update.length (2);
   seq_update[0].name = CORBA::string_dup ("element_zero_long");
   seq_update[0].type_def = CORBA::IDLType::_duplicate (pk_long_ptr);
   seq_update[1].name = CORBA::string_dup ("element_one_double");
   seq_update[1].type_def = CORBA::IDLType::_duplicate (pk_double_ptr);

   // set the members attribute of the StructDef
   this_struct_def-> members (seq_update);

   // read the members attribute information from the StructDef
   CORBA::StructMemberSeq * returned_members;
   returned_members = this_struct_def-> members ();
```

SystemException Class

Describes a system exception condition that has occurred.

File name

sys_excp.h

Intended usage

This class is intended to be caught in the catch clause of a try/catch block that encompasses remote method invocations or calls to ORB services. Instances of SystemException subclasses (see list below) can be thrown from implementations of IDL interfaces, to indicate some exception condition not related to the application logic (for example, unavailable memory). SystemExceptions can be thrown by any operation, regardless of the interface specification (that is, its “raises” clause in IDL).

Each SystemException contains a minor error code, to designate the subcategory of the exception, and a completion status (COMPLETED_YES, COMPLETED_NO, or COMPLETED_MAYBE) to indicate whether the object completed processing the request prior to the exception being thrown. The SystemException class provides a non-default constructor whose arguments are the minor code of the exception (of type CORBA::ULong) and the completion status (of type CORBA::CompletionStatus). When the default constructor is used, the completion status defaults to COMPLETED_NO and the minor code defaults to zero.

The subclasses of SystemException, representing specific error conditions, are defined in std_excp.h as follows:

- BAD_CONTEXT;
- BAD_INV_ORDER;
- BAD_OPERATION;
- BAD_PARAM;
- BAD_TYPECODE;
- COMM_FAILURE;
- DATA_CONVERSION;
- FREE_MEM;
- IMP_LIMIT;
- INITIALIZE;
- INTERNAL;
- INTF_REPOS;
- INVALID_SESSION;
- INVALID_TRANSACTION;
- INV_FLAG;
- INV_IDENT;
- INV_OBJREF;
- MARSHAL;
- NO_IMPLEMENT;
- NO_MEMORY;
- NO_PERMISSION;
- NO_RESOURCES;

- NO_RESPONSE;
- OBJECT_NOT_EXIST;
- OBJ_ADAPTER;
- PERSIST_STORE;
- SESSION_REQUIRED;
- SESSION_RESET_FORCED;
- TRANSACTION_REQUIRED;
- TRANSACTION_ROLLEDBACK;
- TRANSIENT;
- UNKNOWN;

Each subclass of SystemException has corresponding release, is_nil, _duplicate, and _nil methods, and a non-default constructor that mirrors the SystemException non-default constructor.

In the Java implementation, org.omg.CORBA.SystemException derives form java.lang.RuntimeException.

Supported methods

SystemException::_duplicate
SystemException::_nil
SystemException::completed
SystemException::minor

SystemException::_duplicate

Duplicates a SystemException object.

Original class

CORBA::SystemException

IDL syntax

```
static CORBA::SystemException_ptr _duplicate
              (CORBA::SystemException_ptr p);
```

Parameters

p The SystemException object to be duplicated. The reference can be nil, in which case the return value will also be nil.

Return value

CORBA::SystemException_ptr
The new SystemException object reference. This value should subsequently be released using CORBA::release(SystemException_ptr).

Remarks

This method is intended to be used by client and server applications to duplicate a reference to a SystemException object. Both the original and the duplicate reference should subsequently be released using CORBA::release(SystemException_ptr).

SystemException::_nil

Returns a nil CORBA::SystemException reference.

Original class

CORBA::SystemException

IDL syntax

```
static CORBA::SystemException_ptr _nil ();
```

Return value

CORBA::SystemException_ptr
A nil SystemException reference.

Remarks

This method is intended to be used by client and server applications to create a nil SystemException reference.

SystemException::completed

Indicates whether an operation completed before an exception was encountered.

Original class

CORBA::SystemException

IDL syntax

```
CORBA::CompletionStatus completed() const;
void completed(CORBA::CompletionStatus status);
```

Parameters

status The completion status to store in the SystemException.

Return value

CORBA::CompletionStatus
A value indicating whether the operation that threw the SystemException

completed before the exception was encountered (CORBA::COMPLETED_YES, CORBA::COMPLETED_NO, or CORBA::COMPLETED_MAYBE).

Remarks

The first completed method is intended to be used by a client or server application after catching a SystemException in a try/catch block, to determine whether the operation that threw the SystemException completed before the exception was encountered.

The second completed method is used to set the completion status of a SystemException before throwing it.

SystemException::minor

Indicates the minor error code of a SystemException.

Original class

CORBA::SystemException

IDL syntax

```
CORBA::ULong minor() const;
void minor (CORBA::ULong min_id);
```

Parameters

min_id The minor code to store in the SystemException.

Return value

CORBA::ULong
A value indicating the minor error code contained in the SystemException.

Remarks

The first minor method is intended to be used by a client or server application after catching a SystemException in a try/catch block, to determine the minor error code.

The second minor method is used to set the minor code of a SystemException before throwing it.

TypeCode Class

Represents an OMG IDL type.

File name

typecode.h

Intended usage

A TypeCode represents an OMG IDL type. A TypeCode is an integral part of the any type and is used to specify the type of the value. The Interface Repository also uses TypeCodes to store information about types declared in IDL.

A TypeCode consists of a “kind” field and zero or more parameters to fully describe the underlying data type. For example, the TypeCode describing IDL type char has kind tk_char and no parameters. The TypeCode describing the IDL type array has kind tk_array and two parameters, a TypeCode describing the type of elements in the array and a long indicating the length of the array. CORBA::ORB provides methods to create complex TypeCodes (TypeCodes which have parameters). The naming convention for these methods is create_<type>_tc. For example, the method create_array_tc creates a tk_array TypeCode.

Methods are provided to access the various parts of a TypeCode. Since the structure of a TypeCode varies, most methods are only applicable to certain TypeCodes. The BadKind exception is raised if a method is not applicable to the target TypeCode. Methods that deal with indexing raise the Bounds exception if the input index is greater than or equal to the number of members constituting the type.For additional information, see the Any and ORB class descriptions.

Exceptions

BadKind – an operation is not appropriate for the TypeCode kind.
Bounds – index parameter is greater than or equal to the number of members constituting the type.

Supported methods

TypeCode::_duplicate
TypeCode::_nil
TypeCode::content_type
TypeCode::default_index
TypeCode::discriminator_type
TypeCode::equal
TypeCode::id
TypeCode::kind
TypeCode::length
TypeCode::member_count
TypeCode::member_label
TypeCode::member_name
TypeCode::member_type
TypeCode::name

TypeCode::_duplicate

Duplicates a TypeCode object.

Original class

CORBA::TypeCode

IDL syntax

```
static CORBA::TypeCode_ptr _duplicate (CORBA::TypeCode_ptr p);
```

Parameters

p The TypeCode object to be duplicated. The reference can be nil, in which case the return value will also be nil.

Return value

CORBA::TypeCode_ptr
The new TypeCode object reference. This value should subsequently be released using CORBA::release(TypeCode_ptr).

Remarks

This method is intended to be used by client and server applications to duplicate a reference to a TypeCode object. Both the original and the duplicate reference should subsequently be released using CORBA::release(TypeCode_ptr).

TypeCode::_nil

Returns a nil CORBA::TypeCode reference.

Original class

CORBA::TypeCode

IDL syntax

```
static CORBA::TypeCode_ptr _nil ();
```

Return value

CORBA::TypeCode_ptr
A nil TypeCode reference.

Remarks

This method is intended to be used by client and server applications to create a nil TypeCode reference.

TypeCode::content_type

Returns the element type of a sequence or array, or the original type of an alias.

Original class

CORBA::TypeCode

IDL syntax

```
CORBA::TypeCode_ptr content_type() const;
```

Return value

CORBA::TypeCode_ptr
A pointer to the element type of the sequence or array, or the original type of the alias. Ownership of the return value transfers to the caller and must be freed by calling CORBA::release(CORBA::TypeCode_ptr).

Exceptions

CORBA::SystemException
CORBA::TypeCode::BadKind

Remarks

The content_type method can be invoked on sequence, array, and alias TypeCodes. For sequences and arrays, the content_type method returns the element type. For aliases, the content_type method returns the original type.

TypeCode::default_index

Returns the index of the default union member.

Original class

CORBA::TypeCode

IDL syntax

```
CORBA::Long default_index() const;
```

Return value

CORBA::Long
The index of the default union member.

Exceptions

CORBA::SystemException
CORBA::TypeCode::BadKind

Remarks

The default_index method can only be invoked on union TypeCodes. The default_index method returns the index of the default member, or -1 if there is no default member.

TypeCode::discriminator_type

Returns the discriminator TypeCode of a union.

Original class

CORBA::TypeCode

IDL syntax

```
CORBA::TypeCode_ptr discriminator_type() const;
```

Return value

CORBA::TypeCode_ptr
A pointer to the discriminator TypeCode of the union. Ownership of the return value transfers to the caller and must be freed by calling CORBA::release(CORBA::TypeCode_ptr).

Exceptions

CORBA::SystemException
CORBA::TypeCode::BadKind

Remarks

The discriminator_type method can only be invoked on union TypeCodes. The discriminator_type method returns the type of all non-default member labels.

TypeCode::equal

Compares two TypeCodes for equality.

Original class

CORBA::TypeCode

IDL syntax

```
CORBA::Boolean equal(CORBA::TypeCode_ptr tc) const;
```

Parameters

tc A pointer to the TypeCode to be compared against the target TypeCode.

Return value

CORBA::Boolean
A return value of one indicates that the input TypeCode and the target TypeCode are equal. A return value of zero indicates the TypeCodes are not equal.

Exceptions

CORBA::SystemException

Remarks

The equal method can be invoked on any TypeCode. The equal method is used to determine whether two distinct TypeCodes describe the same underlying abstract data type. Equivalent TypeCodes produce the same results when TypeCode methods are invoked on them.

TypeCode::id

Returns the Interface Repository identifier of an interface, structure, union, enumeration, alias, or exception.

Original class

CORBA::TypeCode

IDL syntax

```
const char * id() const;
```

Return value

const char *
The Interface Repository identifier of the interface, structure, union, enumeration, alias, or exception. Ownership of the return value is maintained by the TypeCode; the return value must not be freed by the caller.

Exceptions

CORBA::SystemException
CORBA::TypeCode::BadKind

Remarks

The id method can be invoked on interface, structure, union, enumeration, alias, and exception TypeCodes. The id method returns the RepositoryId of a type.

TypeCode::kind

Categorizes the abstract data type described by a TypeCode.

Original class

CORBA::TypeCode

IDL syntax

```
CORBA::TCKind kind() const;
```

Return value

CORBA::TCKind
TCKind enumeration value.

Exceptions

CORBA::SystemException

Remarks

The kind method can be invoked on all TypeCodes. The kind method is used to classify a TypeCode into one of the categories listed in the TCKind enumeration. Based on the ″kind″ classification, a TypeCode may contain zero or more additional parameters to fully describe the underlying data type. See the TypeCode class description for a list of legal TypeCode kinds and parameters.

TypeCode::length

Returns the bound of a string or sequence, or the number of elements in an array.

Original class

CORBA::TypeCode

IDL syntax

```
CORBA::ULong length() const;
```

Return value

CORBA::ULong
The bound of the string or sequence, the number of elements in the array.

Exceptions

CORBA::SystemException
CORBA::TypeCode::BadKind

Remarks

The length method can be invoked on string, sequence, and array TypeCodes. For strings and sequences, the length method returns a bound, with zero indicating an unbounded string or sequence. For arrays, the length method returns the number of elements in an array.

TypeCode::member_count

Returns the number of members in a structure, union, enumeration, or exception.

Original class

CORBA::TypeCode

IDL syntax

```
CORBA::ULong member_count() const;
```

Return value

CORBA::ULong
The number of members in the structure, union, enumeration, or exception.

Exceptions

CORBA::SystemException
CORBA::TypeCode::BadKind

Remarks

The member_count method can be invoked on structure, union, enumeration, and exception TypeCodes. The member_count method returns the number of members constituting the type.

TypeCode::member_label

Returns the label of a union member.

Original class

CORBA::TypeCode

IDL syntax

```
CORBA::Any_ptr member_label(CORBA::ULong index) const
```

Parameters

index The index of the desired union member, starting at zero.

Return value

CORBA::Any_ptr
A pointer to the label of the union member. Ownership of the return value transfers to the caller and must be freed by calling CORBA::release(CORBA::TypeCode_ptr).

Exceptions

CORBA::SystemException
CORBA::TypeCode::BadKind
CORBA::TypeCode::Bounds

Remarks

The member_label method can only be invoked on union TypeCodes. The member_label method returns the label of the member identified by index. For the default member, the label is the zero octet.

TypeCode::member_name

Returns the simple name of a structure, union, enumeration, or exception member.

Original class

CORBA::TypeCode

IDL syntax

```
const char * member_name(CORBA::ULong index) const;
```

Parameters

index The index of the desired member, starting at zero.

Return value

const char *

The simple name of the structure, union, enumeration, or exception member. Ownership of the return value is maintained by the TypeCode; the return value must not be freed by the caller.

Exceptions

CORBA::SystemException
CORBA::TypeCode::BadKind
CORBA::TypeCode::Bounds

Remarks

The member_name method can be invoked on structure, union, enumeration, and exception TypeCodes. The member_name method returns the simple name of the member identified by index.

TypeCode::member_type

Returns the type of a structure, union, or exception member.

Original class

CORBA::TypeCode

IDL syntax

```
CORBA::TypeCode_ptr member_type(CORBA::ULong index) const;
```

Parameters

index The index of the desired member, starting at zero.

Return value

CORBA::TypeCode_ptr

A pointer to the type of the structure, union, or exception member. Ownership of the return value transfers to the caller and must be freed by calling CORBA::release(CORBA::TypeCode_ptr).

Exceptions

CORBA::SystemException
CORBA::TypeCode::BadKind
CORBA::TypeCode::Bounds

Remarks

The member_type method can be invoked on structure, union, and exception TypeCodes. The member_type method returns the TypeCode describing the type of the member identified by index.

TypeCode::name

Returns the simple name of an interface, structure, union, enumeration, alias, or exception.

Original class

CORBA::TypeCode

IDL syntax

```
const char * name() const;
```

Return value

const char *

The simple name of the interface, structure, union, enumeration, alias, or exception. Ownership of the return value is maintained by the TypeCode; the return value must not be freed by the caller.

Exceptions

CORBA::SystemException
CORBA::TypeCode::BadKind

Remarks

The name method can be invoked on object reference, structure, union, enumeration, alias, and exception TypeCodes. The name method returns the simple name identifying the type within its enclosing scope.

TypedefDef Interface

The TypedefDef interface is an abstract interface used by the Interface Repository as a base interface to represent data types including structures, unions, enumerations, and aliases.

File name

somir.idl

Intended usage

The TypedefDef interface is not itself instantiated as a means of accessing the Interface Repository. As an ancestor to Interface Repository objects that represent OMG IDL data types, it provides a specific operation as noted below. Those Interface Repository objects that inherit (directly or indirectly) the operation defined in TypedefDef include: StructDef, UnionDef, EnumDef, and AliasDef.

Local-only

True

Ancestor interfaces

Contained Interface
IDLType Interface

Exceptions

CORBA::SystemException

IDL syntax

```
module CORBA
   {
      interface TypedefDef:Contained, IDLType
      {
      };
      struct TypeDescription
      {
         Identifier name;
         RepositoryId id;
         RepositoryId defined_in;
         VersionSpec version;
         TypeCode type;
      }
   }
```

Supported operations

TypedefDef::describe

TypedefDef::describe

The describe operation returns a structure containing information about a CORBA::TypedefDef Interface Repository object.

Original interface

TypedefDef Interface

IDL syntax

```
struct TypeDescription
   {
      Identifier name;
      RepositoryId id;
      RepositoryId defined_in;
      VersionSpec version;
      TypeCode type;
   };
   struct Description
   {
      DefinitionKind kind;
      any value;
   };
   Description describe ();
```

Parameters

No input parameters are defined.

Return value

Description *

The returned value is pointer in a CORBA::Contained::Description structure. The memory is owned by the caller and can be removed by invoking delete.

Exceptions

CORBA::SystemException

Remarks

The inherited describe operation returns a structure (CORBA::Contained::Description) that contains information about a CORBA::TypedefDef Interface Repository object. The CORBA::Contained::Description structure has two fields: kind (CORBA::DefinitionKind data type), and value (CORBA::Any data type).

The kind of definition described by the returned structure is provided using the kind field, and the value field is a CORBA::Any that contains the description that is specific to the kind of object described. When the describe operation is invoked on a type definition (CORBA::TypedefDef) object, the kind field is representative of the specific type of CORBA::TypedefDef (either CORBA::dk_Union, CORBA::dk_Struct, CORBA::dk_Alias, or CORBA::dk_Enum). The value field contains the CORBA::TypeDescription structure.

Example

```
// C++
   // assume that 'this_union' has already been initialized
   CORBA::UnionDef * this_union;

   // retrieve a description of the union
   CORBA::UnionDef::Description * returned_description;
```

```
returned_description = this_union-> describe ();

// retrieve the type definition description from the returned
// description structure
CORBA::TypeDescription * type_description;
type_description = (CORBA::TypeDescription *) returned_description
                    value.value ();
```

UnionDef Interface

The UnionDef interface is used within the Interface Repository to represent an OMG IDL union definition.

File name

somir.idl

Intended usage

An instance of a UnionDef object is used within the Interface Repository to represent an OMG IDL union definition. An instance of a UnionDef object can be created using the create_union operation of the Container interface.

Local-only

True

Ancestor interfaces

TypedefDef Interface

Exceptions

CORBA::SystemException

IDL syntax

```
module CORBA
   {
      struct UnionMember
      {
         Identifier name;
         anylabel;
         Typecode type;
         IDLType type_def;
      };
      typedef sequence  UnionMemberSeq;
      interface UnionDef:TypedefDef
      {
         readonlyattribute TypeCode discriminator_type;
```

```
        attribute IDLType discriminator_type_def;
        attribute UnionMemberSeq members;
      };
   };
```

Supported operations

UnionDef::discriminator_type
UnionDef::discriminator_type_def
UnionDef::members
IDLType::type

UnionDef::discriminator_type

The discriminator_type operation returns TypeCode information representative of the discriminator of an Interface Repository UnionDef object.

Original interface

UnionDef Interface

IDL syntax

```
readonly attribute TypeCode discriminator_type;
```

Parameters

No parameters defined.

Return value

TypeCode_ptr
The returned value is a pointer to a TypeCode that represents the type of the union discriminator. The memory is owned by the caller and can be released by invoking CORBA::release.

Exceptions

CORBA::SystemException

Remarks

The discriminator_type attribute describes and identifies the union's discriminator type. The discriminator_type attribute can be accessed using the discriminator_type read operation. The discriminator_type attribute can only be changed by updating the discriminator_type_def attribute.

Example

```
// C++
   // assume that 'this_union' has already been initialized
   CORBA::UnionDef * this_union;
```

```
// retrieve the TypeCode information that represents
// the union discriminator
CORBA::TypeCode * unions_discriminator_tc;
unions_discriminator_tc = this_union-> discriminator_type();
```

UnionDef::discriminator_type_def

The discriminator_type_def read and write operations allow access and update of the discriminator_type_def attribute of an Interface Repository UnionDef object.

Original interface

UnionDef Interface

IDL syntax

```
attribute IDLType discriminator_type_def;
```

Parameters

discriminator_type_def

In Read operation, no input parameters are defined.

In Write operation, CORBA::IDLType_ptr discriminator_type_def. The discriminator_type_def must be of a subset of the simple types (a PrimitiveDef of kind CORBA::pk_long, CORBA::pk_ulong, CORBA::pk_short, CORBA::pk_ushort, CORBA::pk_boolean, CORBA::pk_wchar, or CORBA::pk_char) or an enumeration definition (EnumDef). Setting the discriminator_type_def also updates the discriminator_type attribute.

Return value

IDLType_ptr

In Read operation, CORBA::IDLType_ptr. The returned value is a pointer to a copy of the IDLType that represents the discriminator_type_def attribute. The memory is owned by the caller and can be released by invoking CORBA::release.

In Write operation, no value is returned.

Exceptions

CORBA::SystemException

Remarks

The discriminator_type_def attribute references an IDLType that is a type definition for the discriminator of a union. Both read and write operations are supported with parameters as defined above.

This method is implemented in the OS/390 IR for use by IDL compiler emitted IR install client program. Use of this method in a generalized programming environment is strongly discouraged since it could jeopardize coherency of the IR. In general, the IR install client programs should be the only means for updating the contents of the Interface Repository.

Example

```
// C++
   // assume that 'this_union' and 'pk_long_ptr'
   // have already been initialized
   CORBA::UnionDef * this_union;
   CORBA::PrimitiveDef * pk_long_ptr;

   // set the discriminator_type_def to represent a CORBA::Long data type
   this_union-> discriminator_type_def (pk_long_ptr);

   // retrieve the discriminator_type_def information from the UnionDef
   // object
   CORBA::IDLType * ret_idltype_ptr;
   ret_idltype_ptr = this_union-> discriminator_type_def ();
```

UnionDef::members

The members read and write operations provide for the access and update of the list of elements of an OMG IDL union definition in the Interface Repository.

Original interface

UnionDef Interface

IDL syntax

```
attribute UnionMemberSeq members;
```

Parameters

UnionMemberSeq & members

In Read operation, no input parameters are defined.

In Write operation, CORBA::UnionMemberSeq & members. The members parameter provides the list of union members with which to update the UnionDef.

Return value

UnionMemberSeq *

In Read operation, CORBA::UnionMemberSeq *. The returned pointer references a sequence that is representative of the union members. The memory is owned by the caller and can be released by invoking delete.

Exceptions

CORBA::SystemException

Remarks

The members attribute contains a description of each union member. The members read and write operations allow the access and update of the members attribute.

This method is implemented in the OS/390 IR for use by IDL compiler emitted IR install client program. Use of this method in a generalized programming environment is strongly discouraged since it could jeopardize coherency of the IR. In general, the IR install client programs should be the only means for updating the contents of the Interface Repository.

Example

```
// C++
  // assume 'this_union_def', 'pk_long_ptr', and 'pk_double_ptr'
  // have already been initialized
  CORBA::UnionDef * this_union_def;
  CORBA::PrimitiveDef * pk_long_ptr;
  CORBA::PrimitiveDef * pk_double_ptr;

  // establish and initialize the UnionMemberSeq . . .
  CORBA::UnionMemberSeq seq_update;
  seq_update.length (2);
  seq_update[0].name = CORBA::string_dup ("element_zero_long");
  seq_update[0].label <<= (CORBA::Long) 1;
  seq_update[0].type_def = CORBA::IDLType::_duplicate (pk_long_ptr);


  seq_update[1].name = CORBA::string_dup ("element_one_double");
  seq_update[1].label <<= (CORBA::Long) 2;
  seq_update[1].type_def = CORBA::IDLType::_duplicate (pk_double_ptr);

  // set the members attribute of the UnionDef using 'seq_update'
  this_union_def-> members (seq_update);

  // read the members attribute information from the UnionDef
  CORBA::UnionMemberSeq * returned_members;
  returned_members = this_union_def-> members ();
```

UnknownUserException Class

Describes a generic application-specific exception condition that has occurred.

File name

ukn_excp.h

Intended usage

Request invocations made through the Dynamnic Invocation Interface (DII) may result in user-defined exceptions that cannot be represented in the client program (because the exception type was not known at compile time). The CORBA::UnknownUserException class is intended to be caught in the catch clause of a try/catch block that encompasses a DII invocation.

Applications should never explicitly thrown instances of CORBA::UnknownUserException.

Supported methods

UnknownUserException::_duplicate
UnknownUserException::_nil
UnknownUserException::exception
UnknownUserException::id (inherited from Exception)

UnknownUserException::_duplicate

Duplicates an UnknownUserException object.

Original class

CORBA::UnknownUserException

IDL syntax

```
static CORBA::UnknownUserException_ptr _duplicate
       (CORBA::UnknownUserException_ptr p);
```

Parameters

p The UnknownUserException object to be duplicated. The reference can be nil, in which case the return value will also be nil.

Return value

CORBA::UnknownUserException_ptr
The new UnknownUserException object reference. This value should subsequently be released using CORBA::release(UnknownUserException_ptr).

Remarks

This method is intended to be used by client and server applications to duplicate a reference to an UnknownUserException object. Both the original and the duplicate reference should subsequently be released using CORBA::release(UnknownUserException_ptr).

UnknownUserException::_nil

Returns a nil CORBA::UnknownUserException reference.

Original class

CORBA::UnknownUserException

IDL syntax

```
static CORBA::UnknownUserException_ptr _nil ();
```

Return value

CORBA::UnknownUserException_ptr
A nil UnknownUserException reference.

Remarks

This method is intended to be used by client and server applications to create a nil UnknownUserException reference.

UnknownUserException::exception

Specifies the UserException contained in a CORBA::UnknownUserException.

Original class

CORBA::UnknownUserException

IDL syntax

```
CORBA::Any &exception();
```

Return value

CORBA::Any &
An Any object whose type() indicates the type of the exception that was thrown (some subclass of CORBA::UserException) and whose value() is the exception that was thrown (an instance of some subclass of CORBA::UserException). The UnknownUserException object retains ownership of the returned Any and its contents.

Remarks

This method is intended to be used by applications that catch a CORBA::UnknownUserException when attempting to invoke a method dynamically using the DII. The exception() method can be used to access the specific UserException that was thrown by the remote request.

UserException Class

Describes an application-specific exception condition that has occurred.

File name

usr_excp.h

Intended usage

This class is intended to be caught in the catch clause of a try/catch block that encompasses operation invocations. Typically UserException instances will actually be instances of some application-specific subclass of UserException, or CORBA::UnknownUserException. For each application-specific exception defined in IDL, the C++ bindings define a corresponding subclass of CORBA::UserException, which the interface implementation can throw.

In the Java implementation, org.omg.CORBA.UserException derives from java.lang.Exception.

Supported methods

UserException::_duplicate
UserException::_nil
UserException::id (inherited from Exception)

UserException::_duplicate

Duplicates a UserException object.

Original class

CORBA::UserException

IDL syntax

```
static CORBA::UserException_ptr _duplicate
       (CORBA::UserException_ptr p);
```

Parameters

p The UserException object to be duplicated. The reference can be nil, in which case the return value will also be nil.

Return value

CORBA::UserException_ptr
The new UserException object reference. This value should subsequently be released using CORBA::release(UserException_ptr).

Remarks

This method is intended to be used by client and server applications to duplicate a reference to a UserException object. Both the original and the duplicate reference should subsequently be released using CORBA::release(UserException_ptr).

UserException::_nil

Returns a nil CORBA::UserException reference.

Original class

CORBA::UserException

IDL syntax

```
static CORBA::UserException_ptr _nil ();
```

Return value

CORBA::UserException_ptr
A nil UserException reference.

Remarks

This method is intended to be used by client and server applications to create a nil UserException reference.

WstringDef Interface

The WstringDef interface is used to represent an OMG IDL bounded string of wide characters.

Not supported by OS/390 Component Broker

File name

somir.idl

Intended usage

An instance of a WstringDef object is used by the Interface Repository to represent an OMG IDL bounded wide string data type. The WstringDef object is not a named Interface Repository object (it is in a group of interfaces known as Anonymous types), and as such does not reside as a named object in the Interface Repository database.

An instance of a WstringDef object can be created using the create_wstring operation of the Repository interface. The WstringDef is intended to represent a string of wide characters whose kind is pk_wchar.

Local-only

True

Ancestor interfaces

IDLType Interface

Exceptions

CORBA::SystemException

IDL syntax

```
module CORBA {
  interface WstringDef:IDLType {
  attribute unsigned long bound;
  };
  };
```

Supported operations

WstringDef::bound

WstringDef::bound

The bound read and write operations allow the access and update of the bound attribute of a bounded string definition (CORBA::WstringDef) within the Interface Repository.

Original interface

WstringDef Interface

IDL syntax

```
attribute unsigned longbound;
```

Parameters

bound In Read operation, no input parameters are defined.

In Write operation, CORBA::ULong bound. The bound parameter is the new value to which the bound attribute of the CORBA::WstringDef object is set.

Return value

ULong In Read operation, CORBA::ULong. The returned is the current value of the bound attribute of the string definition (CORBA::WstringDef) object.

In Write operation, no value is returned.

Exceptions

CORBA::SystemException

Remarks

The bound attribute specifies the maximum number of characters in the string, and must not be zero.

Example

```
// C++
  // assume that 'this_string' has already been initialized
  CORBA::WstringDef * this_wstring;

  // change the bound attribute of the string definition
  CORBA::ULong new_bound = 409;
  this_string-> bound (new_bound);

  // obtain the bound of a string definition
  CORBA::ULong returned_bound;
  returned_bound = this_wstring-> bound ();
```

Chapter 3. CosConcurrencyControl in the Concurrency Service

The CosConcurrencyControl module is the only module in the Concurrency Service.

Note: The Concurrency Service is not supported by OS/390 Component Broker. Refer to the *Component Broker Advanced Programming Guide* for more information on how to simulate a concurrency service in OS/390 Component Broker.

CosConcurrencyControl Module

Not supported by OS/390 Component Broker

File stem

CosConcurrency

Types

lock_mode

This type enumerates the various modes in which a lock can be requested.

```
enum lock_mode {
        read,
        write,
        upgrade,
        intention_read,
        intention_write};
```

Interfaces

CosConcurrencyControl::LockCoordinator Interface
CosConcurrencyControl::LockSet Interface
CosConcurrencyControl::LockSetFactory Interface
CosConcurrencyControl::TransactionalLockSet Interface

LockCoordinator Interface

Enables a transaction service to drop a set of related locks that are held by a transaction. One LockCoordinator object exists for each group of related lock sets for each transaction that holds a lock on (or has an outstanding request against) one or more of those lock sets.

This interface forms part of the implementation of the Concurrency Control Service Proposal specified by the Object Management group (OMG).

File stem

CosConcurrency

Intended usage

Instances of the LockCoordinator interface are created and managed internally by the Concurrency Service.

Local-only

True

IDL syntax

```
interface LockCoordinator
{
    void drop_locks();
};
```

Supported operations

LockCoordinator::drop_locks

LockCoordinator::drop_locks

Releases a group of related locks held by a transaction.

Original interface

CosConcurrencyControl::LockCoordinator Interface

IDL syntax

```
void drop_locks();
```

Remarks

The drop_locks operation enables a transaction to drop all the locks that it holds, and to cancel all the lock requests on which it is waiting in a group of related lock sets.

Example

```
/* C++ example */
#include <CosConcurrency.hh>
#include <CosTransactions.hh>
{
    CosConcurrencyControl::LockSet_ptr lockset;
    CosTransactions::Coordinator_ptr coord;
    CosConcurrencyControl::LockCoordinator_ptr lock_coord;
    ...
```

```
    // Get the LockSetFactory object (a new one is created if it does not
    // exist)
    lsfact = CosConcurrencyControl::LockSetFactory::_create();
    // Create a lockset using the LockSetFactory object
    lockset = lsfact->create();
    lock_coord = lockset->get_coordinator(coord);
    lock_coord->drop_locks();
}
```

LockSet Interface

Provides operations to acquire and release locks on behalf of the calling thread or transaction.

This interface forms part of the implementation of the Concurrency Control Service Proposal specified by the Object Management group (OMG).

File stem

CosConcurrency

Intended usage

Use of the LockSet interface *within* the scope of a transaction causes the request to be associated implicitly with the current transaction. Use of the interface *outside* of the scope of a transaction causes the request to be implicitly associated with the current thread.

To acquire or release locks on behalf of an explicitly specified transaction, use the TransactionalLockSet Interface.

Local-only

True

Exceptions

LockNotHeld
: This exception is raised when an operation to unlock or change the mode of a lock is called and the specified lock is not held.

IDL syntax

```
interface LockSet
{
    void lock(in lock_mode mode);
    boolean try_lock(in lock_mode mode);
    void unlock(in lock_mode mode)
        raises(LockNotHeld)
    void change_mode(in lock_mode held_mode
```

```
        in lock_mode new_mode)
        raises(LockNotHeld);
    Lock_Coordinator get_coordinator(
        in Transactions::Coordinator which);
};
```

Supported operations

LockSet::change_mode
LockSet::get_coordinator
LockSet::lock
LockSet::try_lock
LockSet::unlock

LockSet::change_mode

Changes the mode of a single lock.

Original interface

CosConcurrencyControl::LockSet Interface

IDL syntax

```
void change_mode(in lock_mode held_mode,
    in lock_mode new_mode)raises (LockNotHeld);
```

Parameters

held_mode
The current mode of the lock.

new_mode
The mode to which the lock is to be changed.

Exceptions

LockNotHeld
This exception is raised when an operation to unlock or change the mode of a lock is called and the specified lock is not held.

Remarks

This operation changes the mode of a lock on behalf of the current transaction (if the call is made within the scope of a transaction), or on behalf of the current thread (if the call is made outside the scope of a transaction). If the current transaction or thread does not already hold the lock in the held_mode specified as an input parameter, then the LockNotHeld exception is raised.

If granting the lock in the new mode would cause conflict with one or more locks already held on this LockSet, (see Conflicting Locks in a Lock Set), the lock is not acquired immediately. It is placed at the end of a queue of waiting lock requests for this LockSet and the thread calling this operation is blocked. The queue of waiting lock requests is processed (primarily) on a first-in, first-out basis as described in detail in Servicing Lock Requests in a Lock Set.

Example

```
/* C++ example */
#include <CosConcurrency.hh>
{
    CosConcurrencyControl::LockSet_ptr lockset;
    ...
    // Get the LockSetFactory object (a new one is created if it does not
    // exist)
    lsfact = CosConcurrencyControl::LockSetFactory::_create();
    // Create a lockset using the LockSetFactory object
    lockset = lsfact->create();
    ...
    try
    {
         lockset->lock(CosConcurrencyControl::upgrade);
    }
    catch(CosConcurrencyControl::TransactionRolledBack)
    {
           //Handle Exception
    }
    try
    {
         lockset->change_mode(CosConcurrencyControl::upgrade,
                CosConcurrencyControl::write);
    }
    catch (CosConcurrencyControl::LockNotHeld)
    {
         //Handle Exception
    }
    //...Update data...
    try
    {
         lockset->unlock(CosConcurrencyControl::write);
    }
    catch (CosConcurrencyControl::LockNotHeld &exc)
    {
        //Handle exception
    }
}
```

LockSet::get_coordinator

Returns the LockCoordinator associated with a specified transaction and the group of LockSets that are related to the target object.

Original interface

CosConcurrencyControl::LockSet Interface

IDL syntax

```
LockCoordinator get_coordinator (in CosTransactions::Coordinator coord);
```

Parameters

coord A pointer to the transaction for which the lock coordinator is required.

Return value

LockCoordinator
The LockCoordinator associated with the transaction.

Remarks

For every group of related lock sets that a transaction holds one of more locks on, or has one or more outstanding lock requests against, a LockCoordinator object exists for that transaction. This LockCoordinator object enables the transaction to drop all the held locks and to discard the waiting lock requests that it has in the related lock set group (by calling the LockCoordinator::drop_locks operation).

This operation is used to get the LockCoordinator object prior to dropping a group of related locks for a transaction.

Example

```
/* C++ example */
#include <CosConcurrency.hh>
#include <CosTransactions.hh>
{
    CosConcurrencyControl::LockSet_ptr lockset;
    CosTransactions::Coordinator_ptr coord;
    CosConcurrencyControl::LockCoordinator_ptr lock_coord;
    ...
    // Get the LockSetFactory object (a new one is created if it does not
    // exist)
    lsfact = CosConcurrencyControl::LockSetFactory::_create();
    // Create a lockset using the LockSetFactory object
    lockset = lsfact->create();
```

```
    ...
    lock_coord = lockset->get_coordinator(coord);
    lock_coord->drop_locks;
}
```

LockSet::lock

Acquires a lock on a specified LockSet in a specified mode.

Original interface

CosConcurrencyControl::LockSet Interface

IDL syntax

```
void lock(in lock_mode mode);
     raises (TransactionRolledBack)
```

Parameters

mode The specified mode of the lock.

Exceptions

TransactionRolledBack

Remarks

This operation acquires a lock on behalf of the current transaction (if the call is made within the scope of a transaction), or on behalf of the current thread (if the call is made outside the scope of a transaction). The lock is acquired on the LockSet in the mode passed as an input parameter. The mode indicates the level of concurrency that is allowed to other transactions and threads attempting to acquire locks on this LockSet while this lock is held.

If one or more locks that conflict with this lock, (see Conflicting Locks in a Lock Set), are already held on this LockSet, this lock is not acquired immediately. It is placed at the end of a queue of waiting lock requests for this LockSet, and the thread calling this operation is blocked. The queue of waiting lock requests is processed (primarily) on a first-in, first-out basis as described in detail in Servicing Lock Requests in a Lock Set.

If this call is made on behalf of a transaction and is blocked, and the transaction is rolled back, the call returns the TransactionRolledBack exception.

Example

```
/* C++ example */
#include <CosConcurrency.hh>
#include <CosTransactions.hh>
{
    CosConcurrencyControl::LockSetFactory_ptr lsfact;
```

```
    CosConcurrencyControl::LockSet_ptr lockset;
    ...
    // Get the LockSetFactory object (a new one is created if it does not
    // exist)
   lsfact = CosConcurrencyControl::LockSetFactory::_create();
    // Create a lockset using the LockSetFactory object
   lockset = lsfact->create();
    ...
    try
    {
       lockset->lock(CosConcurrencyControl::write);
    }
    catch(CosConcurrencyControl::TransactionRolledBack)
    {
       //Handle Exception
    }
    try
    {
       lockset->unlock(CosConcurrencyControl::write);
    }
    catch (CosConcurrencyControl::LockNotHeld &exc)
    {
        //Handle exception
    }
}
```

LockSet::try_lock

Attempts to acquire a lock on a specified LockSet.

Original interface

CosConcurrencyControl::LockSet Interface

IDL syntax

```
boolean try_lock(in lock_mode mode);
```

Parameters

mode The requested mode of the lock.

Return value

TRUE The lock is acquired

FALSE The lock is not acquired.

Remarks

This operation attempts to acquire a lock on behalf of the current transaction (if the call is made within the scope of a transaction), or on behalf of the current thread (if the call

is made outside the scope of a transaction). The lock is acquired on the LockSet in the *mode* passed as an input parameter. The mode indicates the level of concurrency that is allowed to other transactions and threads attempting to acquire locks on this LockSet while this lock is held.

If one or more *incompatible* locks are already held on this LockSet, this operation returns FALSE, and the attempt to acquire the lock is abandoned. (Unlike the lock operation, this operation does not block the calling thread if the lock cannot be immediately acquired.)

Example

```
/* C++ example */
#include <CosConcurrency.hh>
{
    CosConcurrencyControl::LockSetFactory_ptr lsfact;
    CosConcurrencyControl::LockSet_ptr lockset;
    ...
    // Get the LockSetFactory object (a new one is created if it does not
    // exist)
    lsfact = CosConcurrencyControl::LockSetFactory::_create();
    // Create a lockset using the LockSetFactory object
    lockset = lsfact->create();
    if (lockset->try_lock(CosConcurrencyControl::write))
    {
        // ...Update data
    }
    else
    {
        //...Act on failure to obtain lock ...
    }
}
```

LockSet::unlock

Drops a single lock on the specified LockSet in the specified mode.

Original interface

CosConcurrencyControl::LockSet Interface

IDL syntax

```
void unlock(in lock_mode mode)
    raises(LockNotHeld);
```

Parameters

mode The specified mode of the lock to be dropped.

Exceptions

LockNotHeld

This exception is raised when an operation to unlock or change the mode of a lock is called and the specified lock is not held.

Remarks

This operation drops (releases) a lock on behalf of the current transaction (if the call is made within the scope of a transaction), or on behalf of the current thread (if the call is made outside the scope of a transaction). If the lock is not held on the LockSet in the mode that is passed as an input parameter, the LockNotHeld exception is raised.

Example

```
/* C++ example */
#include <CosConcurrency.hh>
{
    CosConcurrencyControl::LockSetFactory_ptr lsfact;
    CosConcurrencyControl::LockSet_ptr lockset;
    ...
    // Get the LockSetFactory object (a new one is created if it does not
    // exist)
    lsfact = CosConcurrencyControl::LockSetFactory::_create();
    // Create a lockset using the LockSetFactory object
    lockset = lsfact->create();
    try
    {
        lockset->lock(CosConcurrencyControl::write);
    }
    catch(CosConcurrencyControl::TransactionRolledBack)
    {
        //Handle Exception
    }
    // ...Update data ...
    try
    {
        lockset->unlock(CosConcurrencyControl::write);
    }
    catch (CosConcurrencyControl::LockNotHeld)
    {
        //Handle Exception
    }
}
```

LockSetFactory Interface

Creates and initializes LockSet and TransactionalLockSet objects.

This interface forms part of the implementation of the Concurrency Service Proposal specified by the Object Management group (OMG).

File stem

CosConcurrency

Local-only

True

IDL syntax

```
interface LockSetFactory
{
    LockSet create();
    LockSet create_related(in LockSet which);
    TransactionalLockSet create_transactional();
    TransactionalLockSet create_transactional(
        in TransactionalLockSet which);
};
```

Supported operations

LockSetFactory::create
LockSetFactory::create_related
LockSetFactory::create_transactional
LockSetFactory::create_transactional_related

LockSetFactory::create

Creates and initializes a new LockSet.

Original interface

CosConcurrencyControl::LockSetFactory Interface

IDL syntax

```
LockSet create();
```

Return value

LockSet
A new LockSet, not related to any other LockSets.

Remarks

Creates and initializes a new LockSet that is not related to any other LockSets.

Example

```
/* C++ example */
#include <CosConcurrency.hh>
{
    CosConcurrencyControl::LockSetFactory_ptr lsfact;
    CosConcurrencyControl::LockSet_ptr lockset;
    // Get the LockSetFactory object (a new one is created if it does not
    // exist)
    lsfact = CosConcurrencyControl::LockSetFactory::_create();
    // Create a lockset using the LockSetFactory object
    lockset = lsfact->create();
    // Perform locking and unlocking on LockSet object
    release lockset;
    release lsfact;
}
```

LockSetFactory::create_related

Creates and initializes a new LockSet that is related to an existing LockSet.

Original interface

CosConcurrencyControl::LockSetFactory Interface

IDL syntax

```
LockSet create_related(in LockSet lockset);
```

Parameters

lockset
A pointer to the lock set to which the new one is to be related.

Return value

LockSet
A new LockSet, related to an existing LockSet.

Remarks

The LockSet that is created is related to the LockSet that is passed as an input parameter. Relating LockSets together enables a transaction to drop all the locks that it holds on that group of related LockSets with one call to the LockCoordinator interface.

Example

```
/* C++ example */
#include <CosConcurrency.hh>
{
    CosConcurrencyControl::LockSetFactory_ptr lsfact;
    CosConcurrencyControl::LockSet_ptr lockset1, lockset2, lockset3;
    // Get the LockSetFactory object (a new one is created if it does not
```

```
    // exist)
    lsfact = CosConcurrencyControl::LockSetFactory::_create();
    //Create a lockset using the LockSetFactory object
    lockset1 = lsfact->create();
    // Create a lockset that is related to the initial one
    lockset2 = lsfact->create_related(lockset1);
    //Create a lockset that is related to the initial one. Note that the
    //following relates lockset3 with lockset2 as well.
    lockset3 = lsfact->create_related(lockset1);
    ...
    // Perform locking and unlocking on LockSet objects
    ...
    release lockset1;
    release lockset2;
    release lockset3;
    release lsfact;
}
```

LockSetFactory::create_transactional

Creates and initializes a new TransactionalLockSet.

Original interface

CosConcurrencyControl::LockSetFactory Interface

IDL syntax

```
TransactionalLockSet create_transactional();
```

Return value

TransactionalLockSet
A new TransactionalLockSet that is not related to any other TransactionalLockSet.

Remarks

Creates and initializes a new TransactionalLockSet that is not related to any other TransactionalLockSet.

Example

```
/* C++ example */
#include <CosConcurrency.hh>
#include <CosTransactions.hh>
{
    CosConcurrencyControl::LockSetFactory *lsfact;
    CosConcurrencyControl::TransactionalLockSet *tlockset;
    // Get the LockSetFactory object (a new one is created if it does not
    // exist)
```

```
    lsfact = CosConcurrencyControl::LockSetFactory::_create();
    // Create a transactional lockset using the LockSetFactory object
    tlockset = lsfact->create_transactional();
    //Perform locking and unlocking on TransactionalLockSet object //
    ...
    release tlockset;
    release lsfact;
}
```

LockSetFactory::create_transactional_related

Creates a new TransactionalLockSet that is related to an existing TransactionalLockSet.

Original interface

CosConcurrencyControl::LockSetFactory Interface

IDL syntax

```
TransactionalLockSet create_transactional_related(in
    TransactionalLockSet tlockset);
```

Parameters

tlockset
A pointer to the lock set to which the new one is to be related.

Return value

TransactionalLockSet
A new TransactionalLockSet, related to an existing TransactionalLockSet.

Remarks

The TransactionalLockSet that is created is related to the TransactionalLockSet that is passed as an input parameter. Relating TransactionalLockSets together enables a transaction to drop all the locks that it holds on that group of related TransactionalLockSets with one call to the LockCoordinator interface.

Example

```
/* C++ example */
#include <CosConcurrency.hh>
#include <CosTransactions.hh>
{
    CosConcurrencyControl::LockSetFactory_ptr lsfact;
    CosConcurrencyControl::TransactionalLockSet_ptr tlockset1, tlockset2,
                                                    tlockset3;
    // Get the LockSetFactory object (a new one is created if it does not
    // exist)
    lsfact = CosConcurrencyControl::LockSetFactory::_create();
```

```
    // Create a transactional lockset using the LockSetFactory object
    tlockset1 = lsfact->create_transactional();
    // Create a transactional lockset that is related to the initial one
    tlockset2 = lsfact->create_transactional_related(tlockset1);
    // Create a transactional lockset that is related to the initial one.
    //Note that the following relates tlockset3 with tlockset2 as well.
    tlockset3 = lsfact->create_transactional_related(tlockset1);
    ...
    // Perform locking and unlocking on TransactionalLockSet objects
    ...
    release tlockset1;
    release tlockset2;
    release tlockset3;
    release lsfact;
}
```

TransactionalLockSet Interface

Provides operations to acquire and release locks on behalf of a specific transaction.

This interface forms part of the implementation of the Concurrency Control Service Proposal specified by the Object Management group (OMG).

File stem

CosConcurrency

Intended usage

The TransactionalLockSet interface provides operations to acquire and release locks on behalf of a specific transaction. They operate in the same way as the equivalent LockSet operations (see “LockSet Interface” on page 289), except that in the TransactionalLockSet interface, the identity of the transaction is passed explicitly as a reference to the coordinator for the transaction, (instead of being acquired implicitly through an association with the calling thread or transaction).

Local-only

True

Exceptions

LockNotHeld

This exception is raised when an operation to unlock or change the mode of a lock is called and the specified lock is not held.

IDL syntax

```
interface TransactionalLockSet
{
    void lock(in Transactions::Coordinator current,
        in lock_mode mode);
    boolean try_lock(in Transactions::Coordinator current,
        in lock_mode mode);
    void unlock(in Transactions::Coordinator current,
        in lock_mode mode)
        raises(LockNotHeld);
    void change_mode(in Transactions::Coordinator current,
        in lock_mode held_mode, in lock_mode new_mode)
        raises(LockNotHeld);
    Lock_Coordinator get_coordinator(
        in Transactions::Coordinator which);
};
```

Supported operations

TransactionalLockSet::change_mode
TransactionalLockSet::get_coordinator
TransactionalLockSet::lock
TransactionalLockSet::try_lock
TransactionalLockSet::unlock

TransactionalLockSet::change_mode

Changes the mode of a single lock.

Original interface

CosConcurrencyControl::TransactionalLockSet Interface

IDL syntax

```
void change_mode(in CosTransactions::Coordinator coord,
   in lock_mode held_mode, in lock_mode new_mode)
   raises (LockNotHeld);
```

Parameters

coord A pointer to the transaction on which the lock is to be changed.

held_mode
The current mode of the lock.

new_mode
The mode to which the lock is to be changed.

Exceptions

LockNotHeld

This exception is raised when an operation to unlock or change the mode of a lock is called and the specified lock is not held.

Remarks

This operation changes the mode of a lock on behalf of the transaction passed as an input parameter. If the current transaction does not already hold the lock in the held-mode specified as an input parameter, then the LockNotHeld exception is raised.

If granting the lock in the new mode would cause conflict with one or more locks already held on this TransactionalLockSet, (see Conflicting Locks in a Lock Set), the lock is not acquired immediately. It is placed at the end of a queue of waiting lock requests for this TransactionalLockSet and the thread calling this operation is blocked. The queue of waiting lock requests is processed (primarily) on a first-in, first-out basis as described in detail in Servicing Lock Requests in a Lock Set.

Example

```
/* C++ example */
#include <CosConcurrency.hh>
#include <CosTransactions.hh>
{
    CosConcurrencyControl::TransactionalLockSet_ptr tlockset;
    CosTransactions::Coordinator_ptr coord;
    ...
    // Get the LockSetFactory object (a new one is created if it does not
    // exist)
    lsfact = CosConcurrencyControl::LockSetFactory::_create();
    // Create a transactional lockset using the LockSetFactory object
    tlockset = lsfact->create_transactional();
    try
    {
        tlockset->lock(coord,CosConcurrencyControl::upgrade);
    }
    catch(CosConcurrencyControl::TransactionRolledBack)
    {
        //Handle Exception
    }
    // ...Read data ...
    try
    {
        tlockset->change_mode(coord, CosConcurrencyControl::upgrade,
                CosConcurrencyControl::write);
    }
    catch (CosConcurrencyControl::LockNotHeld)
    {
        //Handle Exception
    }
    // ...Update data ...
    try
    {
```

```
        tlockset->unlock(coord, CosConcurrencyControl::write);
    }
    catch (CosConcurrencyControl::LockNotHeld)
    {
        //Handle Exception
    }
}
```

TransactionalLockSet::get_coordinator

Returns the LockCoordinator associated with a specified transaction and the group of related TransactionalLockSets that are related to the target object.

Original interface

CosConcurrencyControl::TransactionalLockSet Interface

IDL syntax

```
LockCoordinator get_coordinator (in CosTransactions::Coordinator coord);
```

Parameters

coord A pointer to the transaction with which the lock coordinator is associated.

Return value

LockCoordinator
The LockCoordinator associated with the specified transaction for this TransactionalLockSet's related group. The caller should not free the returned object; the Concurrency Service retains ownership of it.

Remarks

For every group of related lock sets that a transaction holds one of more locks on, or has one or more outstanding lock requests against, a LockCoordinator object exists for that transaction. This LockCoordinator object enables the transaction to drop all the held locks and to discard the waiting lock requests that it has in the related lockset group (by calling the LockCoordinator::drop_locks operation).

This operation is used to get the LockCoordinator object prior to dropping a group of related locks for a transaction.

Example

```
/* C++ example */
#include <CosConcurrency.hh>
#include <CosTransactions.hh>
{
    CosConcurrencyControl::TransactionalLockSet_ptr tlockset;
```

```
    CosTransactions::Coordinator_ptr coord;
    CosConcurrencyControl::LockCoordinator_ptr lock_coord;
    ...
    // Get the LockSetFactory object (a new one is created if it does not
    // exist)
    lsfact = CosConcurrencyControl::LockSetFactory::_create();
    // Create a transactional lockset using the LockSetFactory object
    tlockset = lsfact->create_transactional();
    // Get the lock coordinator object representing this transaction and
    // the group of related lock sets to which the tlockset belongs.
    lock_coord = tlockset->get_coordinator(coord);
    // Drop all the transaction's locks in the group of related
    // lock sets.
    lock_coord->drop_locks();
}
```

TransactionalLockSet::lock

Acquires a lock on a specified LockSet in a specified mode.

Original interface

CosConcurrencyControl::TransactionalLockSet Interface

IDL syntax

```
void lock(in CosTransactions::Coordinator coord,
    in lock_mode mode);
```

Parameters

coord A pointer to the transaction on which the lock is to be acquired.

mode The specified mode of the lock.

Remarks

This operation acquires a lock on the target TransactionalLockSet object. This lock is acquired on behalf of the transaction (passed as an input parameter), in the mode passed as an input parameter. The mode indicates the level of concurrency that is allowed to other transactions attempting to acquire locks on this TransactionalLockSet while this lock is held.

If one or more locks that conflict with this lock (see Conflicting Locks in a Lock Set) are already held on this TransactionalLockSet, this lock is not acquired immediately. It is placed at the end of a queue of waiting lock requests for this TransactionalLockSet, and the thread calling this operation is blocked. The queue of waiting lock requests is processed (primarily) on a first-in, first-out basis as described in detail in Servicing Lock Requests in a Lock Set.

If one or more incompatible locks are already held on this TransactionalLockSet, this lock is not acquired immediately. It is placed at the end of a queue of waiting lock requests for this TransactionalLockSet, and the thread calling this operation is blocked. When this request reaches the start of the queue, and when no incompatible locks are held on the TransactionalLockSet, the lock is automatically acquired, and the thread is resumed.

Example

```
/* C++ example */
#include <CosConcurrency.hh>
#include <CosTransactions.hh>
{
    CosConcurrencyControl::TransactionalLockSet_ptr tlockset;
    CosTransactions::Coordinator_ptr coord;
    ...
    // Get the LockSetFactory object (a new one is created if it does not
    // exist)
    lsfact = CosConcurrencyControl::LockSetFactory::_create();
    // Create a transactional lockset using the LockSetFactory object
    tlockset = lsfact->create_transactional();
    ...
    tlockset->lock(coord,CosConcurrencyControl::write);
    // ...Update data ...
    try
    {
        tlockset->unlock(coord, CosConcurrencyControl::write);
    }
    catch (CosConcurrencyControl::LockNotHeld)
    {
        //Handle Exception
    }
}
```

TransactionalLockSet::try_lock

Attempts to acquire a lock on the specified TransactionalLockSet.

Original interface

CosConcurrencyControl::TransactionalLockSet Interface

IDL syntax

```
boolean try_lock(in CosTransactions::Coordinator coord,
    in lock_mode mode);
```

Parameters

coord A pointer to the transaction on which the lock is to be acquired.

mode The specified mode of the lock.

Return value

TRUE The lock is acquired.

FALSE The lock is not acquired.

Remarks

This operation attempts to acquire a lock on the target TransactionalLockSet. The lock is acquired on behalf of the transaction (passed as an input parameter) in the mode passed as an input parameter. The mode indicates the level of concurrency that is allowed to other transactions and threads attempting to acquire locks on this TransactionalLockSet while this lock is held.

If one or more incompatible locks are already held on this TransactionalLockSet, this operation returns FALSE, and the attempt to acquire the lock is abandoned. (Unlike the lock operation, this operation does not block the calling thread if the lock cannot be immediately acquired.)

Example

```
/* C++ example */
#include <CosConcurrency.hh>
#include <CosTransactions.hh>
{
    CosConcurrencyControl::TransactionalLockSet_ptr tlockset;
    CosTransactions::Coordinator_ptr coord;
    ...
    // Get the LockSetFactory object (a new one is created if it does not
    // exist)
    lsfact = CosConcurrencyControl::LockSetFactory::_create();
    // Create a transactional lockset using the LockSetFactory object
    tlockset = lsfact->create_transactional();
    if (tlockset->try_lock(coord,CosConcurrencyControl::write))
    {
        // ...Update data ...
    }
    else
    {
        // ...Act on failure to obtain lock ...
    }
}
```

TransactionalLockSet::unlock

Drops a single lock on the specified TransactionalLockSet in the specified mode.

Original interface

CosConcurrencyControl::TransactionalLockSet Interface

IDL syntax

```
void unlock(in CosTransactions::Coordinator coord,
    in lock_mode mode)
    raises(LockNotHeld);
```

Parameters

coord A pointer to the transaction from which the lock is to be dropped.

mode The specified mode of the lock to be dropped.

Exceptions

LockNotHeld
This exception is raised when an operation to unlock or change the mode of a lock is called and the specified lock is not held.

Remarks

This operation drops (releases) a lock on behalf of the transaction passed as an input parameter. If the lock is not held on the TransactionalLockSet in the mode that is passed as an input parameter, the LockNotHeld exception is raised.

Example

```
/* C++ example */
#include <CosConcurrency.hh>
#include <CosTransactions.hh>
{
    CosConcurrencyControl::TransactionalLockSet_ptr tlockset;
    CosTransactions::Coordinator_ptr coord;
    ...
    // Get the LockSetFactory object (a new one is created if it does not
    // exist)
    lsfact = CosConcurrencyControl::LockSetFactory::_create();
    // Create a transactional lockset using the LockSetFactory object
    tlockset = lsfact->create_transactional();
    tlockset->lock(coord,CosConcurrencyControl::write);
    // ...Update data ...
    try
    {
        tlockset->unlock(coord, CosConcurrencyControl::write);
    }
    catch (CosConcurrencyControl::LockNotHeld)
    {
        //Handle Exception
    }
}
```

Chapter 4. CosEventChannelAdmin in the Event Service

The other modules in the Event Service are:
- CosEventComm
- IExtendedEventChannelAdmin

Note: The Event Service is not supported by OS/390 Component Broker.

CosEventChannelAdmin Module

Allows users to implement the Event Channel object to be used in the event channel programming model. The Event Channel object is implemented by the Event Service in the Component Broker. It is not intended to be implemented or modified by any customers.

Not supported by OS/390 Component Broker

File name

CosEventChannelAdmin.idl

Intended usage

This module allows users to implement the Event Channel object to be used in the event channel programming module. The Event Channel object is implemented by the Event Service in the Component Broker. It is not intended to be implemented or modified by any customers.

This module defines the interface that the Event Service uses to connect suppliers and consumers through the event channel. Suppliers and consumers interact by connecting to the same EventChannel. Events created by suppliers of an EventChannel are passed to its consumers through the event channel.

A consumer or supplier connects to an EventChannel through a proxy object. For example, a PushConsumer connects to an EventChannel through a ProxyPushSupplier. The correlation between the object and the proxy object that connects it to the EventChannel is as follows:
- PushSupplier connects to an EventChannel through a ProxyPushConsumer.
- PushConsumer connects to an EventChannel through a ProxyPushSupplier.
- PullSupplier connects to an EventChannel through a ProxyPullConsumer.
- PullConsumer connects to an EventChannel through a ProxyPullSupplier.

The following class objects are associated with every EventChannel:

SupplierAdmin
The factory object for proxy consumer objects; that is, objects that connect suppliers with the event channel.

ConsumerAdmin
The factory object for proxy supplier objects; that is, objects that connect consumers with the event channel.

The for_consumers and for_suppliers methods of an EventChannel obtain references to the related ConsumerAdmin and SupplierAdmin objects, respectively.

All interfaces defined here are abstract. The Event Service will implement these interfaces.

Exceptions

AlreadyConnected
This exception is thrown if the connection has already been established between the proxies and the suppliers/consumers. You can ignore this exception and use the existing connection.

Interfaces

CosEventChannelAdmin::EventChannel Interface
CosEventChannelAdmin::SupplierAdmin Interface
CosEventChannelAdmin::ConsumerAdmin Interface
CosEventChannelAdmin::ProxyPushSupplier Interface
CosEventChannelAdmin::ProxyPushConsumer Interface
CosEventChannelAdmin::ProxyPullSupplier Interface
CosEventChannelAdmin::ProxyPullConsumer Interface

ConsumerAdmin Interface

Defines the first step for connecting consumers to the event channel; clients use it to obtain proxy suppliers.

File stem

CosEventChannelAdmin.idl

Intended usage

ConsumerAdmin is an abstract base class. Instances of this class make connections between consumers and event channels. Operations of this class create instances of proxy supplier objects connected to the associated event channel.

IDL syntax

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

Supported operations

ConsumerAdmin::obtain_push_supplier
ConsumerAdmin::obtain_pull_supplier

ConsumerAdmin::obtain_push_supplier

Creates instances of proxy supplier objects connected to the associated event channel.

Original class

CosEventChannelAdmin::ConsumerAdmin Interface

IDL syntax

```
ProxyPushSupplier obtain_push_supplier();
```

Return value

ProxyPushSupplier

Returns a new ProxyPushSupplier, which a push consumer can use to connect to the event channel.

Exceptions

None

Remarks

This method is designed to be subclassed and overridden. This method is called by clients. Since there is no implementation in this method, the overridden method does not need to pass the call to this parent.

Example

```
CosEventChannelAdmin::EventChannel_ptr ec = NULL;
CosEventChannelAdmin::ConsumerAdmin_ptr ca;
CosEventChannelAdmin::ProxyPushSupplier_ptr pxyPushSupplier;
...       //get the EventChannel

          // Get consumerAdmin
```

```
ca = ec->for_consumers();
          // Obtain ProxyPushSupplier
pxyPushSupplier = ca->obtain_push_supplier();
...
```

ConsumerAdmin::obtain_pull_supplier

Returns a ProxyPullSupplier object. The ProxyPullSupplier object is then used to connect a PullConsumer.

Original class

CosEventChannelAdmin::ConsumerAdmin Interface

IDL syntax

```
ProxyPullSupplier obtain_pull_supplier();
```

Return value

ProxyPullSupplier
Returns a new ProxyPullSupplier, which a pull consumer can use to connect to the event channel.

Exceptions

None

Remarks

This method is designed to be subclassed and overridden. This method is called by clients. Since there is no implementation in this method, the overridden method does not need to pass the call to this parent.

Example

```
CosEventChannelAdmin::EventChannel_ptr ec = NULL;
CosEventChannelAdmin::ConsumerAdmin_ptr ca;
CosEventChannelAdmin::ProxyPullSupplier_ptr pxyPullSupplier;
...       //get the EventChannel

          // Get consumerAdmin
ca = ec->for_consumers();
          // Obtain ProxyPullSupplier
pxyPullSupplier = ca->obtain_pull_supplier();
...
```

EventChannel Interface

Defines three administrative operations: adding consumers, adding suppliers, and destroying the channel.

File stem

CosEventChannelAdmin.idl

Intended usage

An abstract base class from which actual event channel implementation inherits.

IDL syntax

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

Supported operations

EventChannel::for_suppliers
EventChannel::for_consumers
EventChannel::destroy

EventChannel::for_consumers

Returns a ConsumerAdmin object used in the procedure for obtaining a proxy supplier to establish a connection between a consumer and an EventChannel.

Original class

CosEventChannelAdmin::EventChannel Interface

IDL syntax

```
ConsumerAdmin for_consumers();
```

Return value

ConsumerAdmin

Returns a ConsumerAdmin object, that can later obtain proxy suppliers, through which consumers can connect to the EventChannel. Do not free this object. It is automatically freed when the event channel is destroyed by use of the destroy method.

Exceptions

None

Remarks

This method is designed to be subclassed and overridden. This method is called by clients. Since there is no implementation in this method, the overridden method does not need to pass the call to this parent.

Example

```
CosEventChannelAdmin::EventChannel_ptr ec = NULL;
CosEventChannelAdmin::ConsumerAdmin_ptr ca;
   ...    // Initialize server and get object reference
          // Get Consumer Admin through Event Channel.
ca = ec->for_consumers();
...
```

EventChannel::for_suppliers

Returns a SupplierAdmin object used in the procedure for obtaining a proxy consumer to establish a connection between a supplier and an EventChannel.

Original class

CosEventChannelAdmin::EventChannel Interface

IDL syntax

```
SupplierAdmin for_suppliers();
```

Return value

SupplierAdmin

Returns a SupplierAdmin object, that later can obtain proxy consumers, through which suppliers can connect to the EventChannel. Do not free this object. This object is automatically freed when the EventChannel is destroyed using the destroy method.

Exceptions

None

Remarks

This method is designed to be subclassed and overridden. This method is called by clients. Since there is no implementation in this method, the overridden method does not need to pass the call to this parent.

Example

```
CosEventChannelAdmin::EventChannel_ptr ec = NULL;
CosEventChannelAdmin::SupplierAdmin_ptr sa;
   ...    // Initialize server and get object reference
          // Get Supplier Admin through Event Channel
sa = ec->for_suppliers();
...
```

EventChannel::destroy

Destroys the specified event channel and the associated ConsumerAdmin, SupplierAdmin, proxy supplier, and proxy consumer objects.

Original class

CosEventChannelAdmin::EventChannel Interface

IDL syntax

```
void destroy();
```

Exceptions

None

Remarks

This method is designed to be subclassed and overridden. This method is called by clients. Because there is no implementation in this method, the overridden method does not need to pass the call to this parent.

The event channels created by any client can be destroyed. However, never destroy the default event channel provided by the server. If destroyed, unpredicatable results can occur.

Example

```
CosEventChannelAdmin::EventChannel_ptr ec = NULL;
...       //get the EventChannel

          // destroy the EventChannel
ec->destroy();
```

ProxyPullConsumer Interface

Defines the second step for connecting pull suppliers to the event channel.

File stem

CosEventChannelAdmin.idl

Intended usage

ProxyPullConsumer is an abstract base class. Instances of this class serve as front-ends through which a PullSupplier supplies events to an EventChannel.

When a consumer pulls an event from the EventChannel (through an object of the ProxyPullSupplier class), the EventChannel attempts to return a new event.

This release of Component Broker does not support the TypeError exception.

Ancestor interfaces

CosEventComm::PullConsumer Interface

IDL syntax

```
interface ProxyPullConsumer: CosEventComm::PullConsumer
{
  void connect_pull_supplier
    (in CosEventComm::PullSupplier pull_supplier)
                  raises(AlreadyConnected,TypeError);
};
```

Supported operations

ProxyPullConsumer::connect_pull_supplier

ProxyPullConsumer::connect_pull_supplier

Connects a given PullSupplier to this ProxyPullConsumer. This method is part of the procedure for establishing a connection between a PullSupplier and an EventChannel.

Original class

CosEventChannelAdmin::ProxyPullConsumer Interface

IDL syntax

```
void connect_pull_supplier (in CosEventComm::PullSupplier pull_supplier)
  raises (AlreadyConnected);
```

Parameters

pull_supplier
The PullSupplier connecting to this ProxyPullConsumer.

Exceptions

AlreadyConnected

This exception is thrown if the connection has already been established between the ProxyPullConsumer and a PullSupplier. You can ignore this exception and use the existing connection.

Remarks

This method is designed to be subclassed and overridden. This method is called by clients. Since there is no implementation in this method, the overridden method does not need to pass the call to this parent.

Example

```
CosEventChannelAdmin::EventChannel_ptr ec = NULL;
CosEventComm_PullSupplier_Impl *pullSupplier;
CosEventChannelAdmin::SupplierAdmin_ptr sa = NULL;
CosEventChannelAdmin::ProxyPullConsumer_ptr pxyPullConumer;
   ...    // Initialize server and get object reference
          // Create pullSupplier
pullSupplier = new CosEventComm_pullSupplier_Impl;
          // Get Supplier Admin through Event Channel
sa = ec->for_suppliers();
          // Get ProxyPullConsumer through Supplier Admin
pxypullConsumer = sa->obtain_pull_consumer();
      // Connect the pullSupplier to the ProxyPullConsumer
pxypullConsumer->connect_pull_supplier(pullSupplier);
...
```

ProxyPullSupplier Interface

Defines the second step for connecting pull consumers to the event channel.

File stem

CosEventChannelAdmin.idl

Intended usage

ProxyPullSupplier is an abstract base class. Instances of this class serve as front-ends through which a PullConsumer gets events from an EventChannel.

When a consumer pulls an event from the EventChannel (through objects of the ProxyPullSupplier class), the EventChannel tries to return a new event. If such an event is not available, the event channel implementation may pull events from its pull suppliers.

Ancestor interfaces

CosEventComm::PullSupplier Interface

IDL syntax

```
interface ProxyPullSupplier: CosEventComm::PullSupplier
{
   void connect_pull_consumer
   (in CosEventComm::PullConsumer pull_consumer)
                  raises(AlreadyConnected);
};
```

Supported operations

ProxyPullSupplier::connect_pull_consumer

ProxyPullSupplier::connect_pull_consumer

Connects a given PullSupplier to this ProxyPullConsumer. This method is part of the procedure for establishing a connection between a PullSupplier and an EventChannel.

Original class

CosEventChannelAdmin::ProxyPullSupplier Interface

IDL syntax

```
void connect_pull_consumer
  (in CosEventComm::PullConsumer pull_consumer)
      raises (AlreadyConnected);
```

Parameters

pull_consumer
The PullConsumer connecting to the event channel through this ProxyPullSupplier. If the pull_consumer is NULL, the ProxyPullSupplier can disconnect without running disconnect_pull_consumer on the pull_consumer.

Exceptions

AlreadyConnected
This exception is thrown if the connection has already been established between the ProxyPullSupplier and a PullConsumer. You can ignore this exception and use the existing connection.

Remarks

This method is designed to be subclassed and overridden. This method is called by clients. Since there is no implementation in this method, the overridden method does not need to pass the call to this parent.

Example

```
CosEventChannelAdmin::EventChannel_ptr ec = NULL;
CosEventComm_PullSupplier_Impl *pullSupplier;
CosEventChannelAdmin::SupplierAdmin_ptr sa = NULL;
CosEventChannelAdmin::ProxyPullConsumer_ptr pxyPullConumer;
 ...    // Initialize server and get object reference
        // Create pullSupplier
pullSupplier = new CosEventComm_pullSupplier_Impl;
        // Get Supplier Admin through Event Channel
sa = ec->for_suppliers();
        // Get ProxyPullConsumer through Supplier Admin
pxypullConsumer = sa->obtain_pull_consumer();
        // Connect the pullSupplier to the ProxyPullConsumer
pxypullConsumer->connect_pull_supplier(pullSupplier);
...
```

ProxyPushConsumer Interface

Defines the second step for connecting push suppliers to the event channel.

File stem

CosEventChannelAdmin.idl

Intended usage

ProxyPushConsumer is an abstract base class. Instances of this class serve as front-ends through which a PushSupplier supplies events to an EventChannel.

When a new event is pushed on the EventChannel (through objects of the ProxyPushConsumer class), the EventChannel can push the event on its push consumers (through objects of the ProxyPushSupplier class). Further, the EventChannel can buffer these events for later use by a PullConsumer.

Ancestor interfaces

CosEventComm::PushConsumer Interface

IDL syntax

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

Supported operations

ProxyPushConsumer::connect_push_supplier

ProxyPushConsumer::connect_push_supplier

Connects a given PushSupplier to the ProxyPushConsumer. This method is part of the procedure for establishing a connection between a PushSupplier and an EventChannel.

The PushSupplier can be NULL. If the PushSupplier is NULL, the disconnect_push_supplier of the PushSupplier will not be invoked when the ProxyPushConsumer is being destroyed.

Original class

CosEventChannelAdmin::ProxyPushConsumer Interface

IDL syntax

```
void connect_push_supplier (in CosEventComm::PushSupplier push_supplier)
      raises (AlreadyConnected);
```

Parameters

push_supplier
The PushSupplier connecting to the EventChannel through this ProxyPushConsumer. If the push_supplier is NULL, the ProxyPushConsumer can disconnect without invoking disconnect_push_supplier on the PushSupplier.

Exceptions

AlreadyConnected
This exception is thrown if the connection has already been established between the ProxyPushConsumer and a PushSupplier. You can ignore this exception and use the existing connection.

Remarks

This method is designed to be subclassed and overridden. This method is called by clients. Since there is no implementation in this method, the overridden method does not need to pass the call to this parent.

Example

```
CosEventChannelAdmin::EventChannel_ptr ec = NULL;
CosEventComm_PushSupplier_Impl *pushSupplier;
CosEventChannelAdmin::SupplierAdmin_ptr sa;
CosEventChannelAdmin::ProxyPushConsumer_ptr pxyPushConsumer;
   ...    // Initialize server and get object reference
          // Create pushSupplier
pushSupplier = new CosEventComm_PushSupplier_Impl;
          // Get Supplier Admin through EventChannel.
sa = ec->for_suppliers();
          // Get ProxyPushConsumer through Supplier Admin
```

```
pxyPushConsumer = sa->obtain_push_consumer();
          // Connect PushSupplier to the EventChannel
pxyPushConsumer->connect_push_supplier(pushSupplier);
...
```

ProxyPushSupplier Interface

Defines the second step for connecting push consumers to the event channel.

File stem

CosEventChannelAdmin.idl

Intended usage

ProxyPushSupplier is an abstract base class. Instances of this class serve as front-ends through which a PushConsumer gets events from an EventChannel.

When a new event is given to the EventChannel (either due to a push from a PushSupplier or as a result of a pull from a PullSupplier), this event can be pushed on a PushConsumer of the EventChannel (through objects of the ProxyPushSupplier class).

Ancestor interfaces

PullSupplier Interface

IDL syntax

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

Supported operations

ProxyPushSupplier::connect_push_consumer
ProxyPushSupplier::disconnect_push_supplier

ProxyPushSupplier::connect_push_consumer

Connects a given PushConsumer to this ProxyPushSupplier in EventChannel. This method is part of the procedure for establishing a connection between a PushConsumer and an EventChannel.

Original class

CosEventChannelAdmin::ProxyPushSupplier Interface

IDL syntax

```
void connect_push_consumer
  (in CosEventComm::PushConsumer push_consumer)
    raises (AlreadyConnected);
```

Parameters

push_consumer
The PushConsumer that is connecting to the EventChannel through this ProxyPushSupplier.

Exceptions

AlreadyConnected
This exception is thrown if the connection has already been established between the ProxyPushSupplier and a PushConsumer. You can ignore this exception and use the existing connection.

Remarks

This method is designed to be subclassed and overridden. This method is called by clients. Since there is no implementation in this method, the overridden method does not need to pass the call to this parent.

Example

```
CosEventChannelAdmin::EventChannel_ptr ec = NULL;
CosEventComm_PushConsumer_Impl *pushConsumer;
CosEventChannelAdmin::ConsumerAdmin_ptr ca;
CosEventChannelAdmin::ProxyPushSupplier_ptr pxyPushSupplier;
 ...    // Initialize server and get object reference
        // Create pushConsumer
pushConsumer = new CosEventComm_PushConsumer_Impl;
        // Get Consumer Admin through Event Channel.
ca = ec->for_consumers();
        // Get ProxyPushSupplier through Consumer Admin
pxyPushSupplier = ca->obtain_push_supplier();
        // Connect PushConsumer to the Event Channel
pxyPushSupplier->connect_push_consumer(pushConsumer);
...
```

ProxyPushSupplier::disconnect_push_supplier

This overrides the initial implementation provided in the CosEventComm::PushSupplier::disconnect_push_supplier Operation.

Parent operation

CosEventComm::PushSupplier::PushSupplier::disconnect_push_supplier Operation

Remarks

This method informs a ProxyPushSupplier that it is no longer connected to a PushConsumer. After this call is made, the PushConsumer should not expect to receive any events from the ProxyPushSupplier.

SupplierAdmin Interface

Defines the first step for connecting suppliers to the event channel; clients use it to obtain proxy consumers.

File stem

CosEventChannelAdmin.idl

Intended usage

SupplierAdmin is an abstract base class. Instances of this class are used to make connections between suppliers and the EventChannel. Operations of this class create instances of proxy consumer objects connected to the associated event channel.

IDL syntax

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

Supported operations

SupplierAdmin::obtain_push_consumer
SupplierAdmin::obtain_pull_consumer

SupplierAdmin::obtain_push_consumer

Returns a ProxyPushConsumer object. The ProxyPushConsumer object is then used to connect a PushSupplier.

Original class

CosEventChannelAdmin::SupplierAdmin Interface

IDL syntax

```
ProxyPushConsumer obtain_push_consumer();
```

Return value

ProxyPushConsumer
Returns a new ProxyPushConsumer, by which a push consumer can connect to the event channel.

Exceptions

None

Remarks

This method is designed to be subclassed and overridden. This method is called by clients. Since there is no implementation in this method, the overridden method does not need to pass the call to this parent.

Example

```
CosEventChannelAdmin::EventChannel_ptr ec = NULL;
CosEventChannelAdmin::SupplierAdmin_ptr sa;
CosEventChannelAdmin::ProxyPushConsumer_ptr pxyPushConsumer;
...       //get the EventChannel

           // Get Supplier Admin through Event Channel
sa = ec->for_suppliers();
          // Get ProxyPushConsumer through Supplier Admin
pxyPushConsumer = sa->obtain_push_consumer();
```

SupplierAdmin::obtain_pull_consumer

Returns a ProxyPullConsumer object. The ProxyPullConsumer object is then used to connect a PullSupplier.

Original class

CosEventChannelAdmin::SupplierAdmin Interface

IDL syntax

```
ProxyPullConsumer obtain_pull_consumer();
```

Return value

ProxyPullConsumer
Returns a new ProxyPullConsumer that can be used by a pull consumer to connect to the event channel.

Exceptions

None

Remarks

This method is designed to be subclassed and overridden. This method is called by clients. Since there is no implementation in this method, the overridden method does not need to pass the call to this parent.

Example

```
CosEventChannelAdmin::EventChannel_ptr ec = NULL;
CosEventChannelAdmin::SupplierAdmin_ptr sa;
CosEventChannelAdmin::ProxyPullConsumer_ptr pxypullConsumer;
...       //get the EventChannel

          // Get Supplier Admin through Event Channel
sa = ec->for_suppliers();
          // Get ProxyPullConsumer through Supplier Admin
pxypullConsumer = sa->obtain_pull_consumer();
...
```

Chapter 5. CosEventComm in the Event Service

The other modules in the Event Service are:
- CosEventChannelAdmin
- IExtendedEventChannelAdmin

Note: The Event Service is not supported by OS/390 Component Broker.

CosEventComm Module

Allows applications to implement the PushSuppliers, PushConsumers, PullSuppliers, and PullConsumers which are used in the event supplier/consumer programming module.

Not supported by OS/390 Component Broker

File name

CosEventComm.idl

Intended usage

The CosEventComm Module is an abstract class which allows applications to implement the PushSuppliers, PushConsumers, PullSuppliers, and PullConsumers which are used in the event supplier/consumer programming module.

This module is implemented by the applications and is not part of this product. Any method in the interfaces of this module can be invoked from any client as long as the object reference is obtained. This module gives the standard definition of *consumers* and *suppliers* of events. There are two models for passing events:
- push model - The supplier pushes events on the consumer.
- pull model - The consumer pulls events from the supplier.

Both PushConsumer and PullSupplier have to be implemented as servers in order to enable the push method and the pull and try_pull methods.

If the PushSupplier and PullConsumer decide not to implement the disconnect_push_supplier and disconnect_pull_consumer methods, then the PushSupplier and PullConsumer can be implemented as client only applications.

Exceptions

Disconnected

This exception is raised if the event communication has already been disconnected. Reestablish the required connection.

Interfaces

CosEventComm::PushSupplier Interface
CosEventComm::PushConsumer Interface
CosEventComm::PullSupplier Interface
CosEventComm::PullConsumer Interface

PullConsumer Interface

Provides a base interface from which all pull consumers inherit.

File stem

CosEventComm.idl

Intended usage

An abstract base class from which all pull consumers inherit.

IDL syntax

```
interface PullConsumer
{
     void disconnect_pull_consumer();
}
```

Supported operations

PullConsumer::disconnect_pull_consumer

PullConsumer::disconnect_pull_consumer

Informs a PullConsumer that it is no longer connected to a PullSupplier. After this call is made, the PullConsumer object reference of that PullSupplier is disposed.

Original class

CosEventComm::PullConsumer Interface

IDL syntax

```
void disconnect_pull_consumer ();
```

Exceptions

None

Remarks

This method is designed to be subclassed and overridden. This method is called by clients. Since there is no implementation in this method, the overridden method does not need to pass the call to this parent.

Example

This method can be implemented by the PullConsumer as follows:

```
#include "PullCons.ih"
void CosEventComm_PullConsumer_Impl:disconnect_pull_consumer()
{
  cout << "Enter - disconnect_pull_consumer!" << endl;
}
```

This method can be implemented by the PullSupplier as follows:

```
CosEventComm::PullSupplier_ptr PullSupplier = NULL;
 ...    // Initialize server and get object reference
        // disconnect PullConsumer from the PullSupplier
PullConsumer->disconnect_pull_consumer();
...
```

PullSupplier Interface

Provides a base interface from which all pull suppliers inherit.

File stem

CosEventComm.idl

Intended usage

An abstract base class from which all pull suppliers inherit.

IDL syntax

```
interface PullSupplier
{
   any pull () raises(Disconnected);
   any try_pull (out boolean has_event)
               raises(Disconnected);
   void disconnect_pull_supplier();
};
```

Supported operations

PullSupplier::pull
PullSupplier::try_pull

PullSupplier::pull

A consumer requests event data from the supplier by invoking the pull method. The pull method blocks until the event data is available or until a standard CORBA exception is raised. The pull method returns the event data to the consumer. If the event communication has already been disconnected, the Disconnected exception is raised.

Original class

CosEventComm::PullSupplier Interface

IDL syntax

```
any pull () raises (Disconnected);
```

Return value

The caller owns the storage of the any value that is returned.

Exceptions

Disconnected
This exception is raised if the event communication has already been disconnected between the Pull Consumer and the (Proxy)PullSupplier. Reestablish the required connection.

Remarks

This method is designed to be subclassed and overridden. This method is called by clients. Since there is no implementation in this method, an overridden method does not need to pass the call to this parent.

Example

This method can be implemented by the PullSupplier as follows:

```
#include "PullSupp.ih"
CORBA::Any* CosEventComm_PullSupplier_Impl:: pull()
{
  char *str = CORBA::string_alloc(50);
  CORBA::Any *data = new CORBA::Any;
  printf("\nEnter the Event for Pulling: ");
  gets(str);
  *data <<= str;
  CORBA::string_free(str);
  return (data);
}
```

This method can be invoked by the PullConsumer as follows:

```
CosEventComm::PullSupplier_ptr PullSupplier = NULL;
CORBA::Any *data = new CORBA::Any;
 ...    // Initialize server and get object reference
        // pull data from the PullSupplier
data = PullSupplier->pull();
...
```

PullSupplier::try_pull

A consumer requests event data from the supplier by invoking the try_pull method on the supplier. The try_pull method does not block. That is, if the event data is available, the try_pull method returns the event data and sets the has_event parameter to true; if the event data is not available, the try_pull method sets the has_event parameter to false and the event data is returned with an undefined value. If the event communication has already been disconnected, the Disconnected exception is raised.

Original class

CosEventComm::PullSupplier Interface

IDL syntax

```
any try_pull (out boolean has_event)   raises (Disconnected);
```

Parameters

has_event
Indicates whether an event was returned.

Return value

If the value of has_event is TRUE, data of type any is returned. Otherwise, data of type long with an unspecified value is returned.

Exceptions

Disconnected
This exception is raised if the event communication has already been disconnected between the Pull Consumer and the (Proxy)PullSupplier. Reestablish the required connection.

Remarks

This method is designed to be subclassed and overridden. This method is called by clients. Since there is no implementation in this method, the overridden method does not need to pass the call to this parent.

Example

This method can be implemented by the PullSupplier as follows:

```
#include "PullSupp.ih"
CORBA::Any*
CosEventComm_PullSupplier_Impl::try_pull(CORBA::Boolean&has_event)
{
  char *str = CORBA::string_alloc(50);
  CORBA::Any *data = new CORBA::Any;
  char selectc[256];
  int select;
  printf("\nSelect 1 - to return an event:\n");
  printf("Select 2 - not to return an event:\n\n");
  gets(selectc);
  select = atoi(selectc);
  if (select == 1)           // return an event
  {
    has_event = TRUE;
    printf("\nEnter the Event for try_pulling: ");
    gets(str);
    *data <<= str;
  }
  else        // not to return an event
    has_event = FALSE;
```

This method can be invoked by the PullConsumer as follows:

```
CosEventComm::PullSupplier_ptr PullSupplier = NULL;
CORBA::Any *data = new CORBA::Any;
CORBA::Boolean has_event;
   ...    // Initialize server and get object reference
          // pull data from the PullSupplier
data = PullSupplier->try_pull(has_event);
...
```

PullSupplier::disconnect_pull_supplier

This method informs a PullSupplier that it is no longer connected to a PullConsumer. After this call is made, the PullSupplier object reference of that PullConsumer is disposed.

Original class

CosEventComm::PullSupplier Interface

IDL syntax

```
void disconnect_pull_supplier ();
```

Exceptions

None

Remarks

This method is designed to be subclassed and overridden. This method is called by clients. Since there is no implementation in this method, the overridden method does not need to pass the call to this parent.

Example

This method can be implemented by the PullSupplier as follows:

```
#include "PullSupp.ih"
void  CosEventComm_PullSupplier_Impl::disconnect_pull_supplier()
{
    cout << "Enter - disconnect_pull_supplier!" << endl;
}
```

This method can be implemented by the PullConsumer as follows:

```
CosEventComm::PullSupplier_ptr PullSupplier = NULL;
 ...    // Initialize server and get object reference
        // disconnect PullSupplier from the PullConsumer
PullSupplier->disconnect_pull_supplier();
...
```

PushConsumer Interface

Provides a base interface from which all push consumers inherit.

File stem

CosEventComm.idl

Intended usage

An abstract base class from which all push consumers inherit.

IDL syntax

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

Supported operations

PushConsumer::push
PushConsumer::disconnect_push_consumer

PushConsumer::push

This method is called by a PushSupplier to pass data to this PushConsumer. If the event communication has already been disconnected, the Disconnected exception is raised.

Original class

CosEventComm::PushConsumer Interface

IDL syntax

```
void push (in any data)   raises (Disconnected);
```

Parameters

data The event data of type any.

Exceptions

Disconnected
This exception is raised if the event communication has already been disconnectedbetween the Push Supplier and the (Proxy)PushConsumer. Reestablish the required connection.

Remarks

This method is designed to be subclassed and overridden. This method is called by clients. Since there is no implementation in this method, the overridden method need not pass the call to the parent.

Example

This method can be implemented by the PushConsumer as follows:

```
#include "PushCons.ih"
#include <stdio.h>
void CosEventComm_PushConsumer_Impl::push(const CORBA::Any& data)
{
  char *str;

  data >>= str;
  printf("%s\n",  str);
}
```

This method can be invoked by the PushSupplier as follows:

```
CosEventComm::PushConsumer_ptr PushConsumer = NULL;
CORBA::Any *data = new CORBA::Any;
CORBA::Long n;
```

```
...    // Initialize server and get object reference
       // push "n" from the PushSupplier to the PushConsumer
 *data <<= n;
 PushConsumer->push(data);
 ...
```

PushConsumer::disconnect_push_consumer

Informs a PushConsumer that it is no longer connected to a PushSupplier. After this call is made, the PushConsumer object reference of that PushSupplier is disposed.

Original class

CosEventComm::PushConsumer Interface

IDL syntax

```
void disconnect_push_consumer ();
```

Exceptions

None

Remarks

This operation is designed to be subclassed and overridden. This operation is called by clients. Since there is no implementation in this operation, the overridden operation need not pass the call to the parent.

Example

This method may be implemented by the PushConsumer as follows:

```
#include "PushCons.ih"
void CosEventComm_PushConsumer_Impl::

disconnect_push_consumer()
{
  cout << "Enter - disconnect_push_consumer!" << endl;
}
```

This method may be envoked by the PushSupplier as follows:

```
CosEventComm::PushConsumer_ptr PushConsumer = NULL;
 ...    // Initialize server and get object reference
        // disconnect PushConsumer from the PushSupplier
PushConsumer->disconnect_push_consumer();
...
```

PushSupplier Interface

Provides a base interface from which all push suppliers inherit.

File stem

CosEventComm.idl

Intended usage

An abstract base class, from which push suppliers can inherit.

IDL syntax

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

Supported operations

PushSupplier::disconnect_push_supplier

PushSupplier::disconnect_push_supplier

Informs a PushSupplier object that it is no longer connected to a PushConsumer object. After this call is made, the PushSupplier object reference of that PushConsumer is disposed.

Original class

CosEventComm::PushSupplier Interface

IDL syntax

```
void disconnect_push_supplier ();
```

Remarks

This method is designed to be subclassed and overridden. This method is called by clients. Since there is no implementation in this method, the overridden method need not pass the call to parent.

Example

This method can be implemented by the PushSupplier as follows:

```
#include "PushSupp.ih"
void CosEventComm_PushSupplier_Impl::
```

```
disconnect_push_supplier()
{
  cout << "Enter - disconnect_push_supplier!" << endl;
}
```

This method can be invoked by the PushConsumer as follows:

```
CosEventComm::PushSupplier_ptr PushSupplier = NULL;
     ...    // Initialize server and get object reference
            // disconnect PushSupplier from the PushConsumer
PushSupplier->disconnect_push_supplier();
```

Chapter 6. CosLifeCycle in the LifeCycle Service

The other modules in the LifeCycle Service are:
- IExtendedLifeCycle
- ILifeCycleLocalObjectImpl
- ILifeCycleManagedClient

CosLifeCycle Module

Provides abstract interfaces defined by the OMG CORBA services LifeCycle standard.

File name

CosLifeCycle.idl

Types

```
typedef CosNaming::Name Key;
typedef Object Factory;
typedef sequence <Factory> Factories;
typedef struct NVP {
  CosNaming::Istring name;
  any value;
} NameValuePair;
typedef sequence <NameValuePair> Criteria;
```

Exceptions

```
exception NoFactory {Key search_key;};
exception NotCopyable {string reason; };
exception NotMovable {string reason; };
exception NotRemovable {string reason; };
exception InvalidCriteria {Criteria invalid_criteria;};
exception CannotMeetCriteria {Criteria unmet_criteria;}
```

Interfaces

CosLifeCycle::FactoryFinder Interface
CosLifeCycle::GenericFactory Interface
CosLifeCycle::LifeCycleObject Interface

FactoryFinder Interface

Provides the OMG definition for FactoryFinder. It is abstract and does not inherit any other interfaces.

IDL syntax

```
interface FactoryFinder
{
   Factories find_factories(in Key factory_key)
      raises(NoFactory);
};
```

Supported operations

FactoryFinder::find_factories

FactoryFinder::find_factories

Finds all factories that meet the constraints specified by the factory_key parameter and the rules configured into the FactoryFinder itself.

Original interface

CosLifeCycle::FactoryFinder

IDL syntax

```
Factories find_factories (in Key factory_key)
   raises(NoFactory);
```

Parameters

factory_key

The factory_key parameter is of type Key. Key is a CosNaming::Name, which is a sequence of CosNaming::NameComponents. Each NameComponent is a two-part name: id and kind.

See the sections entitled “Factory Keys” and “Factory Key Structures and Strings”, in the *Component Broker Advanced Programming Guide*, for complete information on the meanings and syntax of these values.

Return value

Factories

The sequence of object references of the Factories found.

Exceptions

CosLifeCycle::NoFactory - indicates that a factory could not be found. The factory finder could not find any factories that met both the criteria specified in the Key and the scope configured into the factory finder itself. A minor code of LCERR_FAC_FIND_INVALID_KEY (0x49420201) indicates that the key was not in the correct form. Check that the key used in the find factory operation is correct. Try using a factory_finder with a less restrictive scope. Verify that the registration of the expected factories did not result in any errors by checking the activity log.

Remarks

The find_factories operation will return only after finding all factories that meet the caller's request. Since the caller's request may be very general, it is possible that Lifecycle may have to check a large number of factories before returning the (potentially very large) list of factories.

Once a factory has been found, it can be used to create objects with the interface type specified in the factory_key.

Example

```
// C ++ example
// Note: for code clarity, some standard client seteup, exception handling,
// error checking and cleanup have been omitted from this example.

   extern IExtendedNaming::NamingContext_ptr root_context; // previously
                                                           // initialized

   extern CosLifeCycle::FactoryFinder_var  finder; // previously initialized

// find the factories which supports the "MyTest" interface

   cout << "find MyTest factories \n";
   cout.flush();
   CosLifeCycle::Key  key(1);

   key.id = CORBA::string_dup("MyTest");
   key.kind = CORBA::string_dup("object interface");

   CosLifeCycle::Factories_var temp_seq = finder->find_factories(key);
```

GenericFactory Interface

Provides an abstract interface for developers that wish to provide an implementation of this interface on their factories. It does not inherit any other interfaces. The LifeCycle service does not provide an implementation of this interface. The IHome objects introduced by the programming model provide support for this interface through implementations provided by the instance managers.

IDL syntax

```
interface GenericFactory
{
   boolean supports(in Key k);
   Object create_object(in Key k,
                        in Criteria the_criteria)
      raises(NoFactory, InvalidCriteria,
             CannotMeetCriteria)
};
```

Supported operations

GenericFactory::create_object
GenericFactory::supports

GenericFactory::create_object

Creates the object, using the key and criteria.

Original interface

CosLifeCycle::GenericFactory

IDL syntax

```
Object create_object(in Key k,
                     in Criteria the_criteria)
   raises(NoFactory, InvalidCriteria,
          CannotMeetCriteria);
```

Parameters

k Name which identifies the desired object to be created.

the_criteria
Sequence of Name Value Pairs which are passed through to the factory.

Return value

Object The object that is created.

Exceptions

CORBA standard exceptions and the following user exceptions:
CosLifeCycle::NoFactory - The factory cannot create the object specified by the key.
CosLifeCycle::InvalidCriteria - The target factory does not understand the criteria.
CosLifeCycle::CannotMeetCriteria - The target factory can not satisfy the criteria.

GenericFactory::supports

Returns true if the generic factory can create an object, given the key. Otherwise, false is returned.

Original class

CosLifeCycle::GenericFactory

IDL syntax

```
boolean supports(in Key k);
```

Parameters

k Key.

Return value

true Indicates that the generic factory can create an object, given the key.

false Indicates that the generic factory cannot create an object.

LifeCycleObject Interface

Provides the OMG definition for LifeCycleObject. It is abstract and does not inherit any other interfaces. The LifeCycle service does not provide an implementation of this interface. Managed objects support the remove method on this interface but do not support the copy and move methods provided by this interface.

IDL syntax

```
interface LifeCycleObject
{
   LifeCycleObject copy(in FactoryFinder there,
                        in Criteria the_criteria)
      raises(NoFactory, NotCopyable, InvalidCriteria,
             CannotMeetCriteria);
   void move(in FactoryFinder there,
             in Criteria the_criteria)
      raises(NoFactory, NotMovable, InvalidCriteria,
             CannotMeetCriteria);
   void remove()
      raises(NotRemovable);
};
```

Supported operations

LifeCycleObject::copy
LifeCycleObject::move
LifeCycleObject::remove

LifeCycleObject::copy

Makes a copy of an object, using the FactoryFinder specified by the there parameter and a set of criteria that is understood by the factory.

Original interface

CosLifeCycle::LifeCycleObject

IDL syntax

```
LifeCycleObject copy (in FactoryFinder there,
                      in Criteria the_criteria)
   raises(NoFactory, NotCopyable, InvalidCriteria,
          CannotMeetCriteria);
```

Parameters

there Defines the FactoryFinder.

the_criteria
Sequence of the Name Value Pairs which are passed through to the factory.

Return value

LifeCycleObject
The copy of the object.

Exceptions

CORBA standard exceptions and the following user exceptions:
CosLifeCycle::NoFactory - indicates that a factory could not be found.
CosLifeCycle::NotCopyable - the object refuses to copy itself.
CosLifeCycle::InvalidCriteria - the target factory does not understand the criteria.
CosLifeCycle::CannotMeetCriteria - the target factory can not satisfy the criteria.

LifeCycleObject::move

Moves the object, using the FactoryFinder specified by the there parameter and a set of criteria that is understood by the factory.

Original interface

CosLifeCycle::LifeCycleObject

IDL syntax

```
void move (in FactoryFinder there,
           in Criteria the_criteria)
   raises (NoFactory, NotMovable, InvalidCriteria,
           CannotMeetCriteria);
```

Parameters

there Describes the FactoryFinder.

the_criteria
Sequence of the Name Value Pairs which are passed through to the factory.

Exceptions

CORBA standard exceptions and the following user exceptions:
CosLifeCycle::NoFactory - indicates that a factory could not be found.
CosLifeCycle::NotMoveable - the object refuses to move itself.
CosLifeCycle::InvalidCriteria - the target factory does not understand the criteria.
CosLifeCycle::CannotMeetCriteria - the target factory can not satisfy the criteria.

LifeCycleObject::remove

Deletes the object.

Original interface

CosLifeCycle::LifeCycleObject

IDL syntax

```
void remove()
   raises(NotRemovable);
```

Exceptions

CosLifeCycle::NotRemoveable - the object refuses to remove itself.

Chapter 7. CosNaming in the Naming Service

The other modules in the Naming Service are:
- IExtendedNaming
- IExtendedNamingStringSyntax
- NamingStringSyntax

CosNaming Module

Supports methods that allow the assigning of a name to an object (that is, creating an object-name binding in a context), then finding the object using the assigned name.

File name

CosNaming.idl

Intended usage

The key class in this module is the NamingContext class. Operations in this class can be used to build and manipulate a naming space. A naming space is distributed and federated. Objects in this naming space are managed objects.

Types

```
typedef string Istring;

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

enum BindingType {nobject, ncontext};

struct Binding {
  Name binding_name;
  BindingType binding_type;
};
typedef sequence <Binding> BindingList;
```

Interfaces

CosNaming::BindingIterator Interface
CosNaming::NamingContext Interface

BindingIterator Interface

Provides support for the Object Management Group (OMG) binding iteration.

File name

CosNaming.idl

Intended usage

This class is instantiated and returned as an out parameter in the CosNaming::NamingContext::list method if the targeted naming context contains more name-object bindings than requested.

Types

```
typedef string Istring;

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

enum BindingType {nobject, ncontext};

struct Binding {
  Name binding_name;
BindingType binding_type;
};
typedef sequence <Binding> BindingList;
```

Exceptions

CORBA standard exceptions.

Supported operations

BindingIterator::destroy
BindingIterator::next_n
BindingIterator::next_one

BindingIterator::destroy

Destroys the iterator and frees allocated memory.

Original interface

CosNaming::BindingIterator Interface

IDL syntax

```
void destroy();
```

Exceptions

CORBA standard exceptions.

Remarks

This operation is intended to be used by client applications. It is not typically overridden.

Example

The following example demonstrates the usage of the CosNaming Module.

```
// A CosNaming usage example.
// For simplicity, error and exception checking and cleanup are ommitted.

#include <CosNaming.hh>
#include <stdlib.h>
#include <fstream.h>
#define filename1 "NMUTST1.OUT"
#define filename2 "NMUTST1.OUT"


// Make the name "vehicles"
CosNaming::Name *vehiclesBindingName = new CosNaming::Name;
vehiclesBindingName->length( 1 );
(*vehiclesBindingName)[0].id   = CORBA::string_dup("vehicles");
(*vehiclesBindingName)[0].kind = CORBA::string_dup("");


// Create a new naming context vehiclesNamingContext and bind it to the
// root rootNamingContext with the name "vehicles"
CosNaming::NamingContext_ptr vehiclesNamingContext =
  rootNamingContext->bind_new_context(*vehiclesBindingName);


// Make the name "vehicles.large"
CosNaming::Name *largevehiclesBindingName = new CosNaming::Name;
largevehiclesBindingName->length( 1 );
(*largevehiclesBindingName)[0].id   = CORBA::string_dup("vehicles");
(*largevehiclesBindingName)[0].kind = CORBA::string_dup("large");


// create a new naming context largevehiclesNamingContext and bind
// it to the naming context vehiclesNamingContext with the name
// "vehicles.large"
CosNaming::NamingContext_ptr largevehiclesNamingContext =
  vehiclesNamingContext->bind_new_context(
  *largevehiclesBindingName);


// Make the name "vans"
CosNaming::Name *vansBindingName = new CosNaming::Name;
vansBindingName->length( 1 );
(*vansBindingName)[0].id   = CORBA::string_dup("vans");
```

```
(*vansBindingName)[0].kind = CORBA::string_dup("");


// create a new naming context vansNamingContext and bind it to the
// naming context vehiclesNamingContext with the name "vans"
CosNaming::NamingContext_ptr vansNamingContext =
  vehiclesNamingContext->bind_new_context(*vansBindingName);


// Make the name "trucks"
CosNaming::Name *trucksBindingName = new CosNaming::Name;
trucksBindingName->length( 1 );
(*trucksBindingName)[0].id   = CORBA::string_dup("trucks");
(*trucksBindingName)[0].kind = CORBA::string_dup("");


// create a new naming context trucksNamingContext and bind it to the
// naming context vehiclesNamingContext with the name "trucks"
CosNaming::NamingContext_ptr trucksNamingContext =
  vehiclesNamingContext->bind_new_context(*trucksBindingName);


// Make the name "chrysler"
CosNaming::Name *avehicleBindingName = new CosNaming::Name;
avehicleBindingName->length( 1 );
(*avehicleBindingName)[0].id   = CORBA::string_dup("chrysler");
(*avehicleBindingName)[0].kind = CORBA::string_dup("");


// Create an object avehicleObject
ifstream strm1(filename1);
char refStr[2048];
memset(refStr, 2048, '\0');
strm1 >> refStr;
CORBA::Object_ptr avehicleObject = orb_p->string_to_object(refStr);


// Bind the object avehicleObject to the naming context
// vansNamingContext with the name "chrysler"
vansNamingContext->bind(*avehicleBindingName, avehicleObject);


// Create another object anothervehicleObject
ifstream strm2(filename2);
memset(refStr, 2048, '\0');
strm2 >> refStr;
CORBA::Object_ptr anothervehicleObject = orb_p->string_to_object(refStr);


// Rebind the object anothervehicleObject to the naming
// context vansNamingContext with the name "chrysler"
vansNamingContext->rebind(*avehicleBindingName, anothervehicleObject);


// Bind the context vansNamingContext to the context
```

```
// vehiclesNamingContext with the name "vans"
largevehiclesNamingContext->bind_context(*vansBindingName,
  vansNamingContext);


// Make the name "vans.mini"
CosNaming::Name *miniVansBindingName = new CosNaming::Name;
miniVansBindingName->length( 1 );
(*miniVansBindingName)[0].id   = CORBA::string_dup("vans");
(*miniVansBindingName)[0].kind = CORBA::string_dup("mini");


// Rebind the context vansNamingContext to the context
// vehiclesNamingContext with the name "vans.mini"
largevehiclesNamingContext->rebind_context(*miniVansBindingName,
  vansNamingContext);


// Unbind the object bound to vehiclesNamingContext with the
// name "vans"
largevehiclesNamingContext->unbind(*vansBindingName);


// Unbind the object bound to vehiclesNamingContext with the
// name "vans.mini"
largevehiclesNamingContext->unbind(*miniVansBindingName);


// Make the name "vehicles/vans/crysler.mini"
CosNaming::Name *aMiniVansPathName = new CosNaming::Name;
aMiniVansPathName->length( 3 );
(*aMiniVansPathName)[0].id   = CORBA::string_dup("vehicles");
(*aMiniVansPathName)[0].kind = CORBA::string_dup("");
(*aMiniVansPathName)[1].id   = CORBA::string_dup("vans");
(*aMiniVansPathName)[1].kind = CORBA::string_dup("");
(*aMiniVansPathName)[2].id   = CORBA::string_dup("chrysler");
(*aMiniVansPathName)[2].kind = CORBA::string_dup("");


// Resolve the name from the root naming context
rootNamingContext avehicleObject =
  rootNamingContext->resolve(*aMiniVansPathName);


// list only one binding in the naming root context rootNamingContext
// The remaining bindings can be retrieved from the binding iterator bi.
CosNaming::BindingList_var bl;
CosNaming::BindingIterator_var bi;
vehiclesNamingContext->list(1, bl, bi);


// Retrieve the next binding from the binding iterator
CosNaming::Binding_var b;
bi->next_one(b);
```

```
// Retrieve the next 2 bindings from the binding iterator
CosNaming::BindingList_var bl1;
bi->next_n(2, bl1);


// Destroy the naming context vehiclesNamingContext
largevehiclesNamingContext->destroy();


// Destroy the binding iterator bi
bi->destroy();
```

BindingIterator::next_n

Retrieves at most the specified number of name-object bindings. This operation allows clients to iterate through the bindings in the iterator.

Original interface

CosNaming::BindingIterator Interface

IDL syntax

```
boolean next_n(
  in unsigned long how_many,
  out CosNaming::BindingList blist);
```

Parameters

how_many
The maximum number of bindings to be returned.

blist The returned BindingList.

Return value

TRUE Indicates that more bindings exist.

FALSE Indicates to the client that there are no more bindings.

Exceptions

CORBA standard exceptions.

Remarks

This operation is intended to be used by client applications. It is not typically overridden.

Example

See the CosNaming Usage example for “BindingIterator::destroy” on page 348.

BindingIterator::next_one

Retrieves the next name-object binding.

Original interface

CosNaming::BindingIterator Interface

IDL syntax

```
boolean next_one(out CosNaming::Binding binding);
```

Parameters

binding
The returned Binding.

Return value

TRUE Indicates that the next binding exists.

FALSE Indicates that the next binding does not exist.

Exceptions

CORBA standard exceptions.

Remarks

This operation is intended to be used by client applications. It is not typically overridden.

Example

See the CosNaming Usage example for “BindingIterator::destroy” on page 348.

NamingContext Interface

Provides support for creating and manipulating a system naming tree, binding a name to an object in a naming context, retrieving an object from a naming context using the object name, and listing the bindings in a naming context.

File name

CosNaming.idl

Intended usage

This interface provides the operations necessary to create and manipulate a system naming tree, to bind a name to an object in a naming context, to retrieve an object from a naming context using the object name, and to list the bindings in a naming context.

Types

```
typedef string Istring;

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

enum BindingType {nobject, ncontext};

struct Binding {
  Name binding_name;
  BindingType binding_type;
};
typedef sequence <Binding> BindingList;
```

Exceptions

CORBA standard exceptions and the following user exceptions:

CosNaming::NamingContext::AlreadyBound - raised to indicate that an object is already bound to the name. Re-binding operations unbind the name, then rebinds the name without raising this exception.

CosNaming::NamingContext::CannotProceed{NamingContext ctx; Name rest_of_name;}; - raised to indicate that the implementation has given up for some reason. The client may be able to continue the operation using the returned naming context.

CosNaming::NamingContext::InvalidName - raised to indicate that the name is invalid. A name with a length of zero is invalid. (This exception may be raised upon further implementation restrictions.)

CosNaming::NamingContext::NotFound{NotFoundReason why; Name rest_of_name;}; - raised to indicate that the name does not identify a binding. If a compound name is passed as an argument for the bind operation, it traverses multiple contexts. A NotFound exception is raised if any of the intermediate contexts cannot be resolved.

Supported operations

NamingContext::bind
NamingContext::bind_context

NamingContext::bind_new_context
NamingContext::destroy
NamingContext::list
NamingContext::rebind
NamingContext::rebind_context
NamingContext::resolve
NamingContext::unbind

NamingContext::bind

Creates a binding in a naming context.

Original interface

CosNaming::NamingContext Interface

IDL syntax

```
void bind(
  in CosNaming::Name name,
  in Object obj);
```

Parameters

name The name for the binding.

obj The object to be bound.

Exceptions

CORBA standard exceptions and the following user exceptions:

CosNaming::NamingContext::NotFound - raised to indicate that the name does not identify a binding. If a compound name is passed as an argument for the bind operation, it traverses multiple contexts. A NotFound exception is raised if any of the intermediate contexts cannot be resolved.

CosNaming::NamingContext::CannotProceed - raised to indicate that the implementation has given up for some reason. The client may be able to continue the operation using the returned naming context.

CosNaming::NamingContext::InvalidName - raised to indicate that the name is invalid. A name with a length of zero is invalid. (This exception may be raised upon further implementation restrictions.)

CosNaming::NamingContext::AlreadyBound - raised to indicate that an object is already bound to the name. Re-binding operations unbind the name, then rebinds the name without raising this exception.

Remarks

This operation is intended to be used by client applications. It is not typically overridden.

This operation creates a binding of a name to an object in a naming context. Binding a name to an object in a naming context creates a name-object association relative to the target naming context. Once an object is bound, it can be found through the resolve operation. Naming contexts that are bound using bind do not participate in name resolution when compound names are resolved - bind_context should be used to bind naming context objects.

This operation runs *resolve* to traverse a compound name. An object can be bound to multiple names in a context or across multiple contexts. Within a context, names of an object must be unique. That is, only one object can be bound to a particular name in a naming context.

Example

See the CosNaming Usage example for “BindingIterator::destroy” on page 348.

NamingContext::bind_context

Creates a naming context binding.

Original interface

CosNaming::NamingContext Interface

IDL syntax

```
void bind_context(
  in CosNaming::Name name,
  in CosNaming::NamingContext naming_context);
```

Parameters

name The name for the binding.

naming_context
The naming context object to be bound.

Exceptions

CORBA standard exceptions and the following user exceptions:

CosNaming::NamingContext::NotFound - raised to indicate that the name does not identify a binding. If a compound name is passed as an argument for the bind operation, it traverses multiple contexts. A NotFound exception is raised if any of the intermediate contexts cannot be resolved.

CosNaming::NamingContext::CannotProceed - raised to indicate that the implementation has given up for some reason. The client may be able to continue the operation using the returned naming context.

CosNaming::NamingContext::InvalidName - raised to indicate that the name is invalid. A name with a length of zero is invalid. (This exception may be raised upon further implementation restrictions.)

CosNaming::NamingContext::AlreadyBound - raised to indicate that an object is already bound to the name. Re-binding operations unbind the name, then rebind the name without raising this exception.

Remarks

This operation is intended to be used by client applications. It is not typically overridden.

This operation creates a naming context binding. Binding a name and a naming context object into a naming context creates a name-object association relative to the target naming context. Naming contexts that are bound using bind_context participate in name resolution when compound names are resolved. This operation is used to extend the naming tree by binding sub-contexts to contexts. Like an object, a naming context can be bound, using bind_context, to multiple names in a context or across multiple contexts. Within a context, the names bound to a context must be unique. That is, only one context can be bound to a particular name in a naming context.

Example

See the CosNaming Usage example for “BindingIterator::destroy” on page 348.

NamingContext::bind_new_context

Creates a new naming context in the same server as the target naming context on which the operation was invoked and binds it to a supplied name.

Original interface

CosNaming::NamingContext Interface

IDL syntax

```
CosNaming::NamingContext bind_new_context(in CosNaming::Name name);
```

Parameters

name The name for the naming context object binding.

Return value

CosNaming::NamingContext

Exceptions

CORBA standard exceptions and the following user exceptions:

CosNaming::NamingContext::NotFound - raised to indicate that the name does not identify a binding. If a compound name is passed as an argument for the bind operation, it traverses multiple contexts. A NotFound exception is raised if any of the intermediate contexts cannot be resolved.

CosNaming::NamingContext::CannotProceed - raised to indicate that the implementation has given up for some reason. The client may be able to continue the operation using the returned naming context.

CosNaming::NamingContext::InvalidName - raised to indicate that the name is invalid. A name with a length of zero is invalid. (This exception may be raised upon further implementation restrictions.)

CosNaming::NamingContext::AlreadyBound - raised to indicate that an object is already bound to the name.

Remarks

This operation is intended to be used by client applications. It is not typically overridden.

The naming context that is created has the same implementation as the target naming context to which it is bound. This new context is created in the same process as that of the target naming context. Note that the target naming context in which the new context is bound is denoted by a name that is equivalent to the name name excluding the last name component of name.

Example

See the CosNaming Usage example for “BindingIterator::destroy” on page 348.

NamingContext::destroy

Destroys a naming context.

Original interface

CosNaming::NamingContext Interface

IDL syntax

```
void destroy();
```

Exceptions

CORBA standard exceptions and the following user exception:

CosNaming::NamingContext::NotEmpty - raised if the naming context contains any bindings.

Remarks

This operation is intended to be used by client applications. It is not typically overridden.

This operation destroys the naming context if the context is empty. The naming context cannot contain bindings for this operation to succeed. It is the responsibility of the client to ensure that all bindings have been removed from the naming context before invoking this operation. Use the unbind operation to remove any bindings in the naming context; for more information, refer to the unbind Operation.

Example

See the CosNaming Usage example for “BindingIterator::destroy” on page 348.

NamingContext::list

Retrieves bindings from a naming context.

Original interface

CosNaming::NamingContext Interface

IDL syntax

```
void list(
  in unsigned long how_many,
  out CosNaming::BindingList blist,
  out CosNaming::BindingIterator biterator);
```

Parameters

how_many
The maximum number of bindings to install into the BindingList.

blist The returned BindingList.

biterator
The returned BindingIterator.

Exceptions

CORBA standard exceptions.

Remarks

This operation is intended to be used by client applications. It is not typically overridden.

This operation etrieves bindings from a naming context. At most, the operation returns a number of bindings equal to how_many in blist. If the naming context contains

additional bindings, a BindingIterator is returned, and the calling program can iterate through the remaining bindings. If the naming context does not contain additional bindings, the BindingIterator is a NIL object reference.

The value of how_many should be less than or equal to a maximum of 1000.

The returned binding list is of type BindingList which contains a list of bindings. Each element in the list is of type Binding. Binding consists of two fields: binding_name which is the name part of the binding and binding_type which is the type of the object part of the binding. A binding type is either an object (nobject) or a naming context (ncontext).

Example

See the CosNaming Usage example for “BindingIterator::destroy” on page 348.

NamingContext::new_context

This operation is not part of the programming model and should not be directly invoked or overridden.

NamingContext::rebind

Recreates a name-object binding in a naming context even if the name is already bound in the naming context.

Original interface

CosNaming::NamingContext Interface

IDL syntax

```
void rebind(
  in CosNaming::Name name,
  in Object obj);
```

Parameters

name The name to be re-bound.

obj The Object to be re-bound.

Exceptions

CORBA standard exceptions and the following user exceptions:

CosNaming::NamingContext::NotFound - raised to indicate that the name does not identify a binding. If a compound name is passed as an argument for the bind operation, it traverses multiple contexts. A NotFound exception is raised if any of the intermediate contexts cannot be resolved.

CosNaming::NamingContext::CannotProceed - raised to indicate that the implementation has given up for some reason. The client may be able to continue the operation using the returned naming context.

CosNaming::NamingContext::InvalidName - raised to indicate that the name is invalid. A name with a length of zero is invalid. (This exception may be raised upon further implementation restrictions.)

Remarks

This operation is intended to be used by client applications. It is not typically overridden.

This operation recreates a name binding in a naming context, even if the name is already bound in the naming context. Rebinding a name and object into a naming context recreates a name-object association relative to the target naming context. Naming contexts that are bound using rebind do not participate in name resolution process when compound names are resolved.

If an object is already bound with the same name, the bound object is replaced by the passed argument obj. If the name-object binding does not exist, the rebind method behaves like the bind method.

As a developer, you can use the rebind method to replace an existing binding. You can use the rebind operation in place of the unbind and bind methods.

Example

See the CosNaming Usage example for “BindingIterator::destroy” on page 348.

NamingContext::rebind_context

Recreates a name-naming context binding in a target naming context, even if the name is already bound in the target naming context.

Original interface

CosNaming::NamingContext Interface

IDL syntax

```
void rebind_context(
  in CosNaming::Name name,
  in CosNaming::NamingContext naming_context);
```

Parameters

name The name to be re-bound.

naming_context
The NamingContext object to be re-bound to the name.

Exceptions

CORBA standard exceptions and the following user exceptions:

CosNaming::NamingContext::NotFound - raised to indicate that the name cannot be resolved into a naming context to perform binding. If a compound name is passed as an argument for the bind operation, it traverses multiple contexts. A NotFound exception is raised if any of the intermediate contexts cannot be resolved.

CosNaming::NamingContext::CannotProceed - raised to indicate that the implementation has given up for some reason. The client may be able to continue the operation using the returned naming context.

CosNaming::NamingContext::InvalidName - raised to indicate that the name is invalid. A name with a length of zero is invalid. (This exception may be raised upon further implementation restrictions.)

Remarks

This operation is intended to be used by client applications. It is not typically overridden.

This operation recreates a binding to a naming context, even if the name is already bound in the naming context. Re-binding a name and a naming context object into a naming context recreates a name-object association relative to the target naming context. Naming contexts that are bound using rebind_context participate in name resolution when compound names are resolved.

The rebind_context operation is used to bind or replace a subcontext. If a context is already bound in a context, the bind operation raises the AlreadyBound exception. However, the rebind method replaces the bound object with the passed object.

Example

See the CosNaming Usage example for “BindingIterator::destroy” on page 348.

NamingContext::resolve

Retrieves an Object bound to a name.

Original interface

CosNaming::NamingContext Interface

IDL syntax

```
Object resolve(in CosNaming::Name name);
```

Parameters

name The name for the name-object binding.

Return value

Object The name of the object bound to the supplied name.

Exceptions

CORBA standard exceptions and the following user exceptions:

CosNaming::NamingContext::NotFound - raised to indicate that the name does not identify a binding. If a compound name is passed as an argument for the bind operation, it traverses multiple contexts. A NotFound exception is raised if any of the intermediate contexts cannot be resolved.

CosNaming::NamingContext::CannotProceed - raised to indicate that the implementation has given up for some reason. The client may be able to continue the operation using the returned naming context.

CosNaming::NamingContext::InvalidName - raised to indicate that the name is invalid. A name with a length of zero is invalid. (This exception may be raised upon further implementation restrictions.)

Remarks

This operation is intended to be used by client applications. It is not typically overridden.

This operation retrieves the object bound to name name in the target naming context. The name name could be a simple name. In that case name should match exactly the name bound to the object in the context. Or, the name name could be a compound name that spans multiple contexts. In this case, name resolution traverses multiple contexts. At each context traversed, the name bound to this context, in its super context, should match exactly the name component corresponding to this traversed context. The last name component of the compound name should match exactly the name bound to the object in the last traversed naming context.

The type of the returned object is not provided. Clients are responsible for ″narrowing″ the object to the appropriate type. Clients typically cast the returned object to a more specialized interface.

Example

See the CosNaming Usage example for “BindingIterator::destroy” on page 348.

NamingContext::unbind

Removes a name-object binding.

Original interface

CosNaming::NamingContext Interface

IDL syntax

```
void unbind(in CosNaming::Name name);
```

Parameters

name The name for the name-object binding.

Exceptions

CORBA standard exceptions and the following user exceptions:

CosNaming::NamingContext::NotFound - raised to indicate that the name does not identify a binding. If a compound name is passed as an argument for the bind operation, it traverses multiple contexts. A NotFound exception is raised if any of the intermediate contexts cannot be resolved.

CosNaming::NamingContext::CannotProceed - raised to indicate that the implementation has given up for some reason. The client may be able to continue the operation using the returned naming context.

CosNaming::NamingContext::InvalidName - raised to indicate that the name is invalid. A name with a length of zero is invalid. (This exception may be raised upon further implementation restrictions.)

Remarks

This operation is intended to be used by client applications. It is not typically overridden.

The unbind operation removes a binding from a context. It unbinds the name *name* from the context. It is used to unregister the name *name* with the Naming Service.

This operation can also be used to unbind a naming context. If the naming context was originally bound using bind_context, rebind_context, bind, or rebind, the operation will be allowed to proceed. However, if this context was originally bound using bind_new_context, then a CORBA::PERSIST_STORE exception will be thrown since this request would result in an orphaned name context (which is not supported). In the case of the CORBA::PERSIST_STORE exception, the user is required to call the destroy() method to unbind the name context.

Example

See the CosNaming Usage example for “BindingIterator::destroy” on page 348.

Chapter 8. CosNotification in the Notification Service

The other modules in the Notification Service are:

- CosNotifyChannelAdmin
- CosNotifyComm
- CosNotifyFilter
- INotifyChannelAdminManagedClient
- INotifyFilterManagedClient

CosNotification Module

The CosNotification module defines the Structured Event data type. In addition, this module provides constant declarations for each of the standard quality of service (QoS) supported by the implementation. Some properties also have associated constant declarations that indicate their possible settings. Finally, administrative interfaces are defined for managing sets of QoS.

Not supported by OS/390 Component Broker

File name

CosNotification.idl

Types

```
typedef CosTrading::PropertySeq OptionalHeaderFields;
typedef CosTrading::PropertySeq FilterableEventBody;
typedef CosTrading::PropertySeq QoSProperties;
typedef CosTrading::PropertySeq AdminProperties;

struct PropertyRange
{
   CosTrading::PropertyName name;
   CosTrading::PropertyValue low_val;
   CosTrading::PropertyValue high_val;
};

typedef sequence<PropertyRange> PropertyRangeSeq;

enum QosError_code
{
   UNSUPPORTED,
   UNAVAILABLE,
   UNSUPPORTED_VALUE,
   UNAVAILABLE_VALUE,
   BAD_NAME,
   BAD_TYPE,
   BAD_VALUE
```

```
};

  struct PropertyError
{
   QoSError_code code;
   PropertyRange available_range;
};

typedef sequence<PropertyError> PropertyErrorSeq;

// Define the Structured Event structure
struct FixedEventHeader
{
   string domain_type;
   string event_type;
   string event_name;
};

struct EventHeader
{
   FixedEventHeader fixed_header;
   OptionalHeaderFields variable_header;
};

struct StructuredEvent
{
   EventHeader header;
   FilterableEventBody filterable_data;
   any remainder_of_body;
};

typedef sequence<StructuredEvent> EventBatch;

interface QoSAdmin
{
   QoSProperties get_qos();
   void set_qos ( in QoSProperties qos)
      raises ( UnsupportedQoS );
   void validate_qos {in QoSProperties required_qos,
                      out PropertyRangeSeq available_qos )
      raises ( UnsupportedQos );
};

interface AdminPropertiesAdmin
{
   AdminProperties get_admin();
   void set_admin (in AdminProperties admin)
      raises ( UnsupportedAdmin);
};
```

Constants

```
// The following constant declarations define the standard
// QoS property names and the associated values each
// property can take on. The name/value pairs for each
```

```
// standard property are grouped, beginning with a string
// constant defined for the property name followed by the
// values the property can take on.
const String EventReliablilty = "EventReliability";
const short BestEffort = 0;
const short Persistent = 1;

const string ConnectionReliability =
"ConnectionReliability";
// Can take on the same values as EventReliability

const string Priority = "Priority";
const short Lowestpriority = -32767;
const short HighestPriority = 32767;
const short DefaultPriority = 0;

const string StartTime = "StartTime";
// StartTime takes a value of type TimeBase::UtcT when
// placed in an event header.
// StartTime can also be set to either TRUE or FALSE at
// the Proxy level, indicating
// whether or not the Proxy supports the setting of
// per-message stop times.

const string StopTime = "StopTime";
// StopTime takes a value of type TimeBase::UtcT when
// placed 49 in an event header.
// StopTime can also be set to either TRUE or FALSE at the
// Proxy level, indicating
// whether or not the Proxy supports the setting of
// per-message stop times.

const string Timeout = "Timeout";
// Timeout takes on a value of type TimeBase::TimeT

const string OrderPolicy = "OrderPolicy";
const short AnyOrder = 0;
const short FifoOrder = 1;
const short PriorityOrder = 2;
const short DeadlineOrder = 3;

const string DiscardPolicy = "DiscardPolicy";
// DiscardPolicy takes on the same values as OrderPolicy,
// plus
const short LifoOrder = 4;

const string MaximumBatchSize = "MaximumBatchSize";
// MaximumBatchSize takes on a value of type long

const string PacingInterval = "PacingInterval"
// PacingInterval takes on a value of type TimeBase::TimeT

const string MaxQueueLength = ″MaxQueueLength'';
// MaxQueueLength takes on a value of type long
```

```
const string MaxConsumers = ″MaxConsumers'';
// MaxConsumers takes on a value of type long

const string MaxSuppliers = ″MaxSuppliers'';
// MaxSuppliers takes on a value of type long
```

Exceptions

```
exception UnsupportedQoS { PropertyErrorSeq qos_err; };
```

Interfaces

CosNotification::StructuredEvent Data Structure
CosNotification::QoSAdmin Interface

StructuredEvent Data Structure

This data structure defines the fields that comprise a Structured Event.

File name

CosNotification.idl

IDL syntax

```
struct FixedEventHeader
{
   string domain_type;
   string event_type;
   string event_name;
};

struct EventHeader
{
   FixedEventHeader fixed_header;
   OptionalHeaderFields variable_header;
};

struct StructuredEvent
{
   EventHeader header;
   FilterableEventBody filterable_data;
   any remainder_of_body;
};
```

StructuredEvent::EventHeader Structure

This structure contains two fields, one of which is the FixedEventHeader struct and the other is the variable_header field.

Original interface

CosNotification::StructuredEvent

IDL syntax

```
struct EventHeader
{
   FixedEventHeader fixed_header;
   OptionalHeaderFields variable_header;
};
```

Parameters

FixedEventHeader

See“StructuredEvent::FixedEventHeader”.

variable_header

The variable_header field contains the rest of the Structured Event. The data type of this field is a sequence of name-value pairs, where each name is a string and each value is a CORBA::Any. The name-value pairs are shown in Table 3. The standard variable header fields defined here provide QoS related information about the current Structured Event that should override other QoS settings within the channel when objects within the channel process the current Structured Event.

Table 3. Standard optional header fields.

Header Field Name	Type of Associated Value
EventReliability	0
ConnectionReliability	0
Priority	short (can take values between -32,767 and 32,767)
Stop Time	TimeBase::UtcT (takes absolute time)
Timeout	TimeBase::TimeT (takes relative time)

StructuredEvent::FixedEventHeader

The FixedEventHeader structure contains three fields, the domain_type, event_type, and event_name.

Original interface

CosNotification::StructuredEvent

IDL syntax

```
struct FixedEventHeader
{
   string domain_type;
   string event_type;
   string event_name;
};
```

Parameters

domain_type
This field contains a string that identifies the vertical industry domain (e.g., telecommunications, healthcare, finance etc.) within which the type of event that characterizes a given Structured Event is defined.

event_type
The event_type field contains a string that identifies the type of event contained within a given StructuredEvent. This name should be unique among all event types defined within a given vertical domain, which is identified by the domain_type field.

event_name
The event_name field contains a string that names a specific instance of StructuredEvent. This name is not interpreted by any component of the Notification Service, and thus the semantics associated with it can be defined by end¡users of the service. This field can be used, for instance, to associate names with individual Structured Events that can be used to uniquely identify an instance of a particular type of Structured Event within a given installation of the Notification Service.

StructuredEvent::Body of a StructuredEvent

A StructuredEvent consists of two parts, the header part and the body part. The body of a Structured Event is intended to contain the contents of an instance of a Structured Event being published by a Notification Service supplier.

Original interface

CosNotification::StructuredEvent

IDL syntax

```
struct StructuredEvent
{
   EventHeader header;
   FilterableEventBody filterable_data;
   any remainder_of_body;
};
```

Parameters

filterable_data
The filterable_data portion of the body of a Structured Event is a sequence of name-value pairs, where name is of type string and the value is a CORBA::Any. The main purpose of this portion of the event body is to provide a convenient structure into which event body fields upon that filtering is likely to be performed can be placed.

remainder_of_body
The remainder_of_body portion of the event body is intended to hold event data upon which filtering is not likely to be performed.

QoSAdmin Interface

The QoSAdmin interface defines operations that enable clients to get and set the values of QoS properties. It also defines an operation that can verify whether or not a set of requested QoS property settings can be satisfied, along with returning information about the range of possible settings for additional QoS properties.

File name

CosNotification.idl

Intended usage

QoSAdmin is intended to be an abstract interface that is inherited by the Proxy, Admin, and Event Channel interfaces defined in the CosNotifyChannelAdmin module.

IDL syntax

```
interface QoSAdmin
{
   QoSProperties get_qos();
   void set_qos ( in QoSProperties qos)
      raises ( UnsupportedQoS );
   void validate_qos (in QoSProperties required_qos,
          out PropertyRangeSeq available_qos )
      raises ( UnsupportedQos );
};
```

Supported operations

QoSAdmin::get_qos
QoSAdmin::set_qos
QoSAdmin::validate_qos

QoSAdmin::get_qos

The get_qos operation takes no input parameters, and returns a sequence of name-value pairs that encapsulates the current quality of service settings for the target object (which could be an Event Channel, Admin, or Proxy object).

Original interface

CosNotification::QoSAdmin

IDL syntax

```
QoSProperties get_qos();
```

Parameters

None.

Return value

QoSProperties
A sequence of name-value pairs indicating the current quality of service (QoS) settings.

Exceptions

None.

Example

```
CosNotification::QoSProperties *qos;
CosNotifyChannelAdmin::EventChannel_var ec = NULL;

qos = ec->get_qos();                    // get qos
```

QoSAdmin::set_qos

The set_qos operation takes as an input parameter a sequence of name-value pairs that encapsulates quality of service property settings that a client is requesting the target object (which could be an Event Channel, Admin, or Proxy object) support as its default quality of service.

Original interface

CosNotification::QoSAdmin

IDL syntax

```
void set_qos ( in QoSProperties qos )
   raises ( UnsupportedQoS );
```

Parameters

qos Sequence of QoS name-value pairs that a client is requesting that the target object support as its default quality of service (QoS).

Remarks

If the implementation of the target object is not capable of supporting any of the requested quality of service settings, or if any of the requested settings would be in conflict with a QoS property defined at a higher level of the object hierarchy with respect to QoS, the UnsupportedQoS exception is raised. This exception contains as data a sequence of data structures, each of which identifies the name of a QoS property in the input list whose requested setting could not be satisfied and a range of settings for the property that could be satisfied.

Example

```
CosNotification::QoSProperties qos;
CORBA::Short priority;
time_t time_out;
CosNotifyChannelAdmin::EventChannel_var ec = NULL;

qos.length(2);
qos[0].name = CORBA::string_alloc(20);
strcpy(qos[0].name, "Priority");
priority = 6;
qos[0].value <<= priority;

qos[1].name = CORBA::string_alloc(20);
strcpy(qos[1].name, "Timeout");
time_out = 2 * 24 * 60 * 60 * 10000000; // two days in 100 nanoseconds
qos[1].value <<= time_out;
...                                     // Get Event Channel
ec->set_qos(qos);                       // set qos
```

QoSAdmin::validate_qos

The validate_qos operation accepts as input a sequence of QoS property name-value pairs that specify a set of QoS settings that a client would like to know if the target object is capable of supporting.

Original interface

CosNotification::QoSAdmin

IDL syntax

```
void validate_qos (in QoSProperties required_qos,
                   out PropertyRangeSeq available_qos )
   raises ( UnsupportedQos );
```

Parameters

required_qos
Sequence of QoS name-value pairs a client would like to know if the target object supports.

available_qos
Sequence of QoS name-value pairs the target object supports.

Return value

None.

Exceptions

UnsupportedQos
This exception is thrown when any one of the requested QoS settings could not be satisfied by the target object. Check to see if the name and the value of each of the QoS settings are correct.

Remarks

If all requested QoS property value settings could be satisfied by the target object, the operation returns successfully (without actually setting the QoS properties on the target object) with an output parameter that contains a sequence of PropertyRange data structures. Each element in this sequence includes the name of a an additional QoS property supported by the target object that could have been included on the input list and resulted in a successful return from the operation, along with the range of values that would have been acceptable for each such property.

Example

```
CosNotification::QoSProperties req_qos;
CosNotification::PropertyRangeSeq *avail_qos;
CosNotifyChannelAdmin::EventChannel_var ec = NULL;
CORBA::Short priority;

req_qos.length(1);
req_qos[0].name = CORBA::string_alloc(20);
strcpy(req_qos[0].name, "Priority");
priority = 6;
req_qos[0].value <<= priority;
...
ec->validate_qos(req_qos, avail_qos);
```

Chapter 9. CosNotifyChannelAdmin in the Notification Service

The other modules in the Notification Service are:
- CosNotification
- CosNotifyComm
- CosNotifyFilter
- INotifyChannelAdminManagedClient
- INotifyFilterManagedClient

CosNotifyChannelAdmin Module

The CosNotifyChannelAdmin module defines the interfaces necessary to create, configure, and administer instances of a Notification Service event channel.

Not supported by OS/390 Component Broker

File name

CosNotifyChannelAdmin.idl

Types

```
// Forward declarations
interface ConsumerAdmin;
interface SupplierAdmin;
interface EventChannel;
interface EventChannelFactory;

interface ProxySupplier :
CosNotification::QoSAdmin,
CosNotifyFilter::FilterAdmin
{
   readonly attribute ConsumerAdmin MyAdmin;
   attribute CosNotifyFilter::MappingFilter priority_filter;
   attribute CosNotifyFilter::MappingFilter lifetime_filter
   CosNotifyComm::EventTypeNameSeq
   obtain_offered_types();
   void validate_event_qos (in CosNotification::QoSProperties
      required_qos, out CosNotification::PropertyRangeSeq available_qos)
      raises (CosNotification::UnsupportedQoS);
};

interface ProxyPushSupplier :
ProxySupplier,
CosNotifyComm::NotifySubscribe,
CosEventComm::PushSupplier
{
   void connect_any_push_consumer (in CosEventComm::PushConsumer
```

```
         push_consumer)
      raises(CosEventChannelAdmin::AlreadyConnected,
             CosEventChannelAdmin::TypeError );
   void suspend_connection()
      raises(ConnectionAlreadyActive);
   void resume_connection()
      raises(ConnectionAlreadyActive);
};

interface StructuredProxyPushSupplier
{
ProxySupplier,
CosNotifyComm::StructuredPushSupplier
{
   void connect_structured_push_consumer (
         in CosNotifyComm::StructuredPushConsumer push_consumer)
      raises(CosEventChannelAdmin::AlreadyConnected,
             CosEventChannelAdmin::TypeError );
   void suspend_connection()
      raises(ConnectAlreadyInactive);
   void resume_connection()
      raises(ConnectionAlreadyActive);
};

interface SequenceProxyPushSupplier :
ProxySupplier,
CosNotifyComm::SequencePushSupplier
{
   void connect_sequence_push_consumer (
         in CosNotifyComm::SequencePushConsumer push_consumer)
      raises(CosEventChannelAdmin::AlreadyConnected,
             CosEventChannelAdmin::TypeError );
   void suspend_connection()
      raises(ConnectAlreadyInactive);
   void resume_connection()
      raises(ConnectionAlreadyActive);
};

interface ProxyConsumer :
CosNotification::QoSAdmin,
CosNotifyFilter::FilterAdmin
{
   readonly attribute SupplierAdmin MyAdmin;
   CosNotifyComm::EventTypeNameSeq
   obtain_subscription_types();
   void validate_event_qos (in CosNotification::QoSProperties
         required_qos, out CosNotification::PropertyRangeSeq
         available_qos)
      raises (CosNotification::UnsupportedQoS);
};

interface ProxyPushConsumer :
ProxyConsumer,
CosNotifyComm::NotifyPublish,
CosEventComm::PushConsumer
```

```
{
   void connect_any_push_supplier (in CosEventComm::PushSupplier
         push_supplier)
      raises(CosEventChannelAdmin::ALreadyConnected);
};

interface StructuredProxyPushConsumer :
ProxyConsumer,
CosNotifyComm::StructuredPushConsumer
{
  void connect_structured_push_supplier (
         in CosNotifyComm::StructuredPushSupplier push_supplier)
      raises(CosEventChannelAdmin::AlreadyConnected);
};

interface SequenceProxyPushConsumer :
ProxyConsumer,
CosNotifyComm::SequencePushConsumer
{
   void connect_sequence_push_supplier (
         in CosNotifyComm::SequencePushSupplier push_supplier)
      raises(CosEventChannelAdmin::AlreadyConnected);
};

interface ProxyPullSupplier :
ProxySupplier,
CosNotifyComm::NotufySubscribe,
CosEventComm::PullSupplier
{
   void connect_any_pull_consumer (in CosEventComm::PullConsumer
         pull_consumer)
      raises(CosEventChannelAdmin::AlreadyConnected);
};

interface StructuredProxyPullSupplier :
ProxySupplier,
CosNotifyComm::StructuredPullSupplier
{
   void connect_structured_pull_consumer (
         in CosNotifyComm::StructuredPullConsumer pull consumer)
      raises(CosEventChannelAdmin::AlreadyConnected);
};
interface SequenceProxyPullSupplier :
ProxySupplier,
CosNotifyComm::SequencePullSupplier
{
   void connect_sequence_pull_consumer (
         in CosNotifyComm::SequencePullConsumer pull_consumer)
      raises(CosEventChannelAdmin::AlreadyConnected);
};

interface ProxyPullConsumer :
ProxyConsumer,
CosNotifyComm::NotifyPublish,
CosEventComm::PullConsumer
```

```
{
   void connect_any_pull_supplier (in CosEventComm::PullSupplier
         pull_supplier)
      raises(CosEventChannelAdmin::AlreadyConnected,
            CosEventChannelAdmin::TypeError );
};

interface StructuredProxyPullConsumer :
ProxyConsumer,
CosNotifyComm::StructuredPullConsumer
{
   void connect_structured_pull_supplier (
         in CosNotifyComm::StructuredPullSupplier pull_supplier)
      raises(CosEventChannelAdmin::AlreadyConnected,
            CosEventChannelAdmin::TypeError );
};

interface SequenceProxyPullConsumer :
ProxyConsumer,
CosNotifyComm::SequencePullConsumer
{
   void connect_sequence_pull_supplier (
         in CosNotifyComm::SequencePullSupplier pull_supplier)
      raises(CosEventChannelAdmin::AlreadyConnected,
            CosEventChannelAdmin::TypeError );
};

typedef long ProxyID;
typedef sequence<ProxyID> ProxyIDSeq;

enum ClientType
{
   ANY_EVENT,
   STRUCTURED_EVENT,
   SEQUENCE_EVENT
};

typedef long AdminID;
typedef sequence<AdminID> AdminSeq;

interface SupplierAdmin :
CosNotification::QoSAdmin,
CosNotifyComm::NotifyPublish,
CosNotifyFilter::FilterAdmin,
CosEventChannelAdmin::SupplierAdmin
{
   readonly attribute AdminID MyID;
   readonly attribute EventChannel MyChannel;

   readonly attribute ProxyIDSeq pull_consumers;
   readonly attribute ProxyIDSeq push_consumers;
   ProxyConsumer get_proxy_consumer (in ProxyID proxy_id )
      raises (ProxyNotFound);
   ProxyConsumer obtain_notification_pull_consumer (
      in ClinetType ctype, out ProxyID proxy_id);
```

```
   ProxyConsumer obtain_notification_push_consumer (
      in ClientType ctype, out ProxyID proxy_id);
};

interface Consumer Admin :
CosNotification::QoSAdmin,
CosNotifyComm::NotifySubscribe,
CosNotifyFilter::FilterAdmin,
CosEventChannelAdmin::ConsumerAdmin
{
   readonly attribute AdminID MyID;
   readonly attribute EventChannel MyChannel;
      attribute CosNotifyFilter::MappingFilter
   priority_filter;
   attribute CosNotifyFilter::MappingFilter
   lifetime_filter;
   readonly attribute ProxyIDSeq pull_suppliers;
   readonly attribute ProxyIDSeq push_suppliers;

   ProxySupplier get_proxy_supplier (in ProxyID proxy_id )
      raises (ProxyNotFound);
   ProxySupplier obtain_notification_pull_supplier (
      in ClientType ctype, out ProxyID proxy_id);
   ProxySupplier obtain_notification_push_supplier (
      in ClinetType ctype, out proxyID proxy_id);
};

interface EventChannel :
CosNotification::QoSAdmin,
CosNotification::AdminPropertiesAdmin,
CosEventChannelAdmin::EventChannel
{
   readonly attribute EventChannelFactory MyFactory;
   readonly attribute ConsumerAdmin
   default_consumer_admin;
   readonly attribute SupplierAdmin
   default_supplier_admin;
   ConsumerAdmin new_for_consumers (out AdminID id);
   SupplierAdmin new_for_suppliers (out AdminID id);
   ConsumerAdmin get_consumeradmin (in AdminID id)
      raises (AdminNotFound);
   SupplierAdmin get_supplieradmin (in AdminID id)
      raises (AdminNotFound);
   AdminIDSeq get_all_consumeradmins();
   AdminIDSeq get_all_supplieradmins();
};

typedef long ChannelID;
typedef sequence<ChannelID> ChannelIDSeq;

 interface EventChannelFactory
{
   EventChannel create_channel (
         in CosNotification::QoSProperties initial_qos,
         in CosNotification::Adminproperies initial_admin,
```

```
            out ChannelID id)
         raises(CosNotification::UnsupportedQoS,
      CosNotification::UnsupportedAdmin );
      ChannelIDSeq get_all_channels();
      EventChannel get_event_channel (in ChannelID id)
         raises (ChannelNotFound);
};
```

Exceptions

```
exception ConnectionAlreadyActive {};
exception ConnectionAlreadyInactive {};

exception AdminNotFound {};
exception ProxyNotFound {};

exception ChannelNotFound ();
```

Intended usage

It defines the different types of proxy interfaces that support connections from the various types of clients that are supported, the Admin interfaces, the EventChannel interface, and a factory interface for instantiating new channels.

Interfaces

CosNotiyChannelAdmin::ConsumerAdmin Interface
CosNotifyChannelAdmin::EventChannel Interface
CosNotifyChannelAdmin::EventChannelFactory Interface
CosNotifyChannelAdmin::ProxyConsumer Interface
CosNotifyChannelAdmin::ProxySupplier Interface
CosNotifyChannelAdmin::StructuredProxyPullConsumer Interface
CosNotifyChannelAdmin::StructuredProxyPullSupplier Interface
CosNotifyChannelAdmin::StructuredProxyPushConsumer Interface
CosNotifyChannelAdmin::StructuredProxyPushSupplier Interface
CosNotifyChannelAdmin::SupplierAdmin Interface

ConsumerAdmin Interface

The ConsumerAdmin interface defines the behavior supported by objects that create and manage lists of proxy supplier objects within a Notification Service event channel.

File name

CosNotifyChannelAdmin.idl

Intended usage

A Notification Service event channel can have any number of ConsumerAdmin instances associated with it. Each such instance is responsible for creating and managing a list of proxy supplier objects that share a common set of QoS property

settings, and a common set of filter objects. This feature enables clients to conveniently group proxy supplier objects within a channel into groupings that each support a set of consumers with a common set of QoS requirements and event subscriptions.

The ConsumerAdmin interface inherits the QoSAdmin interface defined within the CosNotification module, enabling each ConsumerAdmin instance to manage a set of QoS property settings. These QoS property settings are assigned as the default QoS property settings for any proxy supplier object created by a ConsumerAdmin instance. In addition, the ConsumerAdmin interface inherits from the FilterAdmin interface defined within the CosNotifyFilter module, enabling each ConsumerAdmin instance to maintain a list of filter objects. These filter objects encapsulate subscriptions that will apply to all proxy supplier objects that have been created by a given ConsumerAdmin instance. In order to enable optimizing the notification of a group of proxy supplier objects that have been created by the same ConsumerAdmin instance of changes to these shared filter objects, the ConsumerAdmin interface also inherits from the NotifySubscribe interface defined in the CosNotifyComm module. This inheritance enables a ConsumerAdmin instance to be registered as the callback object for notification of subscription changes made upon filter objects.

Locally, the ConsumerAdmin interface supports a readonly attribute that maintains a reference to the EventChannel instance that created a given ConsumerAdmin instance. The ConsumerAdmin interface also supports a readonly attribute that contains a numeric identifier that will be assigned to an instance supporting this interface by its associated Notification Service event channel upon creation of the ConsumerAdmin instance. This identifier will be unique among all ConsumerAdmin instances created by a given channel.

Each ConsumerAdmin instance assigns a unique numeric identifier to each proxy supplier object it maintains. The ConsumerAdmin interface supports attributes that maintain the list of these unique identifiers associated with the proxy pull and the proxy push suppliers created by a given ConsumerAdmin instance. The ConsumerAdmin interface also supports an operation which, given the unique identifier of a proxy supplier a given ConsumerAdmin instance has created as input, will return the object reference of that proxy supplier object. Finally, the ConsumerAdmin interface supports operations that can create the various styles of proxy supplier objects supported by the Notification Service event channel.

IDL syntax

```
interface ConsumerAdmin :

CosNotification::QoSAdmin,
CosNotifyComm::NotifySubscribe,
CosNotifyFilter::FilterAdmin,
CosEventChannelAdmin::ConsumerAdmin
{
   readonly attribute AdminID MyID;
   readonly attribute EventChannel MyChannel;

   attribute CosNotifyFilter::MappingFilter
   priority_filter;
   attribute CosNotifyFilter::MappingFilter
```

```
    lifetime_filter;

    readonly attribute ProxyIDSeq pull_suppliers;
    readonly attribute ProxyIDSeq push_suppliers;

    ProxySupplier get_proxy_supplier (in ProxyID proxy_id )
       raises (ProxyNotFound);

    ProxySupplier obtain_notification_pull_supplier (
       in ClientType ctype, out ProxyID proxy_id);

    ProxySupplier obtain_notification_push_supplier (
       in ClinetType ctype, out proxyID proxy_id);
};
```

Supported operations

ConsumerAdmin::MyChannel Attribute
ConsumerAdmin::MyID Attribute
ConsumerAdmin::pull_suppliers Attribute
ConsumerAdmin::push_suppliers Attribute
ConsumerAdmin::get_proxy_supplier
ConsumerAdmin::obtain_notification_pull_supplier
ConsumerAdmin::obtain_notification_push_supplier

ConsumerAdmin::MyChannel Attribute

The MyChannel attribute is a readonly attribute that maintains the object reference of the Notification Service event channel that created a given SupplierAdmin instance.

Original interface

CosNotifyChannelAdmin::ConsumerAdmin

IDL syntax

```
readonly attribute EventChannel MyChannel;
```

ConsumerAdmin::MyID Attribute

The MyID attribute is a readonly attribute that maintains the unique identifier of the target SupplierAdmin instance that is assigned to it upon creation by the Notification Service event channel.

Original interface

CosNotifyChannelAdmin::ConsumerAdmin

IDL syntax

```
readonly attribute AdminID MyID;
```

ConsumerAdmin::pull_suppliers Attribute

The pull_suppliers attribute is a readonly attribute that contains the list of unique identifiers that have been assigned by a ConsumerAdmin instance to each pull-style proxy supplier object it has created.

Original interface

CosNotifyChannelAdmin::ConsumerAdmin

IDL syntax

```
readonly attribute ProxyIDSeq pull_suppliers;
```

ConsumerAdmin::push_suppliers Attribute

The push_suppliers attribute is a readonly attribute that contains the list of unique identifiers that have been assigned by a ConsumerAdmin instance to each push-style proxy supplier object it has created.

Original interface

CosNotifyChannelAdmin::ConsumerAdmin

IDL syntax

```
readonly attribute ProxyIDSeq push_suppliers;
```

ConsumerAdmin::get_proxy_supplier

The get_proxy_supplier operation accepts as an input parameter the numeric unique identifier associated with one of the proxy supplier objects that has been created by the target ConsumerAdmin instance.

Original interface

CosNotifyChannelAdmin::ConsumerAdmin

IDL syntax

```
ProxySupplier get_proxy_supplier (in ProxyID proxy_id )
   raises (ProxyNotFound);
```

Parameters

proxy_id
A numeric unique identifier associated with one of the proxy supplier objects that has been created by the target ConsumerAdmin instance.

Return value

ProxySupplier
A reference to the proxy supplier object whose unique identifier matches the proxy_id.

Exceptions

ProxyNotFound
This exception is raised if the input parameter (id) does not correspond to any of the proxy supplier objects created by the target ConsumerAdmin instance. Make sure that a proxy supplier object instance of that particular id value is created or use another (correct) id value.

ProxyNotFound

Remarks

If the input parameter does correspond to the unique identifier of a proxy supplier object that has been created by the target ConsumerAdmin instance, that proxy supplier object's reference is returned as the result of the operation. Otherwise, the ProxyNotFound exception is raised.

Example

```
CosNotifyChannelAdmin::ProxyID proxy_id;
CosNotifyChannelAdmin::ConsumerAdmin_var ca = NULL;
CosNotifyChannelAdmin::ProxySupplier_var pxyS;
...
pxyS = ca->get_proxy_supplier(proxy_id);
```

ConsumerAdmin::obtain_notification_pull_supplier

The obtain_notification_pull_supplier operation can create instances of the structured type of pull-style proxy supplier objects defined within the CosNotifyChannelAdmin module. Instances of the StructuredProxyPullSupplier interface support connections to pull consumers that receive events as Structured Events.

Original interface

CosNotifyChannelAdmin::ConsumerAdmin

IDL syntax

```
ProxySupplier obtain_notification_pull_supplier (
   in ClientType ctype, out ProxyID proxy_id);
```

Parameters

ctype A flag that indicates that the structured style of pull-style proxy supplier instance should be created.

proxy_id
An output parameter that is an unique identifier (among all proxy suppliers the ConsumerAdmin created) for the newly created StructuredProxyPullSupplier object.

Return value

ProxySupplier
A reference to the new StructuredProxyPullSupplier instance.

Exceptions

None.

Remarks

The obtain_notification_pull_supplier accepts as an input parameter a flag that indicates that the structured style of pull-style proxy supplier instance should be created. The target ConsumerAdmin creates the new StructuredProxyPullSupplier instance and assigns a numeric identifier to it that is unique among all proxy suppliers it has created. The unique identifier is returned as the output parameter of the operation, and the reference to the new proxy supplier instance is returned as the operation result.

Example

```
CosNotifyChannelAdmin::ProxyID proxy_id;
CosNotifyChannelAdmin::ConsumerAdmin_var ca = NULL;
CosNotifyChannelAdmin::StructuredProxyPullSupplier_var pxyPullS;
...
pxyPullS = CosNotifyChannelAdmin::StructuredProxyPullSupplier::_narrow(
                      ca->obtain_notification_pull_supplier(
                      CosNotifyChannelAdmin::STRUCTURED_EVENT, proxy_id));
```

ConsumerAdmin::obtain_notification_push_supplier

The obtain_notification_push_supplier operation can create instances of the structured type of push-style proxy supplier objects defined within the CosNotifyChannelAdmin module. Instances of the StructuredProxyPushSupplier interface support connections to push consumers that receive events as Structured Events.

Original interface

CosNotifyChannelAdmin::ConsumerAdmin

IDL syntax

```
ProxySupplier obtain_notification_push_supplier (
   in ClientType ctype, out proxyID proxy_id);
```

Parameters

ctype A flag that indicates that the structured style of pull-style proxy supplier instance should be created.

proxy_id
An output parameter that is an unique identifier (among all proxy suppliers the ConsumerAdmin created) for the newly created StructuredProxyPushSupplier object.

Return value

ProxySupplier
A reference to the new StructuredProxyPushSupplier instance.

Exceptions

None.

Remarks

The obtain_notification_push_supplier accepts as an input parameter a flag that indicates that a structured style of push-style proxy supplier instance should be created. The target ConsumerAdmin creates the new StructuredProxyPushSupplier instance and assigns a numeric identifier to it that is unique among all proxy suppliers it has created. The unique identifier is returned as the output parameter of the operation, and the reference to the new proxy supplier instance is returned as the operation result.

Example

```
CosNotifyChannelAdmin::ProxyID proxy_id;
CosNotifyChannelAdmin::ConsumerAdmin_var ca = NULL;
CosNotifyChannelAdmin::StructuredProxyPushSupplier_var pxyPushS;
...
pxyPushS = CosNotifyChannelAdmin::StructuredProxyPushSupplier::_narrow(
                      ca->obtain_notification_push_supplier(
                      CosNotifyChannelAdmin::STRUCTURED_EVENT, proxy_id));
```

EventChannel Interface

The EventChannel interface encapsulates the behaviors supported by a Notification Service event channel.

File name

CosNotifyChannelAdmin.idl

Intended usage

The EventChannel interface defined within the CosNotifyChannelAdmin module inherits from the QoSAdmin interface defined within the CosNotification module. Inheritance of this interface enables a Notification Service style event channel to manage lists of associated QoS properties.

Also, the EventChannel interface supports a readonly attribute that maintains a reference to the EventChannelFactory instance that created it. In addition, each instance of the EventChannel interface has an associated default ConsumerAdmin and an associated default SupplierAdmin instance, both of which exist upon creation of the channel and that have the unique identifier of zero (note that admin object identifiers only need to be unique among a given type of admin, implying that the identifiers assigned to ConsumerAdmin objects can overlap those assigned to SupplierAdmin objects). The EventChannel interface supports readonly attributes that maintain references to these default admin objects.

The EventChannel interface supports operations that create new ConsumerAdmin and SupplierAdmin instances. In addition, the EventChannel interface supports operations that can return references to the ConsumerAdmin and SupplierAdmin instances associated with a given EventChannel instance, given the unique identifier of an admin object as input. Finally, the EventChannel interface supports operations that return the sequence of unique identifiers of all ConsumerAdmin and SupplierAdmin instances associated with a given EventChannel instance.

IDL syntax

```
interface EventChannel :

CosNotification::QoSAdmin,
CosNotification::AdminPropertiesAdmin,
CosEventChannelAdmin::EventChannel
{
   readonly attribute EventChannelFactory MyFactory;
   readonly attribute ConsumerAdmin
   default_consumer_admin;
   readonly attribute SupplierAdmin
   default_supplier_admin;
   ConsumerAdmin new_for_consumers (out AdminID id);
   SupplierAdmin new_for_suppliers (out AdminID id);
   ConsumerAdmin get_consumeradmin (in AdminID id)
      raises (AdminNotFound);
   SupplierAdmin get_supplieradmin (in AdminID id)
      raises (AdminNotFound);
   AdminIDSeq get_all_consumeradmins();
   AdminIDSeq get_all_supplieradmins();
};
typedef long ChannelID;
typedef sequence<ChannelID> ChannelIDSeq;

exception ChannelNotFound ();

interface EventChannelFactory
```

```
{
   EventChannel create_channel (
         in CosNotification::QoSProperties initial_qos,
         in CosNotification::Adminproperies initial_admin,
         out ChannelID id)
      raises(CosNotification::UnsupportedQoS,
            CosNotification::UnsupportedAdmin);
   ChannelIDSeq get_all_channels();
   EventChannel get_event_channel (in ChannelID id)
      raises (ChannelNotFound);
};
```

Supported operations

EventChannel::default_consumer_admin Attribute
EventChannel::default_supplier_admin Attribute
EventChannel::MyFactory Attribute
EventChannel::get_all_consumeradmins
EventChannel::get_all_supplieradmins
EventChannel::get_consumeradmin
EventChannel::get_supplieradmin
EventChannel::new_for_consumers
EventChannel::new_for_suppliers

EventChannel::default_consumer_admin Attribute

The default_consumer_admin attribute is a readonly attribute that maintains a reference to the default ConsumerAdmin instance associated with the target EventChannel instance.

Original interface

CosNotifyChannelAdmin::EventChannel

IDL syntax

```
readonly attribute ConsumerAdmin default_consumer_admin;
```

Remarks

Each EventChannel instance has an associated default ConsumerAdmin instance, which exists upon creation of the channel and is assigned the unique identifier of zero. Subsequently, clients can create additional Event Service style ConsumerAdmin instances by invoking the inherited for_consumers operation, and additional Notification Service style ConsumerAdmin instances by invoking the new_for_consumers operation defined by the EventChannel interface.

EventChannel::default_supplier_admin Attribute

The default_supplier_admin attribute is a readonly attribute that maintains a reference to the default SupplierAdmin instance associated with the target EventChannel instance.

Original interface

CosNotifyChannelAdmin::EventChannel

IDL syntax

```
readonly attribute SupplierAdmin default_supplier_admin;
```

Remarks

Each EventChannel instance has an associated default SupplierAdmin instance, which exists upon creation of the channel and is assigned the unique identifier of zero. Subsequently, clients can create additional Event Service style SupplierAdmin instances by invoking the inherited for_suppliers operation, and additional Notification Service style SupplierAdmin instances by invoking the new_for_suppliers operation defined by the EventChannel interface.

EventChannel::get_all_consumeradmins

The get_all_consumeradmins operation takes no input parameters and returns a sequence of the unique identifiers assigned to all Notification Service style ConsumerAdmin instances that have been created by the target EventChannel instance.

Original interface

CosNotifyChannelAdmin::EventChannel

IDL syntax

```
AdminIDSeq get_all_consumeradmins();
```

Parameters

None.

Return value

AdminIDSeq
A sequence of the unique identifiers assigned to all Notification Service style ConsumerAdmin instances that have been created by the target EventChannel instance.

Exceptions

None.

Example

```
CosNotifyChannelAdmin::EventChannel_var ec = NULL;
CosNotifyChannelAdmin::AdminIDSeq* caid_seq;
...
caid_seq = ec->get_all_consumeradmins();
```

EventChannel::get_all_supplieradmins

The get_all_supplieradmins operation takes no input parameters and returns a sequence of the unique identifiers assigned to all Notification Service style SupplierAdmin instances that have been created by the target EventChannel instance.

Original interface

CosNotifyChannelAdmin::EventChannel

IDL syntax

```
AdminIDSeq get_all_supplieradmins();
```

Parameters

None.

Return value

AdminIDSeq
A sequence of the unique identifiers assigned to all Notification Service style SupplierAdmin instances that have been created by the target EventChannel instance.

Exceptions

None.

Example

```
CosNotifyChannelAdmin::EventChannel_var ec = NULL;
CosNotifyChannelAdmin::AdminIDSeq* said_seq;
...
said_seq = ec->get_all_supplieradmins();
```

EventChannel::get_consumeradmin

The get_consumeradmin operation returns a reference to one of the ConsumerAdmin instances associated with the target EventChannel instance.

Original interface

CosNotifyChannelAdmin::EventChannel

IDL syntax

```
ConsumerAdmin get_consumeradmin (in AdminID id)
   raises (AdminNotFound);
```

Parameters

id A numeric value that is intended to be the unique identifier of one of the ConsumerAdmin instances associated with the target EventChannel instance.

Return value

ConsumerAdmin
A reference to the ConsumerAdmin instance whose unique identifier matches the id value.

Exceptions

AdminNotFound
This exception is raised when the id value does not match any of the ConsumerAdmin instances associated with the EventChannel. Make sure that a ConsumerAdmin instance with that particular id is created or use another (correct) id value.

Remarks

The get_consumeradmin operation accepts as an input parameter a numeric value that is intended to be the unique identifier of one of the ConsumerAdmin instances associated with the target EventChannel instance. If this turns out to be the case, the object reference of the associated ConsumerAdmin instance is returned as the operation result. Otherwise, the AdminNotFound exception is raised.

Example

```
CosNotifyChannelAdmin::AdminID id;
CosNotifyChannelAdmin::EventChannel_var ec = NULL;
CosNotifyChannelAdmin::ConsumerAdmin_var ca = NULL;
...
ca = CosNotifyChannelAdmin::ConsumerAdmin::_narrow(
        ec->get_consumeradmin(id));
```

EventChannel::get_supplieradmin

The get_supplieradmin operation returns a reference to one of the SupplierAdmin instances associated with the target EventChannel instance.

Original interface

CosNotifyChannelAdmin::EventChannel

IDL syntax

```
SupplierAdmin get_supplieradmin (in AdminID id)
   raises (AdminNotFound);
```

Parameters

id A numeric value that is intended to be the unique identifier of one of the SupplierAdmin instances associated with the target EventChannel instance.

Return value

SupplierAdmin
A reference to the SupplierAdmin instance whose unique identifier matches the id value.

Exceptions

AdminNotFound
This exception is raised when the id value does not match any of the SupplierAdmin instances associated with the EventChannel. Make sure that a SupplierAdmin instance with that particular id is created or use another (correct) id value.

Remarks

The get_supplieradmin operation accepts as an input parameter a numeric value that is intended to be the unique identifier of one of the SupplierAdmin instances associated with the target EventChannel instance. If this turns out to be the case, the object reference of the associated SupplierAdmin instance is returned as the operation result. Otherwise, the AdminNotFound exception is raised.

Example

```
CosNotifyChannelAdmin::AdminID id;
CosNotifyChannelAdmin::EventChannel_var ec = NULL;
CosNotifyChannelAdmin::SupplierAdmin_var sa = NULL;
...
sa = CosNotifyChannelAdmin::SupplierAdmin::_narrow(
        ec->get_supplieradmin(id));
```

EventChannel::MyFactory Attribute

The MyFactory attribute is a readonly attribute that maintains the object reference of the event channel factory that created a given Notification Service EventChannel instance.

Original interface

CosNotifyChannelAdmin::EventChannel

IDL syntax

```
readonly attribute EventChannelFactory MyFactory;
```

EventChannel::new_for_consumers

The new_for_consumers operation is invoked to create a new Notification Service style ConsumerAdmin instance.

Original interface

CosNotifyChannelAdmin::EventChannel

IDL syntax

```
ConsumerAdmin new_for_consumers (out AdminID id);
```

Parameters

id An output parameter that is a numeric unique identifier assigned to the new ConsumerAdmin instance.

Return value

ConsumerAdmin
A reference to the new ConsumerAdmin instance that was created as the result of the operation.

Exceptions

None.

Remarks

The new_for_consumers operation is invoked to create a new Notification Service style ConsumerAdmin instance. This instance is assigned a unique identifier by the target EventChannel instance that is unique among all ConsumerAdmin instances currently associated with the channel. Upon completion, the operation returns the reference to the new ConsumerAdmin instance as the result of the operation, and the unique identifier assigned to the new ConsumerAdmin instance as the output parameter.

Example

```
CosNotifyChannelAdmin::AdminID id;
CosNotifyChannelAdmin::EventChannel_var ec = NULL;
CosNotifyChannelAdmin::ConsumerAdmin_var ca = NULL;
...
ca = ec->new_for_consumers(id);
```

EventChannel::new_for_suppliers

The new_for_suppliers operation is invoked to create a new Notification Service style SupplierAdmin instance.

Original interface

CosNotifyChannelAdmin::EventChannel

IDL syntax

```
SupplierAdmin new_for_suppliers (out AdminID id);
```

Parameters

id An output parameter that is a numeric unique identifier assigned to the new SupplierAdmin instance.

Return value

SupplierAdmin
A reference to the new SupplierAdmin instance that was created as the result of the operation.

Exceptions

None.

Remarks

The new_for_suppliers operation is invoked to create a new Notification Service style SupplierAdmin instance. This instance is assigned a unique identifier by the target EventChannel instance that is unique among all SupplierAdmin instances currently associated with the channel. Upon completion, the operation returns the reference to the new SupplierAdmin instance as the result of the operation, and the unique identifier assigned to the new SupplierAdmin instance as the output parameter.

Example

```
CosNotifyChannelAdmin::AdminID id;
CosNotifyChannelAdmin::EventChannel_var ec = NULL;
CosNotifyChannelAdmin::SupplierAdmin_var sa = NULL;
...
sa = ec->new_for_suppliers(id);
```

EventChannelFactory Interface

The EventChannelFactory interface defines operations for creating and managing new Notification Service style event channels.

File name

CosNotifyChannelAdmin.idl

Intended usage

The EventChannelFactory interface supports a routine that creates new instances of Notification Service event channels and assigns unique numeric identifiers to them. In addition, the EventChannelFactory interface supports a routine that can return the unique identifiers assigned to all event channels created by a given instance of EventChannelFactory, and another routine that, given the unique identifier of an event channel created by a target EventChannelFactory instance, returns the object reference of that event channel.

IDL syntax

```
interface EventChannelFactory
{
   EventChannel create_channel (
         in CosNotification::QoSProperties initial_qos,
         in CosNotification::Adminproperies initial_admin,
         out ChannelID id)
      raises(CosNotification::UnsupportedQoS,
             CosNotification::UnsupportedAdmin );
   ChannelIDSeq get_all_channels();
   EventChannel get_event_channel (in ChannelID id)
      raises (ChannelNotFound);
};
```

Supported operations

EventChannelFactory::create_channel
EventChannelFactory::get_all_channels
EventChannelFactory::get_event_channel

EventChannelFactory::create_channel

The create_channel operation is invoked to create a new instance of the Notification Service style event channel.

Original interface

CosNotifyChannelAdmin::EventChannelFactory

IDL syntax

```
EventChannel create_channel (
      in CosNotification::QoSProperties initial_qos,
      in CosNotification::Adminproperies initial_admin,
      out ChannelID id)
   raises(CosNotification::UnsupportedQoS,
          CosNotification::UnsupportedAdmin );
```

Parameters

initial_qos
A list of name-value pairs that specify the initial QoS property settings for the new channel.

id An output identifier which is a unique numeric identifier that is assigned to the Notification Service style event channel created.

Return value

EventChannel
A reference to a new Notification Service style event channel created by this operation.

Exceptions

CosNotification::UnsupportedQoS
This exception is thrown when any one of the requested QoS settings could not be satisfied by the target object. Check to see if the name and the value of each of the QoS settings are correct.

Remarks

The create_channel operation is invoked to create a new instance of the Notification Service style event channel. This operation accepts two input parameters. The first input parameter is a list of name-value pairs that specify the initial QoS property settings for the new channel. The second input parameter is a list of name-value pairs that specify the initial administrative property settings for the new channel. If no implementation of the EventChannel interface exists that can support all of the requested QoS property settings, the UnsupportedQoS exception is raised. If this exception is not raised, the create_channel operation will return a reference to a new Notification Service style event channel. In addition, the operation assigns to this new

event channel a numeric identifier that is unique among all event channels created by the target object. This numeric identifier is returned as an output parameter.

Example

```
CosNotifyChannelAdmin::ChannelID cid;
CosNotification::QoSProperties qos;
CosNotifyChannelAdmin::EventChannelFactory_var ef = NULL;
CosNotifyChannelAdmin::EventChannel_var ec = NULL;
ec = ef->create_channel(qos, NULL, cid);
```

EventChannelFactory::get_all_channels

The get_all_channels operation returns a sequence of all of the unique numeric identifiers corresponding to Notification Service event channels that have been created by the target object.

Original interface

CosNotifyChannelAdmin::EventChannelFactory

IDL syntax

```
ChannelIDSeq get_all_channels();
```

Parameters

None.

Return value

ChannelIDSeq
A sequence of the unique identifiers assigned to all Notification Service event channel instances that have been created by the target object.

Exceptions

None.

Example

```
CosNotifyChannelAdmin::EventChannelFactory_var ef = NULL;
CosNotifyChannelAdmin::ChannelIDSeq* cid_seq;
...
cid_seq = ef->get_all_channels();
```

EventChannelFactory::get_event_channel

The get_event_channel operation accepts as input a numeric value that is supposed to be the unique identifier of a Notification Service event channel that has been created by the target object.

Original interface

CosNotifyChannelAdmin::EventChannelFactory

IDL syntax

```
EventChannel get_event_channel (in ChannelID id)
   raises (ChannelNotFound);
```

Parameters

id A numeric value that is supposed to be the unique identifier of a Notification Service event channel that has been created by the target object.

Return value

EventChannel
The object reference of the Notification Service event channel corresponding to the input identifier.

Exceptions

ChannelNotFound
This exception is raised when the id value does not match any of the Notification Service event channel instances associated with the EventChannel. Make sure that the Notification Service event channel instance with that particular id is created or use another (correct) id value.

Remarks

The get_event_channel operation accepts as input a numeric value that is supposed to be the unique identifier of a Notification Service event channel that has been created by the target object. If this input value does not correspond to such a unique identifier, the ChannelNotFound exception is raised. Otherwise, the operation returns the object reference of the Notification Service event channel corresponding to the input identifier.

Example

```
CosNotifyChannelAdmin::ChannelID cid;
CosNotifyChannelAdmin::EventChannelFactory_var ef = NULL;
CosNotifyChannelAdmin::EventChannel_var ec = NULL;
...
ec = ef->get_event_channel(cid);
```

ProxyConsumer Interface

The ProxyConsumer interface is intended to be an abstract interface that is inherited by the different varieties of proxy consumers that can be instantiated within a notification channel.

File name

CosNotifyChannelAdmin.idl

Intended usage

The ProxyConsumer interface encapsulates the behaviors common to all Notification Service proxy consumers. In particular, the ProxyConsumer interface inherits the QoSAdmin interface defined within the CosNotification module, and the FilterAdmin interface defined within the CosNotifyFilter module. The former inheritance enables all proxy consumers to administer a list of associated QoS properties, while the latter inheritance enables all proxy consumers to administer a list of associated filter objects. Locally, the ProxyConsumer interface defines a readonly attribute that contains a reference to the SupplierAdmin object that created it.

IDL syntax

```
interface ProxyConsumer :
CosNotification::QoSAdmin,
CosNotifyFilter::FilterAdmin
{
   readonly attribute SupplierAdmin MyAdmin;
   CosNotifyComm::EventTypeNameSeq
   obtain_subscription_types();
   void validate_event_qos (
         in CosNotification::QoSProperties required_qos,
         out CosNotification::PropertyRangeSeq available_qos)
      raises (CosNotification::UnsupportedQoS);
};
```

Supported operations

ProxyConsumer::validate_event_qos

ProxyConsumer::validate_event_qos

The validate_event_qos operation accepts as input a sequence of QoS property name-value pairs which specify a set of QoS settings that a client is interested in setting on a per-event basis.

Original interface

CosNotifyChannelAdmin::ProxyConsumer

IDL syntax

```
void validate_event_qos (
      in CosNotification::QoSProperties required_qos,
      out CosNotification::PropertyRangeSeq available_qos)
   raises (CosNotification::UnsupportedQoS);
```

Parameters

required_qos
A sequence of QoS property name-value pairs which specify a set of QoS settings that a client is interested in setting on a per-event basis.

available_qos
An output parameter that is a sequence of PropertyRange data structures. Each element in this sequence has a name-value pair of QoS settings that the target object supports.

Return value

None.

Exceptions

CosNotification::UnsupportedQoS
This exception is thrown when any one of the requested QoS settings could not be satisfied by the target object. Check to see if the name and the value of each of the QoS settings are correct.

Remarks

The validate_event_qos operation accepts as input a sequence of QoS property name-value pairs which specify a set of QoS settings that a client is interested in setting on a per-event basis. Note that the QoS property settings contained in the optional header fields of a Structured Event may differ from those that are configured on a given proxy object. This operation is essentially a check to see if the target proxy object will honor the setting of a set of QoS properties on a per-event basis to values that may conflict with those set on the proxy itself. If any of the requested settings would not be honored by the target object on a per-event basis, the operation raises the UnsupportedQoS exception.

If all requested QoS property value settings could be satisfied by the target object, the operation returns successfully with an output parameter that contains a sequence of PropertyRange data structures. Each element in this sequence includes the name of a an additional QoS property whose setting is supported by the target object on a per-event basis and which could have been included on the input list while still resulting in a successful return from the operation. Each element also includes the range of values that would have been acceptable for each such property.

Example

```
CosNotification::QoSProperties *req_qos;
CosNotification::PropertyRangeSeq *avail_qos;
CosNotifyChannelAdmin::StructuredProxyPushConsumer_var pxyPushC;
...
pxyPushC->validate_event_qos(*req_qos, avail_qos);
```

ProxySupplier Interface

The ProxySupplier interface is intended to be an abstract interface that is inherited by the different varieties of proxy suppliers that can be instantiated within a notification channel.

File name

CosNotifyChannelAdmin.idl

Intended usage

The ProxySupplier interface encapsulates the behaviors common to all Notification Service proxy suppliers. In particular, the ProxySupplier interface inherits the QoSAdmin interface defined within the CosNotification module, and the FilterAdmin interface defined within the CosNotifyFilter module. The former inheritance enables all proxy suppliers to administer a list of associated QoS properties, while the latter inheritance enables all proxy suppliers to administer a list of associated filter objects. Locally, the ProxySupplier interface defines a readonly attribute that contains a reference to the ConsumerAdmin object that created it.

IDL syntax

```
interface ProxySupplier :

CosNotification::QoSAdmin,
CosNotifyFilter::FilterAdmin
{
   readonly attribute ConsumerAdmin MyAdmin;
   attribute CosNotifyFilter::MappingFilter priority_filter;
   attribute CosNotifyFilter::MappingFIlter lifetime_filter;
   CosNotifyComm::EventTypeNameSeq
   obtain_offered_types();
   void validate_event_qos (
         in CosNotification::QoSProperties required_qos,
         out CosNotification::PropertyRangeSeq available_qos)
      raises (CosNotification::UnsupportedQoS);
};
```

Supported operations

ProxySupplier::validate_event_qos

ProxySupplier::validate_event_qos

The validate_event_qos operation accepts as input a sequence of QoS property name-value pairs which specify a set of QoS settings that a client is interested in setting on a per-event basis.

Original interface

CosNotifyChannelAdmin::ProxySupplier

IDL syntax

```
void validate_event_qos (
      in CosNotification::QoSProperties required_qos,
      out CosNotification::PropertyRangeSeq available_qos)
   raises (CosNotification::UnsupportedQoS);
```

Parameters

required_qos
A sequence of QoS property name-value pairs which specify a set of QoS settings that a client is interested in setting on a per-event basis.

available_qos
An output parameter that is a sequence of PropertyRange data structures. Each element in this sequence has a name-value pair of QoS settings that the target object supports.

Return value

None.

Exceptions

UnsupportedQoS
This exception is thrown when any one of the requested QoS settings could not be satisfied by the target object. Check to see if the name and the value of each of the QoS settings are correct.

Remarks

The validate_event_qos operation accepts as input a sequence of QoS property name-value pairs that specify a set of QoS settings that a client is interested in setting on a per-event basis. Note that the QoS property settings contained in the optional header fields of a Structured Event may differ from those that are configured on a given proxy object. This operation is essentially a check to see if the target proxy object will honor the setting of a set of QoS properties on a per-event basis to values that may conflict with those set on the proxy itself. If any of the requested settings would not be honored by the target object on a per-event basis, the operation raises the UnsupportedQoS exception.

If all requested QoS property value settings could be satisfied by the target object, the operation returns successfully with an output parameter that contains a sequence of PropertyRange data structures. Each element in this sequence includes the name of an additional QoS property whose setting is supported by the target object on a per-event basis and which could have been included on the input list while still resulting in a successful return from the operation. Each element also includes the range of values that would have been acceptable for each such property.

Example

```
CosNotification::QoSProperties *req_qos;
CosNotification::PropertyRangeSeq *avail_qos;
CosNotifyChannelAdmin::StructuredProxyPushSupplier_var pxyPushS;
...
pxyPushS->validate_event_qos(*req_qos, avail_qos);
```

StructuredProxyPullConsumer Interface

The StructuredProxyPullConsumer interface supports connections to the channel by suppliers who will make events available for pulling to the channel as Structured Events.

File name

CosNotifyChannelAdmin.idl

Intended usage

The StructuredProxyPullConsumer interface also inherits from the StructuredPullConsumer interface defined in the CosNotifyComm module. This interface supports the operation that can be invoked to close down the connection from the supplier to the StructuredProxyPullConsumer. Also, the StructuredProxyPullConsumer interface defines a method that can be invoked by a pull-style supplier of Structured Events in order to establish a connection between the supplier and a notification channel over which the supplier will proceed to send events.

IDL syntax

```
interface StructuredProxyPullConsumer :
ProxyConsumer,
CosNotifyComm::StructuredPullConsumer
{
   void connect_structured_pull_supplier (
         in CosNotifyComm::StructuredPullSupplier pull_supplier)
      raises(CosEventChannelAdmin::AlreadyConnected,
             CosEventChannelAdmin::TypeError );
};
```

Supported operations

StructuredProxyPullConsumer::connect_structured_pull_supplier

StructuredProxyPullConsumer::connect_structured_pull_supplier

The connect_structured_pull_supplier operation accepts as an input parameter the reference to an object supporting the StructuredPullSupplier interface defined within the CosNotifyComm module.

Original interface

CosNotifyChannelAdmin::StructuredProxyPullConsumer

IDL syntax

```
void connect_structured_pull_supplier (
      in CosNotifyComm::StructuredPullSupplier pull_supplier)
   raises(CosEventChannelAdmin::AlreadyConnected,
          CosEventChannelAdmin::TypeError );
```

Parameters

pull_supplier
A reference to an object supporting the CosNotifyComm::StructuredPullSupplier interface. This reference is that of a supplier that plans to make events available for pulling to the channel with which the target object is associated in the form of Structured Events.

Return value

None.

Exceptions

CosEventChannelAdmin::AlreadyConnected
This exception is raised if the target object of this operation is already connected to a pull supplier object. This exception can be ignored since the requested connection has already been made.

Remarks

The connect_structured_pull_supplier operation accepts as an input parameter the reference to an object supporting the StructuredPullSupplier interface defined within the CosNotifyComm module. This reference is that of a supplier that plans to make events available for pulling to the channel with which the target object is associated in the form of Structured Events. This operation is thus invoked in order to establish a connection between a pull-style supplier of events in the form of Structured Events, and the notification channel. Once established, the channel can proceed to receive events from the supplier by invoking the pull_structured_event operation supported by the supplier. If the target object of this operation is already connected to a pull supplier object, the AlreadyConnected exception will be raised.

Example

```
CosNotifyChannelAdmin::StructuredProxyPullConsumer_var pxyPullC;
...
pxyPullC->connect_structured_pull_supplier(NULL);
```

StructuredProxyPullSupplier Interface

The StructuredProxyPullSupplier interface supports connections to the channel by consumers who will pull events from the channel as Structured Events.

File name

CosNotifyChannelAdmin.idl

Intended usage

The StructuredProxyPullSupplier interface supports connections to the channel by consumers who will pull events from the channel as Structured Events. Through inheritance of the ProxySupplier interface, the StructuredProxyPullSupplier interface supports administration of various QoS properties, administration of a list of associated filter objects, and a readonly attribute containing the reference of the ConsumerAdmin object that created it.

The StructuredProxyPullSupplier interface also inherits from the StructuredPullSupplier interface defined in the CosNotifyComm module. This interface supports the operations that enable a consumer of Structured Events to pull them from the StructuredProxyPullSupplier, and also the operation that can be invoked to close down the connection from the consumer to the StructuredProxyPullSupplier.

Also, the StructuredProxyPullSupplier interface defines a method that can be invoked by a pull-style consumer of Structured Events in order to establish a connection between the consumer and a notification channel over which the consumer will proceed to receive events.

IDL syntax

```
interface StructuredProxyPullSupplier :
ProxySupplier,
CosNotifyComm::StructuredPullSupplier
{
   void connect_structured_pull_consumer (
         in CosNotifyComm::StructuredPullConsumer pull_consumer)
      raises(CosEventChannelAdmin::AlreadyConnected);
};
```

Supported operations

StructuredProxyPullSupplier::connect_structured_pull_consumer

StructuredProxyPullSupplier::connect_structured_pull_consumer

The connect_structured_pull_consumer operation accepts as an input parameter the reference to an object supporting the StructuredPullConsumer interface defined within the CosNotifyComm module.

Original interface

CosNotifyChannelAdmin::StructuredProxyPullConsumer

IDL syntax

```
void connect_structured_pull_consumer (
      in CosNotifyComm::StructuredPullConsumer pull_consumer)
   raises(CosEventChannelAdmin::AlreadyConnected);
```

Parameters

pull_consumer
A reference to an object supporting the CosNotifyComm::StructuredPullConsumer interface. This reference is that of a consumer that plans to pull events from the channel to which the target object is associated in the form of Structured Events. This value can be null as described above.

Return value

None.

Exceptions

CosEventChannelAdmin::AlreadyConnected
This exception is raised if the target object of this operation is already connected to a pull consumer object. This exception can be ignored since the requested connection has already been made.

Remarks

The connect_structured_pull_consumer operation accepts as an input parameter the reference to an object supporting the StructuredPullConsumer interface defined within the CosNotifyComm module. This reference is that of a consumer that plans to pull events from the channel to which the target object is associated in the form of Structured Events. This operation is thus invoked in order to establish a connection between a pull-style consumer of events in the form of Structured Events, and the notification channel. Once established, the consumer can proceed to receive events from the channel by invoking the pull_structured_event or try_pull_structured_event operations supported by the target StructuredProxyPullSupplier instance. If the target object of this operation is already connected to a pull consumer object, the AlreadyConnected exception will be raised.

Also, note that the input parameter value (pull_consumer) can be null. In this case the Notification Channel will not be able to communicate back with the pull_consumer, i.e., the channel will not be able to call the disconnect_structured_pull_consumer method on the pull_consumer.

Example

```
CosNotifyChannelAdmin::StructuredProxyPullSupplier_var pxyPullS;
...
pxyPullS->connect_structured_pull_consumer(NULL);
```

StructuredProxyPushConsumer Interface

The StructuredProxyPushConsumer interface supports connections to the channel by suppliers who will push events to the channel as Structured Events.

File name

CosNotifyChannelAdmin.idl

Intended usage

The StructuredProxyPushConsumer interface also inherits from the StructuredPushConsumer interface defined in the CosNotifyComm module. This interface supports the operation that enables a supplier of Structured Events to push them to the StructuredProxyPushConumer, and also the operation that can be invoked to close down the connection from the supplier to the StructuredProxyPushConsumer.

Also, the StructuredProxyPushConsumer interface defines a method that can be invoked by a push-style supplier of Structured Events in order to establish a connection between the supplier and a notification channel over which the supplier will proceed to send events.

IDL syntax

```
interface StructuredProxyPushConsumer :
ProxyConsumer,
CosNotifyComm::StructuredPushConsumer
{
   void connect_structured_push_supplier (
         in CosNotifyComm::StructuredPushSupplier push_supplier)
      raises(CosEventChannelAdmin::AlreadyConnected);
};
```

Supported operations

StructuredProxyPushConsumer::connect_structured_push_supplier

StructuredProxyPushConsumer::connect_structured_push_supplier

The connect_structured_push_supplier operation accepts as an input parameter the reference to an object supporting the StructuredPushSupplier interface defined within the CosNotifyComm module.

Original interface

CosNotifyChannelAdmin::StructuredProxyPushConsumer

IDL syntax

```
void connect_structured_push_supplier (
      in CosNotifyComm::StructuredPushSupplier push_supplier)
   raises(CosEventChannelAdmin::AlreadyConnected);
```

Parameters

push_supplier
A reference to an object supporting the CosNotifyComm::StructuredPushSupplier interface. This reference is that of a supplier that plans to push events to the channel with which the target object is associated in the form of Structured Events. This value can be null.

Return value

None.

Exceptions

CosEventChannelAdmin::AlreadyConnected
This exception is raised if the target object of this operation is already connected to a push supplier object. This exception can be ignored since the requested connection has already been made.

Remarks

The connect_structured_push_supplier operation accepts as an input parameter the reference to an object supporting the StructuredPushSupplier interface defined within the CosNotifyComm module. This reference is that of a supplier that plans to push events to the channel with which the target object is associated in the form of Structured Events. This operation is thus invoked in order to establish a connection between a push-style supplier of events in the form of Structured Events, and the notification channel. Once established, the supplier can proceed to send events to the channel by invoking the push_structured_event operation supported by the target StructuredProxyPushConsumer instance. If the target object of this operation is already connected to a push supplier object, the AlreadyConnected exception will be raised.

Also, note that the input parameter value (push_supplier) can be null. In this case the Notification Channel will not be able to communicate back with the push_supplier, i.e., the channel will not be able to call the disconnect_structured_push_supplier method on the push_supplier.

Example

```
CosNotifyChannelAdmin::StructuredProxyPushConsumer_var pxyPushC;
...
pxyPushC->connect_structured_push_supplier(NULL);
```

StructuredProxyPushSupplier Interface

The StructuredProxyPushSupplier interface supports connections to the channel by consumers who will receive events from the channel as Structured Events.

File name

CosNotifyChannelAdmin.idl

Intended usage

Through inheritance of the ProxySupplier interface, the StructuredProxyPushSupplier interface supports administration of various QoS properties, administration of a list of associated filter objects, and a readonly attribute containing the reference of the ConsumerAdmin object that created it.

The StructuredProxyPushSupplier interface also inherits from the StructuredPushSupplier interface defined in the CosNotifyComm module. This interface supports the operation that can be invoked to close down the connection from the consumer to the StructuredProxyPushSupplier.

Also, the StructuredProxyPushSupplier interface defines the operation that can be invoked by a push consumer to establish the connection over which the push consumer will receive events from the channel. The StructuredProxyPushSupplier interface also defines a pair of operations that can suspend and resume the connection between a StructuredProxyPushSupplier instance and its associated StructuredPushConsumer. During the time such a connection is suspended, the StructuredProxyPushSupplier will accumulate events destined for the consumer but not transmit them until the connection is resumed.

IDL syntax

```
interface StructuredProxyPushSupplier :
ProxySupplier,
CosNotifyComm::StructuredPushSupplier
{
   void connect_structured_push_consumer (
         in CosNotifyComm::StructuredPushConsumer push_consumer)
      raises(CosEventChannelAdmin::AlreadyConnected,
```

```
            CosEventChannelAdmin::TypeError );
   void suspend_connection()
      raises(ConnectAlreadyInactive);
   void resume_connection()
   raises(ConnectionAlreadyActive);
};
```

Supported operations

StructuredProxyPushSupplier::connect_structured_push_consumer
StructuredProxyPushSupplier::resume_connection
StructuredProxyPushSupplier::suspend_connection

StructuredProxyPushSupplier::connect_structured_push_consumer

The connect_structured_push_consumer operation accepts as an input parameter the reference to an object supporting the StructuredPushConsumer interface defined within the CosNotifyComm module.

Original interface

CosNotifyChannelAdmin::StructuredProxyPushSupplier

IDL syntax

```
void connect_structured_push_consumer (
      in CosNotifyComm::StructuredPushConsumer push_consumer)
   raises(CosEventChannelAdmin::AlreadyConnected,
          CosEventChannelAdmin::TypeError );
```

Parameters

push_consumer
A reference to an object supporting the CosNotifyComm::StructuredPushConsumer interface. This reference is that of a consumer that will receive events from the channel with which the target object is associated in the form of Structured Events.

Return value

None.

Exceptions

CosEventChannelAdmin::AlreadyConnected
This exception is raised if the target object of this operation is already connected to a push consumer object. This exception can be ignored since the requested connection has already been made.

CosEventChannelAdmin::AlreadyConnected

Remarks

The connect_structured_push_consumer operation accepts as an input parameter the reference to an object supporting the StructuredPushConsumer interface defined within the CosNotifyComm module. This reference is that of a consumer that will receive events from the channel with which the target object is associated in the form of Structured Events. This operation is thus invoked in order to establish a connection between a push-style consumer of events in the form of Structured Events, and the notification channel. Once established, the StructuredProxyPushSupplier will proceed to send events destined for the consumer to it by invoking its push_structured_event operation. If the target object of this operation is already connected to a push consumer object, the AlreadyConnected exception will be raised.

Example

```
CosNotifyChannelAdmin::StructuredProxyPushSupplier_var pxyPushS;
...
pxyPushS->connect_structured_push_consumer(NULL);
```

StructuredProxyPushSupplier::resume_connection

The resume_connection operation causes the target object supporting the StructuredProxyPushSupplier interface to resume sending events to the StructuredPushConsumer instance connected to it.

Original interface

CosNotifyChannelAdmin::StructuredProxyPushSupplier

IDL syntax

```
void resume_connection()
   raises(ConnectionAlreadyActive);
```

Parameters

None.

Return value

None.

Exceptions

ConnectionAlreadyActive

This exception is raised if the connection has not been previously suspended by using suspend_connection. This exception can be ignored since the requested connection is already active.

Remarks

The resume_connection operation operation takes no input parameters and returns no values. If the connection has not been previously suspended using this operation by invoking suspend_connection (described below), the ConnectionAlreadyActive exception is raised. Otherwise, the StructuredProxyPushSupplier will resume forwarding events to the StructuredPushConsumer connected to it, including those that have been queued during the time the connection was suspended, and have not yet timed out.

Example

```
CosNotifyChannelAdmin::StructuredProxyPushSupplier_var pxyPushS;
...
pxyPushS->resume_connection();
```

StructuredProxyPushSupplier::suspend_connection

The suspend_connection operation causes the target object supporting the StructuredProxyPushSupplier interface to stop sending events to the StructuredPushConsumer instance connected to it.

Original interface

CosNotifyChannelAdmin::StructuredProxyPushSupplier

IDL syntax

```
void suspend_connection()
   raises(ConnectAlreadyInactive);
```

Parameters

None.

Return value

None.

Exceptions

ConnectionAlreadyActive

This exception is raised if the connection has been previously suspended using this operation by invoking suspend_connection and not resumed by invoking resume_operation. This exception can be ignored since the requested connection is already active.

Remarks

The suspend_connection operation takes no input parameters and returns no values. If the connection has been previously suspended using this operation and not resumed

by invoking resume_connection (described above), the ConnectionAlreadyInactive exception is raised. Otherwise, the StructuredProxyPushSupplier will not forward events to the StructuredPushConsumer connected to it until resume_connection is subsequently invoked. During this time, the StructuredProxyPushSupplier will continue to queue events destined for the StructuredPushConsumer, although events that time out prior to resumption of the connection will be discarded. Upon resumption of the connection, all queued events will be forwarded to the StructuredPushConsumer.

Example

```
CosNotifyChannelAdmin::StructuredProxyPushSupplier_var pxyPushS;
...
pxyPushS->suspend_connection();
```

SupplierAdmin Interface

The SupplierAdmin interface defines the behavior supported by objects that create and manage lists of proxy consumer objects within a Notification Service event channel.

File name

CosNotifyChannelAdmin.idl

Intended usage

A Notification Service event channel can have any number of SupplierAdmin instances associated with it. Each such instance is responsible for creating and managing a list of proxy consumer objects that share a common set of QoS property settings, and a common set of filter objects. This feature enables clients to conveniently group proxy consumer objects within a channel into groupings that each support a set of suppliers with a common set of QoS requirements, and that make common event forwarding decisions driven by the association of a common set of filter objects.

The SupplierAdmin interface inherits the QoSAdmin interface defined within the CosNotification module, enabling each SupplierAdmin instance to manage a set of QoS property settings. These QoS property settings are assigned as the default QoS property settings for any proxy consumer object created by a SupplierAdmin instance. In addition, the SupplierAdmin interface inherits from the FilterAdmin interface defined within the CosNotifyFilter module, enabling each SupplierAdmin instance to maintain a list of filter objects. These filter objects encapsulate subscriptions that will apply to all proxy consumer objects that have been created by a given SupplierAdmin instance.

Locally, the SupplierAdmin interface supports a readonly attribute that maintains a reference to the EventChannel instance that created a given SupplierAdmin instance. The SupplierAdmin interface also supports a readonly attribute that contains a numeric identifier that will be assigned to an instance supporting this interface by its associated Notification Service event channel upon creation of the SupplierAdmin instance. This identifier will be unique among all SupplierAdmin instances created by a given channel.

Each SupplierAdmin instance assigns a unique numeric identifier to each proxy consumer object it maintains. The SupplierAdmin interface supports attributes that maintain the list of these unique identifiers associated with the proxy pull and the proxy push consumers created by a given SupplierAdmin instance. The SupplierAdmin interface also supports an operation which, given the unique identifier of a proxy consumer a given SupplierAdmin instance has created as input, will return the object reference of that proxy consumer object. Finally, the SupplierAdmin interface supports operations that can create the various styles of proxy consumer objects supported by the Notification Service event channel.

IDL syntax

```
interface SupplierAdmin :

CosNotification::QoSAdmin,
CosNotifyComm::NotifyPublish,
CosNotifyFilter::FilterAdmin,
CosEventChannelAdmin::SupplierAdmin
{
   readonly attribute AdminID MyID;
   readonly attribute EventChannel MyChannel;

   readonly attribute ProxyIDSeq pull_consumers;
   readonly attribute ProxyIDSeq push_consumers;
   ProxyConsumer get_proxy_consumer (in ProxyID proxy_id)
      raises (ProxyNotFound);
   ProxyConsumer obtain_notification_pull_consumer (
      in ClientType ctype, out ProxyID proxy_id);
   ProxyConsumer obtain_notification_push_consumer (
      in ClientType ctype, out ProxyID proxy_id);
};
```

Supported operations

SupplierAdmin::MyChannel Attribute
SupplierAdmin::MyID Attribute
SupplierAdmin::pull_consumers Attribute
SupplierAdmin::push_consumers Attribute
SupplierAdmin::get_proxy_consumer
SupplierAdmin::obtain_notification_pull_consumer
SupplierAdmin::obtain_notification_push_consumer

SupplierAdmin::get_proxy_consumer

The get_proxy_consumer operation accepts as an input parameter the numeric unique identifier associated with one of the proxy consumer objects that has been created by the target SupplierAdmin instance.

Original interface

CosNotifyChannelAdmin::SupplierAdmin

IDL syntax

```
ProxyConsumer get_proxy_consumer (in ProxyID proxy_id)
   raises (ProxyNotFound);
```

Parameters

proxy_id

A numeric unique identifier associated with one of the proxy consumer objects that has been created by the target SupplierAdmin instance.

Return value

ProxyConsumer

A reference to the proxy consumer object whose unique identifier matches the proxy_id.

Exceptions

ProxyNotFound

This exception is raised if the input parameter (id) does not correspond to any of the proxy consumer objects created by the target SupplierAdmin instance. Make sure that a proxy consumer object instance of that particular id value is created or use another (correct) value.

Remarks

The get_proxy_consumer operation accepts as an input parameter the numeric unique identifier associated with one of the proxy consumer objects that has been created by the target SupplierAdmin instance. If the input parameter does correspond to the unique identifier of a proxy consumer object that has been created by the target SupplierAdmin instance, that proxy consumer object's reference is returned as the result of the operation. Otherwise, the ProxyNotFound exception is raised.

Example

```
CosNotifyChannelAdmin::ProxyID proxy_id;
CosNotifyChannelAdmin::SupplierAdmin_var sa = NULL;
CosNotifyChannelAdmin::ProxyConsumer_var pxyC;
...
pxyC = sa->get_proxy_consumer(proxy_id);
```

SupplierAdmin::MyChannel Attribute

The MyChannel attribute is a readonly attribute that maintains the object reference of the Notification Service event channel that created a given SupplierAdmin instance.

Original interface

CosNotifyChannelAdmin::SupplierAdmin

IDL syntax

```
readonly attribute EventChannel MyChannel;
```

SupplierAdmin::MyID Attribute

The MyID attribute is a readonly attribute that maintains the unique identifier of the target SupplierAdmin instance that is assigned to it upon creation by the Notification Service event channel.

Original interface

CosNotifyChannelAdmin::SupplierAdmin

IDL syntax

```
readonly attribute AdminID MyID;
```

SupplierAdmin::obtain_notification_pull_consumer

The obtain_notification_pull_consumer operation can create instances of the structured type of pull-style proxy consumer objects defined within the CosNotifyChannelAdmin module.

Original interface

CosNotifyChannelAdmin::SupplierAdmin

IDL syntax

```
ProxyConsumer obtain_notification_pull_consumer (
   in ClientType ctype, out ProxyID proxy_id);
```

Parameters

ctype A flag that indicates that the structured style of pull-style proxy consumer instance should be created.

proxy_id
An output parameter that is an unique identifier (among all proxy consumers the SupplierAdmin created) for the newly created StructuredProxyPullConsumer object.

Return value

ProxyConsumer
A reference to the new StructuredProxyPushConsumer instance.

Exceptions

None.

Remarks

The obtain_notification_pull_consumer operation can create instances of the structured type of pull-style proxy consumer objects defined within the CosNotifyChannelAdmin module. Instances of the StructuredProxyPullConsumer interface support connections to pull suppliers that send events as Structured Events.

The obtain_notification_pull_consumer accepts as an input parameter a flag that indicates that the structured style of pull-style proxy consumer instance should be created. The target SupplierAdmin creates the new StructuredProxyPullConsumer instance and assigns a numeric identifier to it that is unique among all proxy consumers it has created. The unique identifier is returned as the output parameter of the operation, and the reference to the new proxy consumer instance is returned as the operation result.

Example

```
CosNotifyChannelAdmin::ProxyID proxy_id;
CosNotifyChannelAdmin::SupplierAdmin_var sa = NULL;
CosNotifyChannelAdmin::StructuredProxyPullConsumer_var pxyPullC;
...
pxyPullC = CosNotifyChannelAdmin::StructuredProxyPullConsumer::_narrow(
              sa->obtain_notification_pull_consumer(
                 CosNotifyChannelAdmin::STRUCTURED_EVENT, proxy_id));
```

SupplierAdmin::obtain_notification_push_consumer

The obtain_notification_push_consumer operation can create instances of the structuredtype of push-style proxy consumer objects defined within the CosNotifyChannelAdmin module.

Original interface

CosNotifyChannelAdmin::SupplierAdmin

IDL syntax

```
ProxyConsumer obtain_notification_push_consumer (
   in ClientType ctype, out ProxyID proxy_id);
```

Parameters

ctype A flag that indicates that the structured style of pull-style proxy consumer instance should be created.

proxy_id
An output parameter that is an unique identifier (among all proxy consumers the SupplierAdmin created) for the newly created StructuredProxyPullConsumer object.

Return value

ProxyConsumer
A reference to the new StructuredProxyPushConsumer instance.

Exceptions

None.

Remarks

The obtain_notification_push_consumer operation can create instances of the structured type of push-style proxy consumer objects defined within the CosNotifyChannelAdmin module. Instances of the StructuredProxyPushConsumer interface support connections to push suppliers that send events as Structured Events.

The obtain_notification_push_consumer accepts as an input parameter a flag that indicates that a structured style of push-style proxy consumer instance should be created. The target SupplierAdmin creates the new StructuredProxyPushConsumer instance and assigns a numeric identifier to it that is unique among all proxy consumers it has created. The unique identifier is returned as the output parameter of the operation, and the reference to the new proxy consumer instance is returned as the operation result.

Example

```
CosNotifyChannelAdmin::ProxyID proxy_id;
CosNotifyChannelAdmin::SupplierAdmin_var sa = NULL;
CosNotifyChannelAdmin::StructuredProxyPushConsumer_var pxyPushC;
...
pxyPushC = CosNotifyChannelAdmin::StructuredProxyPushConsumer::_narrow(
              sa->obtain_notification_push_consumer(
                 CosNotifyChannelAdmin::STRUCTURED_EVENT, proxy_id));
```

SupplierAdmin::pull_consumers Attribute

The pull_consumers attribute is a readonly attribute that contains the list of unique identifiers that have been assigned by a SupplierAdmin instance to each pull-style proxy consumer object it has created.

Original interface

CosNotifyChannelAdmin::SupplierAdmin

IDL syntax

```
readonly attribute ProxyIDSeq pull_consumers;
```

SupplierAdmin::push_consumers Attribute

The push_consumers attribute is a readonly attribute that contains the list of unique identifiers that have been assigned by a SupplierAdmin instance to each push-style proxy consumer object it has created.

Original interface

CosNotifyChannelAdmin::SupplierAdmin

IDL syntax

```
readonly attribute ProxyIDSeq push_consumers;
```

Chapter 10. CosNotifyComm in the Notification Service

The other modules in the Notification Service are:

- CosNotification
- CosNotifyChannelAdmin
- CosNotifyFilter
- INotifyChannelAdminManagedClient
- INotifyFilterManagedClient

CosNotifyComm Module

The CosNotifyComm module defines the interfaces that support Notification Service clients that communicate using Structured Events.

Not supported by OS/390 Component Broker

File name

CosNotifyComm.idl

Types

```
typedef string EventTypeName;
typedef sequence<EventTypeName> EventTypeNameSeq;

interface NotifyPublish
{
   void offer_change (
         in EventTypeNameSeq added, in EventTypedNameSeq removed )
      raises ( InvalidEventType );
};

interface NotifySubscribe
{
   void subscription_change (
         in EventTypeNameSeq added, in EventTypeNameSeq removed )
      raises ( InvalidEventType );
};

interface StructuredPushConsumer : NotifyPublish
{
   // NOTE: If transactional semantics are desired, a
   // transactional variant of this interface should be
   // implemented
   void push_structured_event (
         in CosNotification::StructuredEvent notification)
      raises (CosEventComm::Disconnected);
   void disconnect_structured_push_consumer();
```

```
};

interface StructuredPullConsumer : NotifyPublish
{
   void disconnect_structured_pull_consumer ();
};

interface StructuredPullSuppplier : NotifySubscribe
{
   // NOTE: If transactional semantics are desired, a transactional
   // variant of this interface should be implemented
   CosNotification::StructuredEvent
   pull_structured_event ()
      raises(CosEventComm::Disconnected);
   CosNotification::StructuredEvent
   try_pull_structured_event(out boolean has_event)
      raises(CosEventComm::Disconnected);
   void disconnect_structured_pull_supplier ();
};

interface StructuredPushSupplier : NotifySubscribe
{
   void disconnect_structured_push_supplier ();
};

interface SequencePushConsumer : NotifyPublish
{
   // NOTE: If transactional semantics are desired, a transactional
   // variant of this interface should be implemented
   void push_structured_events(
         in CosNotification::EventBatch notifications)
      raises(CoseventComm::Disconnected);
   void disconnect_sequence_push_consumer();
};

interface SequencePullConsumer : NotifyPublish
{
   void disconnect_sequence_pull_consumer();
};

interface SequencePullSupplier : NotifySubscribe
{
   // NOTE: If transactional semantics are desired, a transactional
   // variant of this interface should be implemented
   CosNotification::EventBatch pull_structured_events (in long max_number)
      raises(CosEventComm::Disconnected);
   CosNotification::StructuredEvent
   try_pull_structured_events(in long max_number, out boolean has_event)
      raises(CosEventComm::Disconnected);
   void disconnect_sequence_pull_supplier();
};
```

```
interface SequencePushSupplier : NotifySubscribe
{
   void disconnect_sequence_push_supplier();
};
```

Exceptions

```
exception InvalidEventType { EventTypeName type; };
```

Interfaces

CosNotifyComm::StructuredPullConsumer Interface
CosNotifyComm::StructuredPullSupplier Interface
CosNotifyComm::StructuredPushConsumer Interface
CosNotifyComm::StructuredPushSupplier Interface

StructuredPullConsumer Interface

The StructuredPullConsumer interface supports the behavior of objects that receive Structured Events using pull-style communication. It defines an operation that can be invoked to disconnect the pull consumer from its associated supplier.

File name

CosNotifyComm.idl

IDL syntax

```
interface StructuredPullConsumer : NotifyPublish
{
   void disconnect_structured_pull_consumer ();
};
```

Supported operations

StructuredPullConsumer::disconnect_structured_pull_consumer

StructuredPullConsumer::disconnect_structured_pull_consumer

The disconnect_structured_pull_consumer operation is invoked to terminate a connection between the target StructuredPullConsumer, and its associated supplier. This operation takes no input parameters and returns no values. The result of this operation is that the target StructuredPullConsumer will release all resources it had allocated to support the connection, and dispose its own object reference.

Original interface

CosNotifyComm::StructuredPullConsumer

IDL syntax

```
void disconnect_structured_pull_consumer ();
```

Parameters

None.

Return value

None.

Exceptions

None.

Example

```
CosNotifyChannelAdmin::StructuredProxyPullConsumer_var pxyPullC;
...
pxyPullC->disconnect_structured_pull_consumer();
```

StructuredPullSupplier Interface

The StructuredPullSupplier interface supports operations that enable suppliers to transmit Structured Events by the pull model. It also defines an operation that can be invoked to disconnect the pull supplier from its associated consumer.

Original interface

CosNotifyComm.idl

IDL syntax

```
interface StructuredPullSuppplier : NotifySubscribe
{
   CosNotification::StructuredEvent
   pull_structured_event ()
      raises(CosEventComm::Disconnected);
   CosNotification::StructuredEvent
   try_pull_structured_event (out boolean has_event)
      raises(CosEventComm::Disconnected);
   void disconnect_structured_pull_supplier ();
};
```

Supported operations

StructuredPullSupplier::disconnect_structured_pull_supplier
StructuredPullSupplier::pull_structured_event
StructuredPullSupplier::try_pull_structured_event

StructuredPullSupplier::disconnect_structured_pull_supplier

The disconnect_structured_pull_supplier operation is invoked to terminate a connection between the target StructuredPullSupplier, and its associated consumer. This operation takes no input parameters and returns no values. The result of this operation is that the target StructuredPullSupplier will release all resources it had allocated to support the connection, and dispose its own object reference.

Original interface

CosNotifyComm::StructuredPullSupplier

IDL syntax

```
void disconnect_structured_pull_supplier ();
```

Parameters

None.

Return value

None.

Exceptions

None.

Example

```
CosNotifyChannelAdmin::StructuredProxyPullSupplier_var pxyPullS;
...
pxyPullS->disconnect_structured_pull_supplier();
```

StructuredPullSupplier::pull_structured_event

The pull_structured_event operation takes no input parameters, and returns a value of type Structured Event as defined in the CosNotification module. Upon invocation, the operation will block until an event is available for transmission, at which time it will return an instance of a Structured Event that contains the event being delivered to its connected consumer. If invoked upon a StructuredPullSupplier that is not currently connected to the consumer of the event, the Disconnected exception will be raised.

Original interface

CosNotifyComm::StructuredPullSupplier

IDL syntax

```
pull_structured_event ()
   raises(CosEventComm::Disconnected);
```

Parameters

None.

Return value

CosNotification::StructuredEvent
Instance of a Structured Event that contains the event being delivered to its connected consumer.

Exceptions

CosEventComm::Disconnected
This exception is raised if the event communication has already been disconnected between the Pull Consumer and the StructuredPullSupplier. Reestablish the required connection.

Example

```
CosNotifyChannelAdmin::StructuredProxyPullSupplier_var pxyPullS;
CosNotification::StructuredEvent *ev_s;
...
ev_s = pxyPullS->pull_structured_event();
```

StructuredPullSupplier::try_pull_structured_event

The try_pull_structured_event operation takes no input parameters, and returns a value of type StructuredEvent as defined in the CosNotification module. It also returns an output parameter of type boolean that indicates whether or not the return value actually contains an event.

Original interface

CosNotifyComm::StructuredPullSupplier

IDL syntax

```
try_pull_structured_event (out boolean has_event)
   raises(CosEventComm::Disconnected);
```

Parameters

has_event
An output parameter of type boolean that indicates whether or not the return value actually contains an event. Set to TRUE if an event is available for delivery else set to FALSE.

Return value

CosNotification::StructuredEvent
An instance of a Structured Event that contains the event being delivered to its connected consumer.

Exceptions

CosEventComm::Disconnected
This exception is raised if the event communication has already been disconnected between the Pull Consumer and the StructuredPullSupplier. Reestablish the required connection.

CosEventComm::Disconnected

Remarks

The try_pull_structured_event operation takes no input parameters, and returns a value of type StructuredEvent as defined in the CosNotification module. It also returns an output parameter of type boolean that indicates whether or not the return value actually contains an event. Upon invocation, the operation will return an instance of a Structured Event that contains the event being delivered to its connected consumer, if such an event is available for delivery at the time the operation was invoked. If an event is available for delivery and thus returned as the result, the output parameter of the operation will be set to TRUE. If no event is available to return upon invocation, the operation will return immediately with the value of the output parameter set to FALSE. In this case, the return value will not contain a valid event. If invoked upon a StructuredPullSupplier that is not currently connected to the consumer of the event, the Disconnected exception will be raised.

Example

```
CORBA::Boolean has_event;
CosNotifyChannelAdmin::StructuredProxyPullSupplier_var pxyPullS;
CosNotification::StructuredEvent *ev_s;
...
ev_s = pxyPullS->try_pull_structured_event(has_event);
```

StructuredPushConsumer Interface

The StructuredPushConsumer interface supports an operation that enables consumers to receive Structured Events by the push model. It also defines an operation that can be invoked to disconnect the push consumer from its associated supplier.

File name

CosNotifyComm.idl

IDL syntax

```
interface StructuredPushConsumer : NotifyPublish
{
   void push_structured_event (
         in CosNotification::StructuredEvent notification)
      raises (CosEventComm::Disconnected);
   void disconnect_structured_push_consumer();
};
```

Supported operations

StructuredPushConsumer::disconnect_structured_push_consumer
StructuredPushConsumer::push_structured_event

StructuredPushConsumer::disconnect_structured_push_consumer

The disconnect_structured_push_consumer operation is invoked to terminate a connection between the target StructuredPushConsumer, and its associated supplier.

Original interface

CosNotifyComm::StructuredPushConsumer

IDL syntax

```
void disconnect_structured_push_consumer();
```

Parameters

None.

Return value

None.

Exceptions

None.

Remarks

This operation takes no input parameters and returns no values. The result of this operation is that the target StructuredPushConsumer will release all resources it had allocated to support the connection, and dispose its own object reference.

Example

```
CosNotifyChannelAdmin::StructuredProxyPushConsumer_var pxyPushC;
...
pxyPushC->disconnect_structured_push_consumer();
```

StructuredPushConsumer::push_structured_event

The push_structured_event operation takes as input a parameter of type StructuredEvent as defined in the CosNotification module. Upon invocation, this parameter will contain an instance of a Structured Event being delivered to the consumer by the supplier to which it is connected. If this operation is invoked upon a StructuredPushConsumer instance that is not currently connected to the supplier of the event, the Disconnected exception will be raised.

Original interface

CosNotifyComm::StructuredPushConsumer

IDL syntax

```
void push_structured_event (
      in CosNotification::StructuredEvent notification)
   raises (CosEventComm::Disconnected);
```

Parameters

notification

A parameter of type CosNotification::StructuredEvent that upon invocation, will contain an instance of a Structured Event being delivered to the consumer by the supplier to which it is connected.

Return value

None.

Exceptions

CosEventComm::Disconnected

This exception is raised if the event communication has already been disconnected between the Push Supplier and the StructuredPushConsumer. Reestablish the required connection.

CosEventComm::Disconnected

Example

```
CosNotification::StructuredEvent *ev_s =
   new CosNotification::StructuredEvent;
CosNotifyChannelAdmin::StructuredProxyPushConsumer_var pxyPushC;
...
pxyPushC->push_structured_event(*ev_s);
```

StructuredPushSupplier Interface

The StructuredPushSupplier interface supports the behavior of objects that transmit Structured Events using push-style communication. It defines an operation that can be invoked to disconnect the push supplier from its associated consumer.

File name

CosNotifyComm.idl

IDL syntax

```
interface StructuredPushSupplier : NotifySubscribe
{
   void disconnect_structured_push_supplier ();
};
```

Supported operations

StructuredPushSupplier::disconnect_structured_push_supplier

StructuredPushSupplier::disconnect_structured_push_supplier

The StructuredPushSupplier interface supports the behavior of objects that transmit Structured Events using push-style communication. It defines an operation that can be invoked to disconnect the push supplier from its associated consumer.

Original interface

CosNotifyComm::StructuredPushSupplier

IDL syntax

```
void disconnect_structured_push_supplier ();
```

Parameters

None.

Return value

None.

Exceptions

None.

Example

```
CosNotifyChannelAdmin::StructuredProxyPushSupplier_var pxyPushS;
...
pxyPushS->disconnect_structured_push_supplier();
```

Chapter 11. CosNotifyFilter in the Notification Service

The other modules in the Notification Service are:

- CosNotification
- CosNotifyChannelAdmin
- CosNotifyComm
- INotifyChannelAdminManagedClient
- INotifyFilterManagedClient

CosNotifyFilter Module

The CosNotifyFilter module defines the following interfaces: Filter, FilterFactory and FilterAdmin. The Filter interface encapsulates the constraints that will be used by a proxy object associated with a notification channel in order to make decisions to forward or discard events. The FilterFactory interface supports the operations required to create the filter objects, and the FilterAdmin interface supports operations that enable an interface (Proxy or Admin) that inherits it to manage a list of Filter instances.

Not supported by OS/390 Component Broker

File name

CosNotifyFilter.idl

Types

```
typedef unsigned long ConstraintID;

struct EventType
{
   string domain_name;
   string type_name;
};

typedef sequence<EventType> EventTypeSeq;

struct ConstraintExp
{
   EventTypeSeq event_types;
   string constraint_expr;
};

typedef sequence<ConstraintID> ConstraintIDSeq;
typedef sequence<ConstraintExp> ConstraintExpSeq;

struct ConstraintInfo
{
   ConstraintExp constraint_expression;
```

```
   ConstraintID constraint_id;
};

typedef sequence<ConstraintInfo> ConstraintInfoSeq;

struct MappingConstraintPair
{
   ConstraintExp constraint_expression;
   any result_to_set;
};
typedef sequence<MappingConstraintPair>
MappingConstraintPairSeq;

struct MappingConstraintInfo
{
   ConstraintExp constraint_expression;
   ConstraintID constraint_id;
   any value;
};
typedef sequence<MappingConstraintInfo>
MappingConstraintInfoSeq;
typedef long CallbackID;
typedef sequence<CallbackID> CallbackIDSeq;

interface Filter
{
   readonly attribute string constraint_grammar;

   ConstraintInfoSeq add_constraints (
         in ConstraintExpSeq constraint_list)
      raises (InvalidConstraint);
   void modify_constraints (
         in ConstraintIDSeq del_list, in ConstraintInfoSeq modify_list)
      raises(InvalidConstraint, ConstraintNotFound);
   ConstraintInfoSeq get_constraints (in ConstraintIDSeq id_list)
      raises(ConstraintNotFound);
   ConstraintInfoSeq get_all_constraints ();
   void remove_all_constraints();
   void destroy();
   boolean match (in any filterable_data)
      raises (UnsupportedFilterableData);
   boolean match_structured (
         in CosNotification::StructuredEvent filterable_data)
      raises (UnsupportedFilterableData);
   boolean match_typed (in CosTrading::PropertySeq filterable_data)
      raises (UnsupportedFilterableData);
   CallbackID attach_callback (
         in CosNotifyComm::NotifySubscribe callback);
   void detach_callback (in CallbackID callback)
      raises (CallbackNotFound);
   CallbackIDSeq get_callbacks ();
};

interface MappingFilter
{
```

```
   readonly attribute string constraint_grammar;
   readonly attribute CORBA::TypeCode value_type;
   readonly attribute any default_value;

   MappingConstraintInfoSeq add_mapping_constraints (
         in MappingConstraintPairSeq pair_list)
      raises(InvalidConstraint, InvalidValue);

   void modify_mapping_constraints (in ConstraintIDSeq del_list,
         in MappingConstraintInfoSeq modify_list)
      raises(InvalidConstraint, InvalidValue, ConstraintNotFound);

   MappingConstraintInfoSeq get_mapping_constraints (
         in ConstraintIDSeq id_list)
      raises(ConstraintNotFound);

   MappingConstraintInfoSeq get_all_mapping_constraints ();

   void remove_all_mapping_constraints();

   void destroy();

   boolean match (in any filterable_data, out any result_to_set)
      raises (UnsupportedFilterableData);

   boolean match_structured (
         in CosNotification::StructuredEvent filterable_data,
         out any result-to_set)
      raises (UnsupportedFilterableData);
   boolean match_typed (
         in CosTrading::PropertySeq filterable_data,
         out any result_to_set)
      raises (UnsupportedFilterableData);
};

interface FilterFactory
{
   Filter create_filter (in string constraint_grammar)
     raises(InvalidGrammar);
   MappingFilter create_mapping_filter (
         in string constraint_grammar, in any default_value)
      raises(InvalidGrammar);
};

typedef long FilterID;
typedef sequence<FilterID> FilterIDSeq;

interface FilterAdmin
{
   FilterID add_filter (in FIlter new_filter);
   void remove_filter (in FilterID filter)
      raises (FilterNotFound);
   Filter get_filter (in FilterID filter)
```

```
      raises (FilterNotFound);
   FilterIDSeq get_all_filters();
   void remove_all_filters();
};
```

Exceptions

```
exception UnsupportedFilterableData {};
exception InvalidGrammar {};
exception InvalidConstraint { ConstraintExp constr; };
exception DuplicateConstraintID { ConstraintID id; };

exception ConstraintNotFound { ConstraintID id; };
exception CallbackNotFound { };

exception InvalidValue {ConstraintExp constr, any value;};

exception FilterNotFound {};
```

Interfaces

CosNotifyFilter::Filter Interface
CosNotifyFilter::FilterAdmin Interface
CosNotifyFilter::FilterFactory Interface

Filter Interface

The Filter interface defines the behaviors supported by objects that encapsulate constraints used by the proxy objects associated with an event channel in order to determine which events they receive will be forwarded, and which will be discarded.

File name

CosNotifyFilter.idl

Intended usage

Each object supporting the Filter interface can encapsulate a sequence of any number of constraints. Each event received by a proxy object that has one or more objects supporting the Filter interface associated with it must satisfy at least one of the constraints associated with one of its associated Filter objects in order to be forwarded (either to another proxy object or to the consumer, depending on the type of proxy the filter is associated with), otherwise it will be discarded. Each constraint encapsulated by a filter object is a structure comprised of two main components. The first component is a sequence of data structures, each of which indicates an event type comprised of a domain and a type name. The second component is a boolean expression over the properties of an event, expressed in the constraint grammar described in “Appendix. Default Filter Constraint Language” on page 937. For a given constraint, the sequence of event type structures in the first component nominates a set of event types to which the constraint expression in the second component applies. Each element of the

sequence can contain strings that will be matched for equality against the domain_name and type_name fields of each event being evaluated by the filter object when determining if the boolean expression should be applied to the event, or the event should simply be discarded without even attempting to apply the boolean expression. The constraint expressions associated with a particular object supporting the Filter interface are expressed as strings that obey the syntax of a particular constraint grammar (i.e. a BNF). This implementation supports constraint expressions expressed in the default constraint grammar described in “Appendix. Default Filter Constraint Language” on page 937.

The Filter interface supports the operations required to manage the constraints associated with an object instance that supports the interface, along with a readonly attribute that identifies the particular constraint grammar in which the constraints encapsulated by this object have meaning. In addition, the Filter interface supports the match_structured operation that can be invoked by an associated proxy object upon receipt of an event, to determine if the event should be forwarded or discarded, based on whether or not the event satisfies at least one criteria encapsulated by the filter object.

IDL syntax

```
interface Filter
{
   readonly attribute string constraint_grammar;

   ConstraintInfoSeq add_constraints (in ConstraintExpSeq constraint_list)
      raises(InvalidConstraint);

   void modify_constraints (
         in ConstraintIDSeq del_list, in ConstraintInfoSeq modify_list)
      raises(InvalidConstraint, ConstraintNotFound);

   ConstraintInfoSeq get_constraints (in ConstraintIDSeq id_list)
      raises(ConstraintNotFound);

   ConstraintInfoSeq get_all_constraints ();

   void remove_all_constraints();

   void destroy();

   boolean match (in any filterable_data)
      raises (UnsupportedFilterableData);

   boolean match_structured (
         in CosNotification::StructuredEvent filterable_data)
      raises (UnsupportedFilterableData);

   boolean match_typed (in CosTrading::PropertySeq filterable_data)
      raises (UnsupportedFilterableData);

   CallbackID attach_callback (
         in CosNotifyComm::NotifySubscribe callback);
```

```
    void detach_callback (in CallbackID callback)
      raises (CallbackNotFound);

    CallbackIDSeq get_callbacks ();
  };
```

Supported operations

Filter::add_constraints
Filter::constraint_grammar
Filter::destroy
Filter::get_all_constraints
Filter::get_constraints
Filter::match_structured
Filter::modify_constraints
Filter::remove_all_constraints

Filter::add_constraints

The add_constraints operation is invoked by a client in order to associate one or more new constraints with the target filter object.

Original interface

CosNotifyFilter::Filter

IDL syntax

```
ConstraintInfoSeq add_constraints (in ConstraintExpSeq constraint_list)
  raises(InvalidConstraint);
```

Parameters

constraint_list
A sequence of data structures, each element of which contains of a sequence of event type structures and a constraint expressed within the constraint grammar specified in “Appendix. Default Filter Constraint Language” on page 937.

Return value

ConstraintInfoSeq
A sequence in which each element will be a structure including one of the input constraint expressions, along with the unique identifier assigned to it by the target filter object.

Exceptions

InvalidConstraint
This exception is raised if any of the constraints in the input sequence is not a

valid expression within the supported constraint grammar. Make sure that all the contraints in the input sequence are valid.

Remarks

The add_constraints operation accepts as input a sequence of constraint data structures, each element of which consists of a sequence of event type structures and a constraint expressed within the constraint grammar supported by the target object. Upon processing each constraint, the target object associates a numeric identifier with the constraint that is unique among all constraints it encapsulates. If any of the constraints in the input sequence is not a valid expression within the supported constraint grammar, the InvalidConstraint exception is raised. Upon successful processing of all input constraint expressions, the add_constraints operation returns a sequence in which each element will be a structure including one of the input constraint expressions, along with the unique identifier assigned to it by the target filter object.

Example

```
CosNotifyFilter::ConstraintExpSeq *cl =
   new CosNotifyFilter::ConstraintExpSeq;
CosNotifyFilter::Filter_var fi;
CosNotifyFilter::ConstraintInfoSeq* cis;

(*cl).length(3);
(*cl)[0].constraint_expr = CORBA::string_alloc(100);
(*cl)[1].constraint_expr = CORBA::string_alloc(100);
(*cl)[2].constraint_expr = CORBA::string_alloc(100);
strcpy((*cl)[0].constraint_expr, "$a > 1");
strcpy((*cl)[1].constraint_expr, "$b > 1");
strcpy((*cl)[2].constraint_expr, "$c > 1");
...
cis = fi->add_constraints(*cl);
```

Filter::constraint_grammar

The constraint_grammar attribute is a readonly attribute that identifies the particular grammar within which the constraint expressions encapsulated by the target filter object have meaning.

Original interface

CosNotifyFilter::Filter

IDL syntax

```
readonly attribute string constraint_grammar;
```

Parameters

constraint_grammar
This value is set to IBM_NTF_CTG that implies that our Constraint Grammar (in “Appendix. Default Filter Constraint Language” on page 937) is used as the default grammar.

Return value

None.

Exceptions

None.

Remarks

The value of this attribute is set upon creation of a filter object instance, based on the input provided to the factory creation operation for the filter instance.

The dependency of a filter object on its constraints being expressed within a particular constraint grammar manifests itself within the implementation of the match operations described below, that must be able to parse the constraints to determine whether or not a particular event satisfies one of them.

This implementation of the Notification Service supports an implementation of the Filter interface that supports the default constraint grammar described in “Appendix. Default Filter Constraint Language” on page 937. The value that the constraint_grammar attribute is set to in the case the target filter object supports this default grammar will be ″IBM_NTF_CTG″.

Filter::destroy

This destroy operation destroys the target filter object, invalidating its object reference.

Original interface

CosNotifyFilter::Filter

IDL syntax

```
void destroy();
```

Parameters

None.

Return value

None.

Exceptions

None.

Example

```
CosNotifyFilter::Filter_var fi;
...
fi->destroy();
```

Filter::get_all_constraints

The get_all_constraints operation returns all of the constraints currently encapsulated by the target filter object.

Original interface

CosNotifyFilter::Filter

IDL syntax

```
ConstraintInfoSeq get_all_constraints ();
```

Parameters

None.

Return value

ConstraintInfoSeq
A sequence of structures, each of which contains one of the constraints encapsulated by the target object along with its associated unique identifier.

Exceptions

None.

Example

```
CosNotifyFilter::Filter_var fi;
CosNotifyFilter::ConstraintInfoSeq* cis;
...
cis = fi->get_all_constraints();
```

Filter::get_constraints

The get_constraints operation is invoked to return a sequence of a subset of the constraints associated with the target filter object.

Original interface

CosNotifyFilter::Filter

IDL syntax

```
ConstraintInfoSeq get_constraints (in ConstraintIDSeq id_list)
   raises(ConstraintNotFound);
```

Parameters

id_list A sequence of numeric values that should correspond to the unique identifiers of constraints encapsulated by the target object.

Return value

ConstraintInfoSeq
A sequence of data structures, each of which contains one of the input identifiers along with its associated constraint.

Exceptions

ConstraintNotFound
This exception is raised if one of the input values does not correspond to the identifier of some encapsulated constraint. Modify the input list so that all its values correspond to the identifier of some encapsulated constraint.

Remarks

The operation accepts as input a sequence of numeric values that should correspond to the unique identifiers of constraints encapsulated by the target object. If one of the input values does not correspond to the identifier of some encapsulated constraint, the ConstraintNotFound exception is raised. Upon successful completion, this operation returns a sequence of data structures, each of which contains one of the input identifiers along with its associated constraint.

Example

```
CosNotifyFilter::ConstraintIDSeq id_list =
   new CosNotifyFilter::ConstraintIDSeq;
CosNotifyFilter::Filter_var fi;
CosNotifyFilter::ConstraintInfoSeq* cis;

(*id_list).length(2);
```

```
(*id_list)[0] = 3;                    // get the 3rd constraint
(*id_list)[1] = 5;                    // get the 5rd constraint
...
cis = fi->get_constraints(*id_list);
```

Filter::match_structured

The match_structured operation evaluates the filter constraints associated with the target filter object against an instance of an event supplied to the channel in the form of a Structured Event.

Original interface

CosNotifyFilter::Filter

IDL syntax

```
boolean match_structured (
      in CosNotification::StructuredEvent filterable_data)
   raises (UnsupportedFilterableData);
```

Parameters

filterable_data
A data structure of type CosNotification::StructuredEvent that contains an event to be evaluated.

Return value

boolean
A boolean value that will be TRUE in cases where the input event satisfies one of the filter constraints, and FALSE otherwise.

Exceptions

UnsupportedFilterableData
This exception is raised if the input parameter contains data that the match operation cannot handle. Check and make sure that the data in the event has the correct format.

Remarks

The operation accepts as input a data structure of type CosNotification::StructuredEvent that contains an event to be evaluated, and returns a boolean value that will be TRUE in cases where the input event satisfies one of the filter constraints, and FALSE otherwise. If the input parameter contains data that the match operation is not designed to handle, the UnsupportedFilterableData exception will be raised.

Example

```
CosNotification::StructuredEvent *ev_s = new CosNotification::StructuredEvent;
CORBA::Short short_var;
time_t time_out, stop_time;
tm t1;
CosNotifyFilter::Filter_var fi;
CORBA::Boolean match;

ev_s->header.fixed_header.domain_type = CORBA::string_alloc(100);
strcpy(ev_s->header.fixed_header.domain_type, "dt");
ev_s->header.fixed_header.event_type = CORBA::string_alloc(100);
strcpy(ev_s->header.fixed_header.event_type, "et");
ev_s->header.fixed_header.event_name = CORBA::string_alloc(100);
strcpy(ev_s->header.fixed_header.event_name, "en");

ev_s->header.variable_header.length(3);
ev_s->header.variable_header[0].name = CORBA::string_alloc(100);
strcpy(ev_s->header.variable_header[0].name, "Priority");
short_var = 6;
ev_s->header.variable_header[0].value <<= short_var;

ev_s->header.variable_header[1].name = CORBA::string_alloc(100);
strcpy(ev_s->header.variable_header[1].name, "Timeout");
time_out = 2 * 24 * 60 * 60 * 10000000;   // 2 days in 100 nanoseconds
ev_s->header.variable_header[1].value <<= time_out;
ev_s->header.variable_header[2].name = CORBA::sting_alloc(100);
strcpy(ev_s->header.variable_header[2].name, "StopTime");
t1.tm_year    = 99;           // 1999 July 4, 12.01
t1.tm_mon     = 6;            // month is zero-based
t1.tm_mday    = 4;            // day is one-based
t1.tm_hour    = 12;
t1.tm_min     = 1;
t1.tm_sec     = 0;
stop_time = mktime(&t1);   // convert into time_t format
ev_s->header.variable_header[2].value <<= stop_time;

ev_s->filterable_data.length(2);
ev_s->filterable_data[0].name = CORBA::string_alloc(10);
strcpy(ev_s->filterable_data[0].name, "COMPANY");
ev_s->filterable_data[0].value <<= "IBM";

ev_s->filterable_data[1].name = CORBA::string_alloc(10);
strcpy(ev_s->filterable_data[1].name, "PRICE");
short_var = 150;
ev_s->filterable_data[1].value <<= short_var;

ev_s->remainder_of_body <<= short_var;
...
match = fi->match_structured(*ev_s);
```

Filter::modify_constraints

The modify_constraints operation is invoked by a client in order to modify the constraints associated with the target filter object.

Original interface

CosNotifyFilter::Filter

IDL syntax

```
void modify_constraints (in ConstraintIDSeq del_list,
                         in ConstraintInfoSeq modify_list)
   raises(InvalidConstraint, ConstraintNotFound);
```

Parameters

del_list
Sequence of numeric values that are each intended to be the unique identifier associated with one of the constraints currently encapsulated by the target filter object.

modify_list
Sequence of structures each of which contains a constraint structure and a numeric value. The numeric value contained by each element of the sequence is intended to be the unique identifier associated with one of the constraints currently encapsulated by the target filter object.

Return value

None.

Exceptions

InvalidConstraint
This exception is raised if any of the constraints in the second input sequence is not a valid expression within the supported constraint grammar. Make sure that all the constraints in the second input sequence are valid.

ConstraintNotFound
This exception is raised if any of the numeric values supplied within either of the two input sequences does not correspond to the unique identifier associated with some constraint currently encapsulated by the target filter object. Modify the input list so that all its values correspond to the identifier of some encapsulated constraint.

The modify_constraints operation can be used both to remove constraints currently associated with the target filter object, and to modify the constraint expressions of constraints that have previously been added to the target filter object.

The operation accepts two input parameters. The first input parameter is a sequence of numeric values that are each intended to be the unique identifier associated with one of the constraints currently encapsulated by the target filter object. If all input values supplied within a particular invocation of this operation are valid, then the specific constraints identified by the values contained in the first input parameter will be deleted from the list of those encapsulated by the target filter object.

The second input parameter to this operation is a sequence of structures, each of which contains a constraint structure and a numeric value. The numeric value contained by each element of the sequence is intended to be the unique identifier associated with one of the constraints currently encapsulated by the target filter object. If all input values supplied within a particular invocation of this operation are valid, then the constraint expression associated with the already encapsulated constraint identified by the numeric value contained within each element of the input sequence will be modified to the new constraint expression that is contained within the same sequence element.

If any of the numeric values supplied within either of the two input sequences does not correspond to the unique identifier associated with some constraint currently encapsulated by the target filter object, the ConstraintNotFound exception is raised. If any of the constraint expressions supplied within an element of the second input sequence is not a valid expression in terms of the constraint grammar supported by the target object, the InvalidConstraint exception is raised.

Example

```
CosNotifyFilter::ConstraintIDSeq *del_list =
   new CosNotifyFilter::ConstraintIDSeq;
CosNotifyFilter::ConstraintInfoSeq *modify_list =
   new CosNotifyFilter::ConstraintInfoSeq;
CosNotifyFilter::Filter_var fi;

(*del_list).length(2);
(*del_list)[0] = 3;                     // delete the 3rd constraint
(*del_list)[1] = 5;                     // delete the 5rd constraint

(*modify_list).length(3);
(*modify_list)[0].constraint_id = 2;    // modify the 2nd constraint
(*modify_list)[0].constraint_expression.constraint_expr =
   CORBA::string_alloc(100);
strcpy((*modify_list)[0].constraint_expression.constraint_expr, "$x > 1");
(*modify_list)[1].constraint_id = 4;    // modify the 4nd constraint
(*modify_list)[1].constraint_expression.constraint_expr =
   CORBA::string_alloc(100);
strcpy((*modify_list)[1].constraint_expression.constraint_expr, "$y > 1");
(*modify_list)[2].constraint_id = 6;    // modify the 6nd constraint
(*modify_list)[2].constraint_expression.constraint_expr =
   CORBA::string_alloc(100);
strcpy((*modify_list)[2].constraint_expression.constraint_expr, "$z > 1");
...
fi->modify_constraints(*del_list, *modify_list);
```

Filter::remove_all_constraints

The remove_all_constraints operation is invoked to remove all of the constraints currently encapsulated by the target filter object. Upon completion, the target filter object will still exist but have no constraints associated with it.

Original interface

CosNotifyFilter::Filter

IDL syntax

```
void remove_all_constraints();
```

Parameters

None.

Return value

None.

Exceptions

None.

Example

```
CosNotifyFilter::Filter_var fi;
...
fi->remove_all_constraint();
```

FilterAdmin Interface

The FilterAdmin interface defines operations that enable an object supporting this interface to manage a list of filter objects, each of which supports the Filter interface.

File name

CosNotifyFilter.idl

Intended usage

This interface is intended to be an abstract interface that is inherited by all of the Proxy and Admin interfaces defined by the Notification Service.

IDL syntax

```
interface FilterAdmin
{
   FilterID add_filter (in FIlter new_filter);
   void remove_filter (in FilterID filter)
      raises (FilterNotFound);

   Filter get_filter (in FilterID filter)
      raises (FilterNotFound);

   FilterIDSeq get_all_filters();
   void remove_all_filters();
};
```

Supported operations

FilterAdmin::add_filter
FilterAdmin::get_all_filters
FilterAdmin::get_filter
FilterAdmin::remove_all_filters
FilterAdmin::remove_filter

FilterAdmin::add_filter

The add_filter operation accepts as input the reference to an object supporting the Filter interface. The affect of this operation is that the input filter object is appended to the list of filter objects associated with the target object upon which the operation was invoked. The operation associates with the newly added filter object a numeric identifier that is unique among all filter objects currently associated with the target, and returns that value as the result of the operation.

Original interface

CosNotifyFilter::FilterAdmin

IDL syntax

```
FilterID add_filter (in Filter new_filter);
```

Parameters

new_filter
A reference to an object supporting the Filter interface.

Return value

None.

Exceptions

None.

Example

```
CosNotifyFilter::Filter_var fi = NULL;
CosNotifyChannelAdmin::SupplierAdmin_var sa = NULL;
...
sa->add_filter(fi);
```

FilterAdmin::get_all_filters

The get_all_filters operation accepts no input parameters, and returns the list of unique identifiers that correspond to all of the filters currently associated with the target object.

Original interface

CosNotifyFilter::FilterAdmin

IDL syntax

```
FilterIDSeq get_all_filters();
```

Example

```
CosNotifyChannelAdmin::SupplierAdmin_var sa = NULL;
CosNotifyFilter::FilterIDSeq *f_seq;
...
f_seq = sa->get_all_filters();
```

FilterAdmin::get_filter

The get_filter operation accepts as input a numeric identifier that is intended to correspond to one of the filter objects currently associated with the target object. If this is the case, the object reference of the corresponding filter object is returned. Otherwise, the FilterNotFound exception is raised.

Original interface

CosNotifyFilter::FilterAdmin

IDL syntax

```
Filter get_filter (in FilterID filter
   raises (FilterNotFound);
```

Parameters

filter A numeric identifier that is intended to correspond to one of the filter objects currently associated with the target object.

Return value

Filter The object reference of the corresponding filter object.

Exceptions

FilterNotFound

This exception is raised if the input identifier FilterID does not correspond with any filter object currently associated with the target object. Use the correct value for the FilterID.

Example

```
CosNotifyChannelAdmin::SupplierAdmin_var sa = NULL;
CosNotifyFilter::Filter_var fi;
CosNotifyFilter::FilterID fid;
...
fi = sa->get_filter(fid);
```

FilterAdmin::remove_all_filters

The remove_all_filters operation accepts no input parameters, and removes all filter objects from the list of those currently associated with the target object.

Original interface

CosNotifyFilter::FilterAdmin

IDL syntax

```
void remove_all_filters();
```

Parameters

None.

Return value

None.

Exceptions

None.

Example

```
CosNotifyChannelAdmin::SupplierAdmin_var sa = NULL;
...
sa->remove_all_filters();
```

FilterAdmin::remove_filter

The remove_filter operation accepts as input a numeric value that is intended to be the unique identifier of a filter object that is currently associated with the target object. If identifier supplied does correspond to a filter object currently associated with the target object, then the corresponding filter object will be removed from the list of filters associated with the target object. Otherwise, the FilterNotFound exception will be raised.

Original interface

CosNotifyFilter::FilterAdmin

IDL syntax

```
void remove_filter (in FilterID filter)
   raises (FilterNotFound);
```

Parameters

filter A numeric value that is intended to be the unique identifier of a filter object that is currently associated with the target object.

Return value

None.

Exceptions

FilterNotFound
This exception is raised if the input identifier FilterID does not correspond with any filter object currently associated with the target object. Use the correct value for the FilterID.

Example

```
CosNotifyFilter::FilterID fid;
CosNotifyChannelAdmin::SupplierAdmin_var sa = NULL;
...
sa->remove_filter(fid);
```

FilterFactory Interface

The FilterFactory interface defines operations for creating filter objects.

File name

CosNotifyFilter.idl

IDL syntax

```
interface FilterFactory
{
   Filter create_filter (in string constraint_grammar)
      raises(InvalidGrammar);
};
```

Supported operations

FilterFactory::create_filter

FilterFactory::create_filter

The create_filter operation is responsible for creating a new forwarding filter object.

Original interface

CosNotifyFilter::FilterFactory

IDL syntax

```
Filter create_filter (in string constraint_grammar)
   raises(InvalidGrammar);
```

Parameters

contraint_grammar
A string parameter that identifies the grammar in which constraints associated with this filter will have meaning. In this implementation it is ''IBM_NTF_CTG''.

Return value

Filter A reference to an object supporting the Filter interface.

Exceptions

InvalidGrammar
This exception is raised if the client invoking this operation supplies as input the name of a grammar that is not supported by any forwarding filter implementation this factory is creating. Make sure that the correct grammar (IBM_NTF_CTG) is passed to this function.

Remarks

It takes as input a string parameter that identifies the grammar in which constraints associated with this filter will have meaning. If the client invoking this operation supplies as input the name of a grammar that is not supported by any forwarding filter implementation this factory is capable of creating, the InvalidGrammar exception is raised. Otherwise, the operation returns the reference to an object supporting the Filter interface, that can subsequently be configured to support constraints in the appropriate grammar.

Example

```
INotifyFilterManagedClient::FilterFactory_var ff = NULL;
CosNotifyFilter::Filter_var fi = NULL;
...
fi = ff->create_filter("IBM_NTF_CTG");
```

Chapter 12. CosObjectIdentity in the Identity Service

CosObjectIdentity is the only module in the Identity Service.

CosObjectIdentity Module

Supports methods for identifying objects in a heterogeneous CORBA-based environment and establishing whether two object references are for the same object.

File name

CosObjectIdentity.idl

Intended usage

The CosObjectIdentity module contains only one interface: IdentifyableObject Interface. This interface is an abstract interface which has only abstract behavior that individual interfaces can then inherit for implementation.

Types

```
typedef unsigned long ObjectIdentifier;
```

Interfaces

CosObjectIdentity::IdentifiableObject Interface

IdentifiableObject Interface

Supports operations for identifying objects in a heterogeneous CORBA-based environment and establishing whether two object references are for the same object.

File name

CosObjectIdentity.idl

Types

```
typedef unsigned long ObjectIdentifier;
```

Exceptions

CORBA standard exceptions.

Attributes

IdentifiableObject::constant_random_id

Supported operations

IdentifiableObject::is_identical

IdentifiableObject::constant_random_id

Returns the constant random id of an object.

Original interface

CosObjectIdentity::IdentifiableObject Interface

IDL syntax

```
readonly attribute ObjectIdentifier constant_random_id;
```

Remarks

This attribute is used to perform a first order identity check on two objects. The constant random id is not guaranteed to be unique; that is, another identifiable object can have the same constant random id. However, if objects return different constant random id, the client can determine that the two identifiable objects are not identical. To determine if the two identifiable objects are identical when a first order test fails, the is_identical Operation must be used.

Example

```
// A CosObjectIdentity usage example.
// For simplicity, error and exception checking and cleanup are ommitted.

  #include <CosObjectIdentity.hh>
  CORBA::Object_ptr objPtr1;
  CORBA::Object_ptr objPtr2;

  CosObjectIdentity::ObjectIdentifier objId;
  CosObjectIdentity::IdentifiableObject_ptr iobjPtr1;
  CosObjectIdentity::IdentifiableObject_ptr iobjPtr2;

  IString path1 = "vehicules/vans";
  IString path2 = "vehicules/vans/chrysler";

  // Resolve the first NamingContext, "vehicules/vans"
  objPtr1  = NamingContext_ptr->resolve_with_string(path1);
  iobjPtr1 = CosObjectIdentity::IdentifiableObject::_narrow(objPtr1);

  // Resolve the second NamingContext, "vehicules/vans/chrysler"
  objPtr2  = NamingContext_ptr->resolve_with_string(path2);
  iobjPtr2 = CosObjectIdentity::IdentifiableObject::_narrow(objPtr2);


  // Get the constant random id of an object.
  objId = iobjPtr1->constant_random_id();
```

```
// Compare two NamingContext objects and see if they are identical.
if ( iobjPtr1->is_identical(iobjPtr2) )
  cout << "NC of " << path1 << " equals " << path2 << endl;
else
  cout << "NC of " << path1 << " does not equal " << path2 << endl;
```

IdentifiableObject::is_identical

Determines if two objects are identical.

Original interface

CosObjectIdentity::IdentifiableObject Interface

IDL syntax

```
boolean is_identical (in IdentifiableObject other_object);
```

Parameters

other_object
Identity comparison requires two objects: a target object and some other object. The is_identical method is invoked on a target object and compares the identity of the target object to the other_object.

Return value

TRUE Indicates that the target object and the other_object are identical.

FALSE Indicates that the target object and the other_object are not identical.

Exceptions

CORBA standard exceptions.

Remarks

This operation is used to determine if two objects are identical.

Example

```
// A CosObjectIdentity usage example.
// For simplicity, error and exception checking and cleanup are ommitted.

  #include <CosObjectIdentity.hh>
  CORBA::Object_ptr objPtr1;
  CORBA::Object_ptr objPtr2;

  CosObjectIdentity::ObjectIdentifier objId;
  CosObjectIdentity::IdentifiableObject_ptr iobjPtr1;
```

```
CosObjectIdentity::IdentifiableObject_ptr iobjPtr2;

IString path1 = "vehicules/vans";
IString path2 = "vehicules/vans/chrysler";

// Resolve the first NamingContext, "vehicules/vans"
objPtr1  = NamingContext_ptr->resolve_with_string(path1);
iobjPtr1 = CosObjectIdentity::IdentifiableObject::_narrow(objPtr1);

// Resolve the second NamingContext, "vehicules/vans/chrysler"
objPtr2  = NamingContext_ptr->resolve_with_string(path2);
iobjPtr2 = CosObjectIdentity::IdentifiableObject::_narrow(objPtr2);


// Get the constant random id of an object.
objId = iobjPtr1->constant_random_id();


// Compare two NamingContext objects and see if they are identical.
if ( iobjPtr1->is_identical(iobjPtr2) )
  cout << "NC of " << path1 << " equals " << path2 << endl;
else
  cout << "NC of " << path1 << " does not equal " << path2 << endl;
```

Chapter 13. CosQuery in the Query Service

The other modules in the Query Service are:
- CosQueryCollection
- IExtendedQuery

CosQuery Module

The CosQuery module describes the interface of OMG standard Query service. This module supports methods that allow queries to be submitted and managed.

File name

CosQuery.idl

Intended usage

The key interface in this module is the CosQuery::QueryEvaluator Interface. Queries can be submitted using operations on this interface.

Types

```
typedef CosQueryCollection::ParameterList ParameterList;
typedef CosQueryCollection::ArgList ArgList;
```

Interfaces

CosQuery::QueryEvaluator Interface

QueryEvaluator Interface

Provides an abstract base class to submit queries and obtain supported and default query language types.

File name

CosQuery.idl

Intended usage

The QueryEvaluator interface contains ql_types and default_ql_type attributes and evaluate method. The ql_types and default_ql_type specify the supported query language types and default query language type, and the evaluate method is used to submit a query. The result of the query submitted using evaluate method is returned using the return value of type any.

IDL syntax

```
interface QueryEvaluator {

     readonly attribute QLType_sequence ql_types;
     readonly attribute QLType default_ql_type;

     any evaluate (
                  in string query,
                  in QLType ql_type,
                  in ParameterList params)
     raises(QueryTypeInvalid,
            QueryInvalid,
            QueryProcessingError);
};
```

Supported operations

QueryEvaluator::evaluate

QueryEvaluator::evaluate

Used to submit a query.

Original interface

CosQuery::QueryEvaluator Interface

IDL syntax

```
any evaluate(
             in string query,
             in QLType ql_type,
             in ParameterList params)
    raises(QueryTypeInvalid,
           QueryInvalid,
           QueryProcessingError);
```

Parameters

query Specifies the OOSQL query string for the query.

ql_type
Specifies the language type used for the query. In the current version of the component broker series, only OOSQL language type is supported and therefore this parameter is ignored.

params
Used for passing in optional parameters for the query. In the current version of the component broker series no optional parameter is required.

Return value

any The value field of the returned CORBA::Any is a IManagedCollection::IIterator. The result of the query can be accessed using this iterator

Exceptions

CosQuery::QueryTypeInvalid - Query language is not supported by the QueryEvaluator.

CosQuery::QueryInvalid - Query string syntax or semantics is incorrect or the input parameter list is incorrect. A string variable providing text explanation of the error is contained in the exception.

CosQuery::QueryProcessingError - Any error is encountered during query processing. A string variable providing text explanation of the error is contained in the exception.

Remarks

This operation can be used to submit an OOSQL query against Home or any other queryable collections. This operation is defined by OMG query service standard. This operation is supported in Component Broker mainly to preserve compliance with OMG query service standard. IExtendedQuery::QueryEvaluator::evaluate_to_iterator and IExtendedQuery::QueryEvaluator::evaluate_to_data_array operations provide more direct and flexible ways of submitting an OOSQL query and are therefore more preferable.

Note: The collating sequence depends on NLS settings on the Component Broker Server system.

Chapter 14. CosQueryCollection in the Query Service

The other modules in the Query Service are:
- CosQuery
- IExtendedQuery

CosQueryCollection Module

The CosQueryCollection module contains data types used in the OMG standard query service.

AIX WIN Supported on AIX and Windows NT only

File name

CosQColl.idl

Types

```
typedef string Istring;
typedef NVPair {Istring name; any value;};
typedef sequence<NVPair> ParameterList;
typedef sequence<any> ArgList;
```

Chapter 15. CosStream in the Externalization Service

The other module in the Externalization Service is:
- IExtendedStream

CosStream Module

Provides interfaces that support the streaming of object state or data into a sequentially organized buffer.

File name

CosStream.idl

Intended usage

This module provides interfaces that allow an object to place it's essential state into a sequential memory buffer that can, for example, be written to disk or shipped across a network. The information can then be used to create another instance of the same type of object, with the same state, in the same or another process. The Externalization Service is not a general purpose service for client programming. See the Externalization Service chapter in the *Component Broker Advanced Programming Guide* for more information on how and where the Externalization Service is used in Component Broker.

Exceptions

CosStream::ObjectCreationError
CosStream::StreamDataFormatError

Interfaces

CosStream::Streamable Interface
CosStream::StreamIO Interface

Streamable Interface

Defines a set of methods that an object must support in order to be externalizable in one process, and internalizable in the same or any other process.

File name

CosStream.idl

Intended usage

This interface is intended to provide the functionality necessary in order for an object to be externalized in one process, and internalized in the same or any other process, thereby reproducing the object in another distinct process.

Ancestor interfaces

CosObjectIdentity::IdentifiableObject

Exceptions

CosStream::ObjectCreationError
CosStream::StreamDataFormatError

IDL syntax

```
interface Streamable : CosObjectIdentity::IdentifiableObject {
    readonly attribute CosLifeCycle::Key external_form_id;
    void externalize_to_stream (
      in StreamIO targetStreamIO);
    void internalize_from_stream (
      in StreamIO sourceStreamIO,
      in CosLifeCycle::FactoryFinder there)
    raises (CosLifeCycle::NoFactory,
      ObjectCreationError,
      StreamDataFormatError);
};
```

Attributes

Streamable::external_form_id

Supported operations

Streamable::externalize_to_stream
Streamable::internalize_from_stream

Streamable::external_form_id

This attribute is defined to return the CosLifeCycle::Key data attribute for the Streamable object, which is intended to be usable as input to a CosLifeCycle::FactoryFinder when trying to locate a factory for creating objects of this same type.

Original interface

CosStream::Streamable

IDL syntax

```
readonly attribute CosLifeCycle::Key external_form_id;
```

Return value

CosLifeCycle::Key
The CosLifeCycle::Key associated with the Streamable object.

Remarks

This attribute is not actually used by Component Broker and the value returned for this attribute generally will not be usable as input to a CosLifeCycle::FactoryFinder. In some cases, it may throw a CORBA::NO_IMPLEMENT exception.

Streamable::externalize_to_stream

Writes to the stream the object state data necessary to reinitialize an object instance with that state.

Original interface

CosStream::Streamable

IDL syntax

```
void externalize_to_stream (in StreamIO targetStreamIO);
```

Parameters

StreamIO::targetStreamIO

Remarks

This operation is used to write the essential object state to a stream. This operation is responsible for decomposing an externalizable object's internal state into a series of primitive data type values or object references, or both. It must write out all the necessary primitive data values using the write_<type>() operations on the targetStreamIO for non-object data types. For object references, the reference must first be stringified using the ORB::object_to_string method and then written to the targetStreamIO using the write_string() operation. It is up to the Streamable implementor to decide which data values and referenced objects compose the essential state and therefore need to be externalized.

Streamable::internalize_from_stream

Provides the ability to reinitialize an object from the contents of a stream.

Original interface

CosStream::Streamable

IDL syntax

```
void internalize_from_stream (in StreamIO sourceStreamIO,
                              in CosLifeCycle::FactoryFinder there);
                      raises (CosLifeCycle::NoFactory,
                              ObjectCreationError,
                              StreamDataFormatError);
```

Parameters

sourceStreamIO
The source StreamIO containing the object state data

there The CosLifeCycle::FactoryFinder intended to be used to find a factory capable of creating an objects of the correct type as those which are imbeded in the stream. This parameter is not used by Component Broker and generally the nil object is passed on calls to internalize_from_stream.

Exceptions

StreamDataFormatError - originates from the read_<type>() operation on the sourceStreamIO parameter when the data in the stream cannot be converted to the type defined for that operation.

ObjectCreationError - This exception is not raised by the Component Broker implementation.

CosLifeCycle::NoFactory - This exception is not raised by the Component Broker implementation.

Remarks

This operation is intended to be used to reinitialize objects whose object state data has been previously externalized into a StreamIO. The order and type of the read_<type>() methods coded in this method must exactly match the order and type of the write_<type>() methods used in the externalize_to_stream method when the objects state was originally streamed.

StreamIO Interface

Provides the functionality necessary to write object state information into a sequential memory buffer, and to read object state information from a memory buffer. This memory buffer is also referred to as the StreamIO.

The Externalization Service is not thread-safe. Accessing the same StreamIO from different threads can corrupt the StreamIO object.

File name

CosStream.idl

Intended usage

This interface is intended to allow a user to write object state information into a StreamIO object, that contains a memory buffer, and read object state information from this StreamIO.

There are two different data formats for StreamIO objects. One format is the CORBA-defined format for StreamIO, referred to as the ″CORBA Standard Stream Data Format,″ which is intended to allow different CORBA-compliant ORBs to exchange objects through the use of the Externalization Service. This format is not currently used within Component Broker.

The other format is the IBM-defined data format, referred to as the ″IBM Stream Data Format,″ which optimizes processing by only performing data conversions (e.g., endian format, code pages) when the target platform is different from the source platform. This data format can only be used when the processes writing to and reading from the StreamIO are IBM CBSeries processes.

Local-only

True - This interface itself is not local only, but all implementations of this interface in Component Broker are local-only.

Exceptions

StreamDataFormatError - error detected in the data format of the external representation.

Supported operations

StreamIO::read_boolean
StreamIO::read_char
StreamIO::read_double
StreamIO::read_float
StreamIO::read_string
StreamIO::read_long
StreamIO::read_long_long
StreamIO::read_octet
StreamIO::read_short
StreamIO::read_unsigned_long
StreamIO::read_unsigned_long_long
StreamIO::read_unsigned_short
StreamIO::write_boolean
StreamIO::write_char
StreamIO::write_double
StreamIO::write_float
StreamIO::write_long
StreamIO::write_long_long
StreamIO::write_octet
StreamIO::write_short
StreamIO::write_string

StreamIO::write_unsigned_long
StreamIO::write_unsigned_long_long
StreamIO::write_unsigned_short

StreamIO::read_boolean

This method reads a boolean variable from the StreamIO.

Original interface

CosStream::StreamIO

IDL syntax

```
boolean read_boolean() raises (StreamDataFormatError);
```

Return value

The boolean value read from the stream.

Exceptions

CosStream::StreamDataFormatError

StreamIO::read_char

This method will read and return a single character from the StreamIO. The character must be a 1 byte character.

Original interface

CosStream::StreamIO

IDL syntax

```
char read_char () raises (StreamDataFormatError);
```

Return value

A single character.

Exceptions

CosStream::StreamDataFormatError

StreamIO::read_double

This method reads a double value from the StreamIO.

Original interface

CosStream::StreamIO

IDL syntax

```
double read_double() raises (StreamDataFormatError);
```

Return value

The double value read from the StreamIO.

Exceptions

CosStream::StreamDataFormatError

StreamIO::read_float

Reads a float value from the StreamIO.

Original interface

CosStream::StreamIO

IDL syntax

```
float read_double() raises (StreamDataFormatError);
```

Return value

The float value read from the StreamIO.

Exceptions

CosStream::StreamDataFormatError

StreamIO::read_long

Reads a long value from the StreamIO.

Original interface

CosStream::StreamIO

IDL syntax

```
long read_long() raises (StreamDataFormatError);
```

Return value

The long value read from the StreamIO.

Exceptions

CosStream::StreamDataFormatError

StreamIO::read_long_long

Reads a long long value from the StreamIO.

Supported on AIX and Windows NT only

Original interface

CosStream::StreamIO

IDL syntax

```
long long read_long_long() raises (StreamDataFormatError);
```

Return value

The long long value read from the StreamIO.

Exceptions

CosStream::StreamDataFormatError

StreamIO::read_octet

Reads an octet value from the StreamIO.

Original interface

CosStream::StreamIO

IDL syntax

```
octet read_octet() raises (StreamDataFormatError);
```

Return value

The octet value read from the StreamIO.

Exceptions

CosStream::StreamDataFormatError

StreamIO::read_short

Reads a short value from the StreamIO.

Original interface

CosStream::StreamIO

IDL syntax

```
short read_short() raises (StreamDataFormatError);
```

Return value

The short value read from the StreamIO.

Exceptions

CosStream::StreamDataFormatError

StreamIO::read_string

Reads a string value from the StreamIO. The string may be from a single-byte or multi-byte character set and is terminated by a single byte of x'00'.

Original interface

CosStream::StreamIO

IDL syntax

```
string read_string() raises (StreamDataFormatError);
```

Return value

The string value read from the StreamIO.

Exceptions

CosStream::StreamDataFormatError

StreamIO::read_unsigned_long

Reads an unsigned long value from the StreamIO.

Original interface

CosStream::StreamIO

IDL syntax

```
unsigned long read_unsigned_long() raises (StreamDataFormetError);
```

Return value

The unsigned long value read from the StreamIO.

Exceptions

CosStream::StreamDataFormatError

StreamIO::read_unsigned_long_long

Reads an unsigned long long value from the StreamIO.

Supported on AIX and Windows NT only

Original interface

CosStream::StreamIO

IDL syntax

```
unsigned long long read_unsigned_long_long() raises (StreamDataFormatError);
```

Return value

The unsigned long long value read from the StreamIO.

Exceptions

CosStream::StreamDataFormatError

StreamIO::read_unsigned_short

Reads an unsigned short value from the StreamIO.

Original interface

CosStream::StreamIO

IDL syntax

```
unsigned short read_unsigned_short() raises (StreamDataFormatError);
```

Return value

The unsigned short value read from the StreamIO.

Exceptions

CosStream::StreamDataFormatError

StreamIO::read_wchar

Reads a wchar from the StreamIO. A wchar is a wide character which is either 2 or 4 bytes.

Original interface

CosStream::StreamIO

IDL syntax

```
wchar read_wchar () raises (StreamDataFormatError) ;
```

Exceptions

CosStream::StreamDataFormatError

StreamIO::read_wstring

Reads a wstring from the StreamIO. A wstring is composed of fixed size characters which are all either 2 or 4 bytes each. The wstring is terminated by a wide character (2 or 4 bytes) of x'00'.

Original interface

CosStream::StreamIO

IDL syntax

```
wstring read_wstring () raises (StreamDataFormatError) ;
```

Exceptions

CosStream::StreamDataFormatError

StreamIO::write_boolean

Writes a boolean value to the StreamIO.

Original interface

CosStream::StreamIO

IDL syntax

```
void write_boolean (in boolean aBoolean);
```

Parameters

The boolean value to be written to the StreamIO.

StreamIO::write_char

Writes a char value to the StreamIO. The character value must be a 1 byte character.

Original interface

CosStream::StreamIO

IDL syntax

```
void write_char (in char aChar);
```

Parameters

The char value to be written to the StreamIO.

StreamIO::write_double

Writes a double value to the StreamIO.

Original interface

CosStream::StreamIO

IDL syntax

```
void write_double (in double aDouble);
```

Parameters

The double value to be written to the StreamIO.

StreamIO::write_float

Writes a float value to the StreamIO.

Original interface

CosStream::StreamIO

IDL syntax

```
void write_float (in float aFloat);
```

Parameters

The float value to be written to the StreamIO.

StreamIO::write_long

Writes a long value to the StreamIO.

Original interface

CosStream::StreamIO

IDL syntax

```
void write_long (in long aLong);
```

Parameters

The long value to be written to the StreamIO.

StreamIO::write_long_long

Writes a long long value to the StreamIO.

Supported on AIX and Windows NT only

Original interface

CosStream::StreamIO

IDL syntax

```
void write_long_long (in long long val);
```

Parameters

The long long value to be written to the StreamIO.

StreamIO::write_octet

Writes an octet value to the StreamIO.

Original interface

CosStream::StreamIO

IDL syntax

```
void write_octet (in octet anOctet);
```

Parameters

The octet value to be written to the StreamIO.

StreamIO::write_short

Writes a short value to the StreamIO.

Original interface

CosStream::StreamIO

IDL syntax

```
void write_short (in short aShort);
```

Parameters

The short value to be written to the StreamIO.

StreamIO::write_string

Writes a string value to the StreamIO. The string may be from a single-byte or multi-byte character set and is terminated by a single byte of x'00'.

Original interface

CosStream::StreamIO

IDL syntax

```
void write_string (in string aString);
```

Parameters

The string value to be written to the StreamIO.

StreamIO::write_unsigned_long

Writes an unsigned long value to the StreamIO.

Original interface

CosStream::StreamIO

IDL syntax

```
void write_unsigned_long (in unsigned long anUnsignedLong);
```

Parameters

The unsigned long value to be written to the StreamIO.

StreamIO::write_unsigned_long_long

Writes an unsigned long long value to the StreamIO.

Supported on AIX and Windows NT only

Original interface

CosStream::StreamIO

IDL syntax

```
void write_unsigned_long_long (in unsigned long long val);
```

Parameters

The unsigned long long value to be written to the StreamIO.

StreamIO::write_unsigned_short

Writes an unsigned short value to the StreamIO.

Original interface

CosStream::StreamIO

IDL syntax

```
void write_unsigned_short (in unsigned short anUnsignedShort);
```

Parameters

The unsigned short value to be written to the StreamIO.

StreamIO::write_wchar

Writes a wchar to the StreamIO. A wchar is a wide character from either a 2 or 4 byte fixed size code set.

Original interface

CosStream::StreamIO

IDL syntax

```
void write_wchar (in wchar aWChar) ;
```

Parameters

The wchar to be written to the StreamIO.

StreamIO::write_wstring

Writes a wstring to the StreamIO. A wstring is composed of fixed size characters which are all either 2 or 4 bytes each. The wstring must be terminated by a wide character (2 or 4 bytes) of x'00'.

Original interface

CosStream::StreamIO

IDL syntax

```
void write_wstring (in wstring aWString) ;
```

Parameters

The wstring to be written to the StreamIO.

Chapter 16. CosTransactions in the Transaction Service

The CosTransactions Module is the only module in the Transaction Service.

CosTransactions Module

File stem

CosTransactions

Types

Status This type enumerates the various states through which a transaction goes during its lifetime. A special value is used to indicate that there is no transaction.

```
enum Status
{
    StatusActive,
    StatusMarkedRollback,
    StatusPrepared,
    StatusCommitted,
    StatusRolledBack,
    StatusUnknown,
    StatusNoTransaction,
    StatusPreparing,
    StatusCommitting,
    StatusRollingBack
};
```

StatusActive
The transaction has begun, has not yet been committed or rolled back, and has not been marked rollback-only.

StatusMarkedRollBack
The transaction has begun, has not yet been committed or rolled back, and has been marked rollback-only.

StatusPrepared
The transaction is “indoubt”. This means the local CosTransactions::Coordinator object is waiting for information from another object to decide the outcome of the transaction. This status can be returned by a coordinator after it has prepared, or inside a Resource object’s commit, rollback or commit_one_phase operation.

StatusCommitted
The transaction has been committed. This status is returned by RecoveryCoordinator::replay_completion. Note that it is not returned generally, because the objects associated with a transaction are destroyed immediately after the transaction has committed.

StatusRolledBack
The transaction has been rolled back. This status is returned in a Resource object's rollback operation, or by RecoveryCoordinator::replay_completion.

StatusUnknown
The status of the transaction is not currently known. This occurs in a subordinate server process during recovery, when the superior has not been contacted. This status is returned only by RecoveryCoordinator::replay_completion.

StatusNoTransaction
There is no current transaction. This status is returned only by Current::get_status.

Vote This type enumerates the votes available to a Resource for the Resource::prepare operation.

```
enum Vote
{
    VoteCommit,
    VoteRollback,
    VoteReadOnly
};
```

Exceptions

HeuristicRollback
HeuristicCommit
HeuristicMixed
HeuristicHazard
Inactive
InvalidControl
NotPrepared
NoTransaction
NotSubtransaction
SubtransactionsUnavailable
Unavailable

Interfaces

CosTransactions::Control Interface

CosTransactions::Coordinator Interface

CosTransactions::Current Interface

CosTransactions::RecoveryCoordinator Interface

CosTransactions::Resource Interface

CosTransactions::Synchronization Interface

CosTransactions::Terminator Interface

CosTransactions::TransactionalObject Interface

CosTransactions::TransactionFactory Interface

Control Interface

The Control interface defines operations to allow a program to explicitly manage or propagate a transaction. An object supporting the Control interface is implicitly associated with one transaction only.

File name

CosTransactions

Intended usage

A Control object is used to represent a transaction when, for example, you suspend or resume it.

IDL syntax

```
interface Control
{
    Terminator get_terminator()
        raises(Unavailable);
    Coordinator get_coordinator()
        raises(Unavailable);
};
```

Exceptions

Unavailable

Supported operations

Control::get_coordinator
Control::get_terminator

Control::get_coordinator

Returns an object that supports the Coordinator Interface for the transaction represented by the Control object.

Original interface

CosTransactions::Control Interface

IDL syntax

```
Coordinator get_coordinator()
  raises (Unavailable);
```

Return value

Coordinator

An object that supports the Coordinator interface for the transaction represented by the Control object. The caller should not free this object; the Transaction Service retains ownership of it.

Exceptions

Unavailable

Remarks

The Coordinator object can be used to register resources for the transaction associated with the Control. The Unavailable exception is raised if the Control cannot provide the requested object. The TransactionRolledBack standard exception is raised if the Control object represents a transaction that has rolled back.

Example

AIX 390 WIN For AIX, OS/390, and Windows NT:

```
/* C++ example */
#include <CosTransactions.hh>
{
   CosTransactions::Current_ptr my_current;
   CosTransactions::Control_ptr control;
   CosTransactions::Coordinator_ptr coord;
   ...
   // Access the CosTransactions::Current object.
   CORBA::Object_ptr orbCurrentPtr =
     CBSeriesGlobal::orb()->resolve_initial_references("TransactionCurrent");
      my_current = CosTransactions::Current::_narrow(orbCurrentPtr);
   my_current->begin();
   control = my_current->get_control();
   coord = control->get_coordinator();
   ...
}
```

SOLARIS For Solaris:

```
/* C++ example */
#include <CosTransactions.hh>
{
   CosTransactions::Current_ptr my_current;
   CosTransactions::Control_ptr control;
   CosTransactions::Coordinator_ptr coord;
   ...
   // Access the CosTransactions::Current object.
   CORBA::Object_ptr orbCurrentPtr =
     CBSeriesGlobal::orb()->get_current("CosTransactions::Current");
      my_current = CosTransactions::Current::_narrow(orbCurrentPtr);
   my_current->begin();
```

```
   control = my_current->get_control();
   coord = control->get_coordinator();
   ...
}
```

Control::get_terminator

Returns an object that supports the Terminator Interface for the transaction represented by the Control object.

Original interface

CosTransactions::Control Interface

IDL syntax

```
Terminator get_terminator ()
    raises (Unavailable);
```

Return value

Terminator

An object that supports the Terminator interface for the transaction represented by the Control object. It can be used to commit or roll back the transaction. The caller should not free the returned object; the Transaction Service retains ownership of it.

Exceptions

Unavailable

Remarks

The Terminator object can be used to rollback or commit the transaction associated with the Control. The Unavailable exception is raised if the Control cannot provide the requested object. The TransactionRolledBack standard exception is raised if the Control object represents a transaction that has rolled back.

Example

AIX 390 WIN For AIX, OS/390, and Windows NT:

```
/* C++ example */
#include <CosTransactions.hh>
{
   CosTransactions::Current_ptr my_current;
   CosTransactions::Control_ptr control;
   CosTransactions::Terminator_ptr term;
   ...
   // Access the CosTransactions::Current object.
```

```
   CORBA::Object_ptr orbCurrentPtr =
      CBSeriesGlobal::orb()->resolve_initial_references("TransactionCurrent");
   my_current = CosTransactions::Current::_narrow(orbCurrentPtr);
   my_current->begin();
   control = my_current->get_control();
   term = control->get_terminator();
}
```

SOLARIS For Solaris:

```
/* C++ example */
#include <CosTransactions.hh>
{
   CosTransactions::Current_ptr my_current;
   CosTransactions::Control_ptr control;
   CosTransactions::Terminator_ptr term;
   ...
   // Access the CosTransactions::Current object.
   CORBA::Object_ptr orbCurrentPtr =
      CBSeriesGlobal::orb()->get_current("CosTransactions::Current");
   my_current = CosTransactions::Current::_narrow(orbCurrentPtr);
   my_current->begin();
   control = my_current->get_control();
   term = control->get_terminator();
}
```

Coordinator Interface

The Coordinator interface provides common operations for top-level transactions. Participants in a transaction are typically either recoverable objects, or agents of recoverable objects, such as subordinate coordinators. An object supporting the Coordinator interface is implicitly associated with one transaction only.

File stem

CosTransactions

Intended usage

See the *Component Broker Advanced Programming Guide* for an explanation of how to use this interface and its operations to associate a resource object with a transaction.

Exceptions

Inactive
SubtransactionsUnavailable

IDL syntax

```
interface Coordinator
{
    Status get_status();
```

```
    Status get_parent_status();
    Status get_top_level_status();
    boolean is_same_transaction(in Coordinator coord);
    boolean is_related_transaction(in Coordinator coord);
    boolean is_anscestor_transaction(in Coordinator coord);
    boolean is_descendant_transaction(in Coordinator coord);
    boolean is_top_level_transaction();
    unsigned long hash_transaction(in unsigned long maximum);
    unsigned long hash_top_level_tran(in unsigned long maximum);
    RecoveryCoordinator register_resource(in Resource res)
        raises(Inactive);
    void register_synchronization (in Synchronization sync)
        raises(Inactive);
    void register_subtran_aware(in SubtransactionalAwareResource res)
        raises(Inactive, SubtransactionsUnavailable);
    void rollback_only()
        raises(Inactive);
    string get_transaction_name();
    Control create_subtransaction()
        raises(SubtransactionsUnavailable Inactive);
    CosTSInteroperation::PropagationContext get_txcontext ()
        raises(Unavailable);
};
```

Supported operations

Coordinator::create_subtransaction
Coordinator::get_parent_status
Coordinator::get_status
Coordinator::get_top_level_status
Coordinator::get_transaction_name
Coordinator::hash_top_level_transaction
Coordinator::hash_transaction
Coordinator::is_ancestor_transaction
Coordinator::is_descendant_transaction
Coordinator::is_related_transaction
Coordinator::is_top_level_transaction
Coordinator::register_subtran_aware
Coordinator::rollback_only

Coordinator::create_subtransaction

Creates a new subtransaction, whose parent is the transaction associated with the target object. Component Broker does not currently support subtransactions, so if you attempt to invoke this method, the SubtransactionsUnavailable exception will be raised.

Original interface

CosTransactions::Coordinator Interface

IDL syntax

```
Control create_subtransaction()
    raises(SubtransactionsUnavailable, Inactive);
```

Return value

Control

Enables the subtransaction to be terminated, and allows recoverable objects to participate in the transaction.

Exceptions

SubtransactionsUnavailable
Inactive

Remarks

The SubtransactionsUnavailable exception is normally raised if the transaction associated with the target object is completing, or if nested transactions are not supported. Because Component Broker does not currently support subtransactions, any attempt to invoke this method will raise the SubtransactionsUnavailable exception.

The Inactive exception is raised if the target transaction has already been prepared.

Example

```
/* C++ example */

CosTransactions::Control *control, *nest;
CosTransactions::Coordinator *coord;
...
coord = control->get_coordinator();
nest = coord->create_subtransaction();
```

Coordinator::get_parent_status

Returns to a caller the status of the parent transaction of the transaction associated with the target object.

Original interface

CosTransactions::Coordinator Interface

IDL syntax

```
Status get_parent_status();
```

Return value

Status The status of the parent transaction of the transaction associated with the target object.

Remarks

If the transaction associated with the target object is a top-level transaction, this operation is equivalent to the Coordinator::get_status Operation.

If the transaction is not a top-level transaction, this operation returns the status of the parent of the transaction associated with the target object.

Example

```
/* C++ example */
CosTransactions::Coordinator *coord;
CosTransactions::Status *status;
status = coord->get_parent_status();
```

Coordinator::get_status

Returns to a caller the status of the transaction associated with the target object.

Original interface

CosTransactions::Coordinator Interface

IDL syntax

```
Status get_status();
```

Return value

Status The status of the transaction associated with the target object.

Remarks

The operation returns the status of the transaction.

Example

```
/* C++ example */
CosTransactions::Coordinator *coord;
CosTransactions::Status *status;
status = coord->get_status();
```

Coordinator::get_top_level_status

Returns to a caller the status of the top-level ancestor of the transaction associated with the target object.

Original interface

CosTransactions::Coordinator Interface

IDL syntax

```
Status get_top_level_status();
```

Return value

Status The status of the top-level ancestor of the transaction associated with the target object.

Remarks

If the transaction is a top-level transaction, this operation is equivalent to the Coordinator::get_status Operation. If it is not a top level transaction, this operation gets the status of the top-level ancestor.

Example

```
/* C++ example */
CosTransactions::Coordinator *coord;
CosTransactions::Status *status;
status = coord->get_top_level_status();
```

Coordinator::get_transaction_name

Supports debugging by returning a string describing the transaction associated with the target object.

Original interface

CosTransactions::Coordinator Interface

IDL syntax

```
string get_transaction_name();
```

Return value

string A string describing the transaction associated with the target object.

Remarks

Returns a printable string describing the transaction. If there is no transaction associated with the current thread, an empty string is returned.

Example

```
/* C++ example */
CosTransactions::Coordinator *coord;
string name;
name = coord->get_transaction_name();
cout << "Transaction name is " << name << endl;
```

Coordinator::get_txcontext

This operation is not part of the programming model and should not be directly invoked or overridden.

Coordinator::hash_top_level_transaction

Returns a hash value based on the top-level ancestor of the transaction associated with the target object.

Original interface

CosTransactions::Coordinator Interface

IDL syntax

```
unsigned long hash_top_level_tran();
```

Return value

unsigned long

A hash value based on the top-level ancestor of the transaction associated with the target object.

Remarks

Each transaction has a single hash value. Hash values for transactions should be uniformly distributed. This operation is equivalent to the Coordinator::hash_transaction Operation when the transaction associated with the target object is a top-level transaction.

Example

```
/* C++ example */
CosTransactions::Coordinator *coord;
unsigned long hashval;
hashval = coord->hash_top_level_tran();
```

Coordinator::hash_transaction

Returns a hash value based on the transaction associated with the target object.

Original interface

CosTransactions::Coordinator Interface

IDL syntax

```
unsigned long hash_transaction();
```

Return value

unsigned long
A hash value based on the transaction associated with the target object.

Remarks

Each transaction has a single hash value although multiple transactions may share the same value. Hash values for transactions should be uniformly distributed.

Example

```
/* C++ example */
CosTransactions::Coordinator *coord;
unsigned long hashval;
hashval = coord->hash_transaction();
```

Coordinator::is_ancestor_transaction

Determines whether the transaction associated with the target object is an ancestor of the transaction associated with the parameter object.

Original interface

CosTransactions::Coordinator Interface

IDL syntax

```
boolean is_ancestor_transaction(in Coordinator tc);
```

Parameters

tc A pointer to the Coordinator object for a transaction.

Return value

TRUE The transaction associated with the target object is an ancestor of the transaction associated with the parameter object.

FALSE The transaction associated with the target object is not an ancestor of the transaction associated with the parameter object.

Remarks

A transaction is an ancestor of another transaction if, and only if, the transactions are the same, or the first is an ancestor of the parent of the second.

Example

```
/* C++ example */
CosTransactions::Coordinator *c1, *c2;
if( c1->is_ancestor_transaction(c2) )
{
    cout << "c1 is an ancestor of c2" << endl;
}
else
{
    cout << "c1 is not an ancestor of c2" << endl;
}
```

Coordinator::is_descendant_transaction

Determines whether the transaction associated with the target object is a descendant of the transaction associated with the parameter object.

Original interface

CosTransactions::Coordinator Interface

IDL syntax

```
boolean is_descendant_transaction(in Coordinator tc);
```

Parameters

tc A pointer to the Coordinator object for a transaction.

Return value

TRUE The transaction associated with the target object is a descendant of the transaction associated with the parameter object.

FALSE The transaction associated with the target object is not a descendant of the transaction associated with the parameter object.

Remarks

A transaction is an descendant of another transaction if, and only if, the second transaction is an ancestor of the first. For an definition of ancestors of transactions, see the Coordinator::is_ancestor_transaction Operation.

Example

```
/* C++ example */
CosTransactions::Coordinator *c1, *c2;
if( c1->is_descendant_transaction(c2))
{
    cout << "c1 is a descendant of c2" << endl;
}
else
{
    cout << "c1 is not a descendant of c2" << endl;
}
```

Coordinator::is_related_transaction

Determines whether the transaction associated with the target object is related to the transaction associated with the parameter object.

Original interface

CosTransactions::Coordinator Interface

IDL syntax

```
boolean is_related_transaction(in Coordinator tc);
```

Parameters

tc A pointer to the Coordinator object for a transaction.

Return value

TRUE The transaction associated with the target object is related to the transaction associated with the parameter object.

FALSE The transaction associated with the target object is not related to the transaction associated with the parameter object.

Remarks

A transaction is related to another transaction if, and only if, they share a common ancestor transaction.

Example

```
/* C++ example */
CosTransactions::Coordinator *c1, *c2;
if( c1->is_related_transaction(c2) )
{
    cout << "c1 is related to c2" << endl;
}
else
{
    cout << "c1 is not related to c2" << endl;
}
```

Coordinator::is_same_transaction

Determines whether the target object and the parameter object both refer to the same transaction.

Original interface

CosTransactions::Coordinator Interface

IDL syntax

```
boolean is_same_transaction(in Coordinator tc);
```

Parameters

tc A pointer to the Coordinator object for a transaction.

Return value

TRUE The target object and the parameter object both refer to the same transaction.

FALSE The target object and the parameter object do not both refer to the same transaction.

Remarks

Determines whether the target object and the parameter object both refer to the same transaction.

Example

```
/* C++ example */
CosTransactions::Coordinator *c1, *c2;
if( c1->is_same_transaction(c2) )
{
    cout << "c1 represents the same transaction as c2" << endl;
}
```

```
else
{
    cout << "c1 does not represent the same transaction as c2" << endl;
}
```

Coordinator::is_top_level_transaction

Determines whether the transaction associated with the target object is a top-level transaction.

Original interface

CosTransactions::Coordinator Interface

IDL syntax

```
boolean is_top_level_transaction();
```

Parameters

tc A pointer to a Coordinator interface transaction.

Return value

TRUE The transaction associated with the target object is a top-level transaction.

FALSE The transaction associated with the target object is not a top-level transaction.

Remarks

A transaction is a top-level transaction if it has no parent.

Example

```
/* C++ example */
CosTransactions::Coordinator *coord;
if( coord->is_top_level_transaction() )
{
    cout << "Coordinator represents a top-level transaction" << endl;
}
else
{
    cout << "Coordinator represents a subtransaction" << endl;
}
```

Coordinator::register_resource

This operation is not part of the programming model and should not be directly invoked or overridden.

Coordinator::register_subtran_aware

This operation is not part of the programming model and should not be directly invoked or overridden.

Coordinator::register_synchronization

This operation is not part of the programming model and should not be directly invoked or overridden.

Coordinator::rollback_only

Modifies the transaction associated with the target object so that it cannot be committed, but only rolled back.

Original interface

CosTransactions::Coordinator Interface

IDL syntax

```
void rollback_only()
    raises (Inactive);
```

Exceptions

Inactive

Remarks

If the transaction is completing, the Inactive exception is raised.

Example

```
/* C++ example */
CosTransactions::Coordinator *coord;
coord->rollback_only();
```

Current Interface

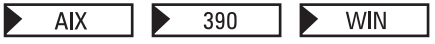
The Current interface defines operations that allow a client of the transaction framework to explicitly manage the association between threads and transactions. It also defines operations that simplify the use of the transaction framework for most applications. These operations can be used to begin and end transactions, and to obtain information about the current transaction.

File stem

CosTransactions

Intended usage

The Current interface defines methods that allow a client of the transaction service to manage the association between thread and transactions. The Current interface provides indirect control of the transaction. It also defines methods that simplify the use of the transaction service. The methods can be used to begin and end transactions, and to obtain information about the current transaction.



On AIX, OS/390, and Windows NT, the Current interface is derived from a CORBA::Current and can be obtained by invoking CORBA::ORB::resolve_initial_references(″TransactionCurrent″) and narrowing the resulting CORBA::Object to a CosTransactions::Current.

On Solaris, the Current interface is derived from a CORBA::Current and can be obtained by invoking CORBA::ORB::get_current(″CosTransactions::Current″) and narrowing the resulting CORBA::Current to a CosTransactions::Current.

Local-only

True

Exceptions

HeuristicHazard
HeuristicMixed
SubtransactionsUnavailable
NoTransaction

IDL syntax

```
interface Current : CORBA::Current
{
    void begin()
        raises(SubtransactionsUnavailable);
    void commit(in boolean report_heuristics)
        raises(NoTransaction, HeuristicMixed,
        HeuristicHazard);
    void rollback()
        raises(NoTransaction);
    void rollback_only()
        raises(NoTransaction);
    Status get_status();
    string get_transaction_name();
    void set_timeout(in unsigned long seconds);
    Control get_control();
```

```
    Control suspend();
    void resume(in Control which)
        raises(InvalidControl);
};
```

Supported operations

Current::begin
Current::commit
Current::get_control
Current::get_status
Current::get_transaction_name
Current::resume
Current::rollback
Current::rollback_only
Current::set_timeout
Current::suspend

Current::begin

Creates a new transaction.

Original interface

CosTransactions::Current Interface

IDL syntax

```
void begin() raises(SubtransactionsUnavailable);
```

Exceptions

SubtransactionsUnavailable

Remarks

The transaction context of the current thread is modified so that the thread is associated with the new transaction.

The SubtransactionsUnavailable exception is raised if the current thread already has an associated transaction and the transaction framework does not support subtransactions, or the current transaction is completing. Component Broker does not currently support subtransactions, so an attempt to begin a transaction on the current thread when a transaction is already associated with the current thread will result in the SubtransactionsUnavailable exception.

The INITIALIZE standard exception is raised if begin is being used for the first time and the Transaction Service cannot be initialized.

Example

AIX 390 WIN For AIX, OS/390, and Windows NT:

```
#include <CosTransactions.hh> // CosTransactions module
...
//Access the CosTransactions::Current object.
CORBA::Object_ptr orbCurrentPtr =
   CBSeriesGlobal::orb()->resolve_initial_references("TransactionCurrent");
CosTransactions::Current_ptr current =
   CosTransactions::Current::_narrow(orbCurrentPtr);
...
// Invoke the begin operation on the CosTransactions::Current object.
current->begin();
...
```

SOLARIS For Solaris:

```
#include <CosTransactions.hh> // CosTransactions module
...
//Access the CosTransactions::Current object.
CORBA::Current_ptr orbCurrentPtr =
   CBSeriesGlobal::orb()->get_current("CosTransactions::Current;
CosTransactions::Current_ptr current =
   CosTransactions::Current::_narrow(orbCurrentPtr);
...
// Invoke the begin operation on the CosTransactions::Current object.
current->begin();
...
```

Current::commit

Completes the current transaction.

Original interface

CosTransactions::Current Interface

IDL syntax

```
void commit(in boolean report_heuristics)
    raises (NoTransaction,HeuristicMixed,HeuristicHazard);
```

Parameters

report_heuristics
Flag indicating whether heuristic reporting is required.

Exceptions

HeuristicHazard
HeuristicMixed
NoTransaction

Remarks

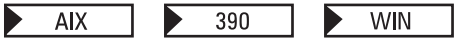
If there is no current transaction, the NoTransaction exception is raised. If the current thread does not have permission to commit the transaction, the standard NO_PERMISSION exception is raised. (The commit operation is restricted to the transaction originator in some circumstances.)

The effect of this operation is equivalent to performing the Terminator::commit Operation in the corresponding Terminator Interface.

The current thread transaction is modified as follows: If the transaction has been begun by a thread (using begin) in the same execution environment, the thread's transaction context is restored to its state prior to the begin request. Otherwise, the thread's transaction context is set to NULL.

If transactions are rolling back for no apparent reason you may be trying to invoke the commit operation on an object with an active exception.

Example



For AIX, OS/390, and Windows NT:

```
 #include <CosTransactions.hh>    // CosTransactions module
...
::CORBA::Boolean    rollback_required = FALSE;
...

//Access the CosTransactions::Current object.

CORBA::Object_ptr orbCurrentPtr =
   CBSeriesGlobal::orb()->resolve_initial_references("TransactionCurrent");
CosTransactions::Current_ptr current =
   CosTransactions::Current::_narrow(orbCurrentPtr);

// Invoke the begin operation on the CosTransactions::Current object.

current->begin();

// Perform work for the transaction and set rollback_required to TRUE if
// an error is detected.

...

// Invoke commit or rollback depending on whether rollback_required is
// set. This must be called within a try...catch structure as the
// transaction service may raise an exception if an error occurs.
```

```
try
{
    if (rollback_required == TRUE)
    {
        current->rollback();
    }
    else // commit required
    {
        current->commit(/* report_heuristics = */ TRUE);
    }
}
catch (CORBA::TRANSACTION_ROLLEDBACK  &exc)
{

    // The application called commit, but the transaction service rolled
    // the transaction back because an error was detected.

    ...
}
catch (CosTransactions::HeuristicMixed  &exc)
{

        // The transaction service has reported that some or all of the
        // resource objects have made a heuristic decision.  This has
        // resulted in heuristic damage.

        ...
}
catch (CosTransactions::HeuristicHazard  &exc)
{

        // The transaction service has reported that not all of the
        // resource objects could participate properly in determining the
        // outcome of the transaction.  There is a possibility of
        // heuristic damage.

        ...
}
catch (CORBA::UserException  &exc)
{
        // Another type of user exception has occurred.
        ...
}
catch (CORBA::SystemException  &exc)
{
        // The application called commit, but the transaction service
        // rolled the transaction back because an error was detected.

        ...
}
catch (...)
{
```

```
        // A general exception has occurred.
        ...
}
    ...
```

SOLARIS For Solaris:

```
 #include <CosTransactions.hh>    // CosTransactions module
...
::CORBA::Boolean    rollback_required = FALSE;
...

//Access the CosTransactions::Current object.

CosTransactions::Current *current = get_CosTransactions_Current();

// Invoke the begin operation on the CosTransactions::Current object.

current->begin();

// Perform work for the transaction and set rollback_required to TRUE if
// an error is detected.

...

// Invoke commit or rollback depending on whether rollback_required is
// set. This must be called within a try...catch structure as the
// transaction service may raise an exception if an error occurs.

try
{
    if (rollback_required == TRUE)
    {
        current->rollback();
    }
    else // commit required
    {
        current->commit(/* report_heuristics = */ TRUE);
    }
}
catch (CORBA::TRANSACTION_ROLLEDBACK  &exc)
{

    // The application called commit, but the transaction service rolled
    // the transaction back because an error was detected.

    ...
}
catch (CosTransactions::HeuristicMixed  &exc)
{

        // The transaction service has reported that some or all of the
        // resource objects have made a heuristic decision.  This has
        // resulted in heuristic damage.
```

```
        ...
}
catch (CosTransactions::HeuristicHazard  &exc)
{

        // The transaction service has reported that not all of the
        // resource objects could participate properly in determining the
        // outcome of the transaction.  There is a possibility of
        // heuristic damage.

        ...
}
catch (CORBA::UserException  &exc)
{
        // Another type of user exception has occurred.
        ...
}
catch (CORBA::SystemException  &exc)
{
        // The application called commit, but the transaction service
        // rolled the transaction back because an error was detected.

        ...
}
catch (...)
{
        // A general exception has occurred.
        ...
}
    ...
```

Current::get_control

Returns an object representing the transaction context currently associated with the current thread.

Original interface

CosTransactions::Current Interface

IDL syntax

```
Control get_control();
```

Return value

Control

Represents the transaction context currently associated with the current thread. The caller should not free the returned object; the Transaction Service retains ownership of it.

Remarks

If the current thread is not associated with a transaction, a NULL object reference is returned.

The Control object returned can be given to the Current::resume Operation to reestablish this transaction context in the same thread or a different thread.

Example

AIX 390 WIN For AIX, OS/390, and Windows NT:

```
/* C++ example */
#include <CosTransactions.hh>
{
   CosTransactions::Current_ptr my_current;
   CosTransactions::Control_ptr control;
   ...
   // Access the CosTransactions::Current object.
   CORBA::Object_ptr orbCurrentPtr =
     CBSeriesGlobal::orb()->resolve_initial_references("TransactionCurrent");
   my_current = CosTransactions::Current::_narrow(orbCurrentPtr);
   control = my_current->get_control();
   ...
}
```

SOLARIS For Solaris:

```
/* C++ example */
#include <CosTransactions.hh>
{
    CosTransactions::Current_ptr my_current;
    CosTransactions::Control_ptr control;
    ...
    // Access the CosTransactions::Current object.
    CORBA::Current_ptr orbCurrentPtr =
        CBSeriesGlobal::orb()->get_current("CosTransactions::Current");
    my_current = CosTransactions::Current::_narrow(orbCurrentPtr);
    control = my_current->get_control();
    ...
}
```

Current::get_status

Determines the status of the current transaction.

Original interface

CosTransactions::Current Interface

IDL syntax

```
Status get_status();
```

Return value

Status The status of the current transaction.

Exceptions

Remarks

If there is no transaction associated with the current thread, the StatusNoTransaction value is returned.

The effect of this request is equivalent to performing the get_status Operation in the corresponding Coordinator Interface.

Example

AIX 390 WIN For AIX, OS/390, and Windows NT:

```
/* C++ example */
#include <CosTransactions.hh>
{
   CosTransactions::Current_ptr my_current;
   CosTransactions::Status_ptr status;
   ...
   // Access the CosTransactions::Current object.
   CORBA::Object_ptr orbCurrentPtr =
     CBSeriesGlobal::orb()->resolve_initial_references("TransactionCurrent");
   my_current = CosTransactions::Current::_narrow(orbCurrentPtr);
   status = my_current->get_status();
   ...
}
```

SOLARIS For Solaris:

```
/* C++ example */
#include <CosTransactions.hh>
{
    CosTransactions::Current_ptr my_current;
    CosTransactions::Status_ptr status;
    ...
    // Access the CosTransactions::Current object.
    CORBA::Current_ptr orbCurrentPtr =
        CBSeriesGlobal::orb()->get_current("CosTransactions::Current");
```

```
    my_current = CosTransactions::Current::_narrow(orbCurrentPtr);
    status = my_current->get_status();
    ...
}
```

Current::get_transaction_name

Supports debugging by returning a string describing the transaction.

Original interface

CosTransactions::Current Interface

IDL syntax

```
string get_transaction_name();
```

Return value

string A printable string describing the transaction.

Remarks

If there is no transaction associated with the current thread, an empty string is returned.

The effect of this request is equivalent to performing the get_transaction_name Operation in the corresponding Coordinator Interface.

Example

AIX 390 WIN For AIX, OS/390, and Windows NT:

```
/* C++ example */
#include <CosTransactions.hh>
{
   CosTransactions::Current_ptr my_current;
   string name;
   ...
   // Access the CosTransactions::Current object.
   CORBA::Object_ptr orbCurrentPtr =
     CBSeriesGlobal::orb()->resolve_initial_references("TransactionCurrent");
   my_current = CosTransactions::Current::_narrow(orbCurrentPtr);
   name = my_current->get_transaction_name();
   cout << "Current transaction name is " << name << endl;
   ...
}
```

SOLARIS For Solaris:

```
/* C++ example */
#include <CosTransactions.hh>
{
   CosTransactions::Current_ptr my_current;
   string name;
   ...
   // Access the CosTransactions::Current object.
   CORBA::Object_ptr orbCurrentPtr =
     CBSeriesGlobal::orb()->get_current("CosTransactions::Current");
   my_current = CosTransactions::Current::_narrow(orbCurrentPtr);
   name = my_current->get_transaction_name();
   cout << "Current transaction name is " << name << endl;
   ...
}
```

Current::resume

Associates the current thread with the supplied transaction context (in place on any previously associated transaction context).

Original interface

CosTransactions::Current Interface

IDL syntax

```
void resume(in Control which)
    raises(InvalidControl);
```

Parameters

which The Control object representing the transaction context.

Exceptions

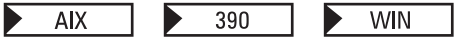
InvalidControl

Remarks

If the parameter is not valid in the current execution context (that is, it is an object reference with an invalid value), the current thread is associated with no transaction.

The INITIALIZE standard exception is raised if resume is being used for the first time and the Transaction Service cannot be initialized.

Example



For AIX, OS/390, and Windows NT:

```
#include <CosTransactions.hh> // CosTransactions module
...
CosTransactions::Control_ptr control = control_parameter;
...
//Access the CosTransactions::Current object.
CORBA::Object_ptr orbCurrentPtr =
   CBSeriesGlobal::orb()->resolve_initial_references("TransactionCurrent");
CosTransactions::Current_ptr current =
   CosTransactions::Current::_narrow(orbCurrentPtr);
...
// Resume the association between the transaction and the thread.
try
{
    current->resume(control);
}
catch(CosTransactions::InvalidControl)
{
    cout << "Error: control object passed to resume was invalid" << endl;
}
```

SOLARIS For Solaris:

```
#include <CosTransactions.hh> // CosTransactions module
...
CosTransactions::Control_ptr control = control_parameter;
...
//Access the CosTransactions::Current object.
CosTransactions::Current *current = get_CosTransactions_Current();
...
// Resume the association between the transaction and the thread.
try
{
    current->resume(control);
}
catch(CosTransactions::InvalidControl)
{
    cout << "Error: control object passed to resume was invalid" << endl;
}
```

Current::rollback

Rolls back the transaction associated with the current thread.

Original interface

CosTransactions::Current Interface

IDL syntax

```
void rollback() raises(NoTransaction);
```

Exceptions

NoTransaction

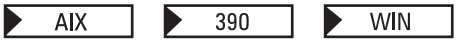
Remarks

If there is no transaction associated with the current thread, the NoTransaction exception is raised. If the current thread does not have permission to rollback the transaction, the standard NO_PERMISSION exception is raised.

The effect of this request is equivalent to performing the rollback Operation in the corresponding Terminator Interface.

The current thread transaction is modified as follows: If the transaction has been begun by a thread (using begin) in the same execution environment, the thread's transaction context is restored to its state prior to the begin request. Otherwise, the thread's transaction context is set to NULL.

Example



For AIX, OS/390, and Windows NT:

```
#include <CosTransactions.hh>    // CosTransactions module
...
::CORBA::Boolean    rollback_required = FALSE;
...

//Access the CosTransactions::Current object.

CORBA::Object_ptr orbCurrentPtr =
   CBSeriesGlobal::orb()->resolve_initial_references("TransactionCurrent");
CosTransactions::Current_ptr current =
   CosTransactions::Current::_narrow(orbCurrentPtr);

// Invoke the begin operation on the CosTransactions::Current object.

current->begin();

// Perform work for the transaction and set rollback_required to TRUE if
// an error is detected.

...

// Invoke commit or rollback depending on whether rollback_required is
// set. This must be called within a try...catch structure as the
// transaction service may raise an exception if an error occurs.

try
{
    if (rollback_required == TRUE)
    {
        current->rollback();
    }
```

```
    else // commit required
    {
        current->commit(/* report_heuristics = */ TRUE);
    }
}
catch (CORBA::TRANSACTION_ROLLEDBACK  &exc)
{

    // The application called commit, but the transaction service rolled
    // the transaction back because an error was detected.

    ...
}
catch (CosTransactions::HeuristicMixed  &exc)
{

        // The transaction service has reported that some or all of the
        // resourceobjects have made a heuristic decision.  This has
        // resulted in heuristic damage.

        ...
}
catch (CosTransactions::HeuristicHazard  &exc)
{

        // The transaction service has reported that not all of the
        // resource objects could participate properly in determining the
        // outcome of the transaction.  There is a possibility of
        // heuristic damage.

        ...
}
catch (CORBA::UserException  &exc)
{
        // Another type of user exception has occurred.
        ...
}
catch (CORBA::SystemException  &exc)
{
        // The application called commit, but the transaction service
        // rolledthe transaction back because an error was detected.

        ...
}
catch (...)
{
        // A general exception has occurred.
        ...
}
    ...
```

SOLARIS For Solaris:

```
#include <CosTransactions.hh>    // CosTransactions module
...
::CORBA::Boolean    rollback_required = FALSE;
...

//Access the CosTransactions::Current object.

CosTransactions::Current *current = get_CosTransactions_Current();

// Invoke the begin operation on the CosTransactions::Current object.

current->begin();

// Perform work for the transaction and set rollback_required to TRUE if
// an error is detected.

...

// Invoke commit or rollback depending on whether rollback_required is
// set. This must be called within a try...catch structure as the
// transaction service may raise an exception if an error occurs.

try
{
    if (rollback_required == TRUE)
    {
        current->rollback();
    }
    else // commit required
    {
        current->commit(/* report_heuristics = */ TRUE);
    }
}
catch (CORBA::TRANSACTION_ROLLEDBACK  &exc)
{

    // The application called commit, but the transaction service rolled
    // the transaction back because an error was detected.

    ...
}
catch (CosTransactions::HeuristicMixed  &exc)
{

        // The transaction service has reported that some or all of the
        // resourceobjects have made a heuristic decision.  This has
        // resulted in heuristic damage.

        ...
}
catch (CosTransactions::HeuristicHazard  &exc)
{

        // The transaction service has reported that not all of the
        // resource objects could participate properly in determining the
```

```
        // outcome of the transaction.  There is a possibility of
        // heuristic damage.

        ...
}
catch (CORBA::UserException  &exc)
{
        // Another type of user exception has occurred.
        ...
}
catch (CORBA::SystemException  &exc)
{
        // The application called commit, but the transaction service
        // rolledthe transaction back because an error was detected.

        ...
}
catch (...)
{
        // A general exception has occurred.
        ...
}
    ...
```

Current::rollback_only

Modifies the transaction such that it cannot be committed, but can only be rolled back.

Original interface

CosTransactions::Current Interface

IDL syntax

```
void rollback_only() raises(NoTransaction);
```

Exceptions

NoTransaction

Remarks

If there is no transaction associated with the current thread, the NoTransaction exception is raised.

The effect of this request is equivalent to performing the rollback_only Operation in the correspondingCoordinator Interface.

Example

AIX 390 WIN For AIX, OS/390, and Windows NT:

```
#include <CosTransactions.hh> // CosTransactions module
...
//Access the CosTransactions::Current object.
CORBA::Object_ptr orbCurrentPtr =
   CBSeriesGlobal::orb()->resolve_initial_references("TransactionCurrent");
CosTransactions::Current_ptr current =
   CosTransactions::Current::_narrow(orbCurrentPtr);
// Invoke the rollback_only operation on the CosTransactions::Current object.
current->rollback_only();
...
```

SOLARIS For Solaris:

```
#include <CosTransactions.hh> // CosTransactions module
...
//Access the CosTransactions::Current object.
CORBA::Object_ptr orbCurrentPtr =
   CBSeriesGlobal::orb()->resolve_initial_references("TransactionCurrent");
CosTransactions::Current *current = get_CosTransactions_Current();
// Invoke the rollback_only operation on the CosTransactions::Current object.
current->rollback_only();
...
```

Current::set_timeout

Sets the timeout value to be used for all subsequent transactions.

Original interface

CosTransactions::Current Interface

IDL syntax

```
void set_timeout(in unsigned long seconds);
```

Parameters

seconds
The value of the time limit in seconds.

Remarks

Subsequent transactions created are subject to being rolled back if they do not complete within the time limit specified on this parameter. The default value for the time limit is platform dependent. If the parameter is zero, there is no application-specific time limit.

The INITIALIZE standard exception is raised if set_timeout is being used for the first time and the Transaction Service cannot be initialized.

Example

AIX 390 WIN For AIX, OS/390, and Windows NT:

```
#include <CosTransactions.hh> // CosTransactions module
...
//Access the CosTransactions::Current object.
CORBA::Object_ptr orbCurrentPtr =
   CBSeriesGlobal::orb()->resolve_initial_references("TransactionCurrent");
CosTransactions::Current_ptr current =
   CosTransactions::Current::_narrow(orbCurrentPtr);
// Invoke the set_timeout operation on the CosTransactions::Current object.
current->set_timeout(60 /* seconds */);
...
// Start a transaction
...
```

SOLARIS For Solaris:

```
#include <CosTransactions.hh> // CosTransactions module
...
//Access the CosTransactions::Current object.
CORBA::Object_ptr orbCurrentPtr =
   CBSeriesGlobal::orb()->resolve_initial_references("TransactionCurrent");
CosTransactions::Current *current = get_CosTransactions_Current();
// Invoke the set_timeout operation on the CosTransactions::Current object.
current->set_timeout(60 /* seconds */);
...
// Start a transaction
...
```

Current::suspend

Returns an object that represents the transaction context currently associated with the current thread, and disassociates the currently associated transaction context from the current thread.

Original interface

CosTransactions::Current Interface

IDL syntax

```
Control suspend();
```

Parameters

Control

Represents the transaction context currently associated with the current thread. The caller should not free the returned object; the Transaction Service retains ownership of it.

Remarks

If there is no current transaction, a NULL reference is returned.

This object can be given to the resume operation to reestablish this context in the same, or a different, thread within the same server process.

Example

AIX 390 WIN For AIX, OS/390, and Windows NT:

```
#include <CosTransactions.hh> // CosTransactions module
...
CosTransactions::Control_ptr control = NULL;
...
//Access the CosTransactions::Current object.
CORBA::Object_ptr orbCurrentPtr =
   CBSeriesGlobal::orb()->resolve_initial_references("TransactionCurrent");
CosTransactions::Current_ptr current =
   CosTransactions::Current::_narrow(orbCurrentPtr);
...
// Invoke the begin operation on the CosTransactions::Current object.
current->begin();
...
// Suspend the association between the transaction and the thread.
control = current->suspend();
if (!control)
{
    // There was no transaction associated with this thread prior to the
    // suspend. Perform appropriate action.
    cout << "Error: No transaction prior to suspend" << endl;
}
```

SOLARIS For Solaris:

```
#include <CosTransactions.hh> // CosTransactions module
...
CosTransactions::Control_ptr control = NULL;
...
//Access the CosTransactions::Current object.
CosTransactions::Current *current = get_CosTransactions_Current();
...
// Invoke the begin operation on the CosTransactions::Current object.
current->begin();
...
// Suspend the association between the transaction and the thread.
```

```
control = current->suspend();
if (!control)
{
    // There was no transaction associated with this thread prior to the
    // suspend. Perform appropriate action.
    cout << "Error: No transaction prior to suspend" << endl;
}
```

RecoveryCoordinator Interface

This interface is not part of the programming model and should not be directly invoked or overridden.

Resource Interface

This interface is not part of the programming model and should not be directly invoked or overridden.

Synchronization Interface

The Synchronization interface defines the operations invoked by the Transaction Service during transaction completion on each resource registered using the register_synchronization operation of the Coordinator Interface.

File name

CosTransactions

IDL syntax

```
interface Synchronization : TransactionalObject
{
    void before_completion();
    void after_completion ( in Status status);
};
```

Supported operations

Synchronization::after_completion
Synchronization::before_completion

Synchronization::after_completion

Informs the target object that the transaction that it represents has been completed. The current status of the transaction (as determined by the register_synchronization operation of the Coordinator Interface) is provided as input.

Original interface

CosTransactions::Synchronization Interface

IDL syntax

```
void after_completion(in Status status);
```

Parameters

status The current status of the transaction, as determined by the Coordinator::register_synchronization operation.

Remarks

Called from within the Transaction Service.

Synchronization::before_completion

Informs the target object that the transaction that it represents is about to be completed. The target object can perform transactional work before it returns.

Original interface

CosTransactions::Synchronization Interface

IDL syntax

```
void before_completion();
```

Remarks

Called from within the Transaction Service.

Terminator Interface

Defines operations to complete a transaction, either by requesting commitment or demanding rollback. Typically, these operations are used by the transaction originator. An object that supports the Terminator interface is implicitly associated with one transaction only.

File stem

CosTransactions

Exceptions

HeuristicHazard
HeuristicMixed

IDL syntax

```
interface Terminator
{
    void commit(in boolean report_heuristics)
        raises(HeuristicMixed,
        HeuristicHazard);
    void rollback();
};
```

Supported operations

Terminator::commit
Terminator::rollback

Terminator::commit

Requests that the transaction be committed.

Original interface

CosTransactions::Terminator Interface

IDL syntax

```
void commit(in boolean report_heuristics)
    raises(HeuristicMixed, HeuristicHazard);
```

Parameters

report_heuristics
Flag indicating whether heuristic reporting is required.

Exceptions

HeuristicHazard
HeuristicMixed

Remarks

If the transaction has not been marked as rollback-only, and all the participants in the transaction agree to commit, the transaction is committed, and the operation terminates normally. Otherwise, the transaction is rolled back and the standard TransactionRolledBack exception is raised.

If the report_heuristics parameter is true, the transaction framework reports inconsistent or possibly-inconsistent outcomes using the HeuristicMixed or HeuristicHazard exceptions.

The commit operation can rollback the transaction if there are existing or potential activities associated with the transaction that have not completed.

When a top-level transaction is committed, all changes to transactional objects made in the scope of this transaction are made permanent and visible to other transactions or clients.

Note that the suspend operation of the CosTransactions::Current Interface must be used to suspend an active transaction before the commit operation of the Terminator interface is used to commit the transaction.

Example

```
/* C++ example */
{
   try
   {
     CosTransactions::Control_var control;
     CosTransactions::Terminator_var term;

     .....

     control = current->suspend();
     term = control->get_terminator();
     term->commit(TRUE);
   }
   catch ( .........
}
```

Terminator::rollback

Demands that the transaction be rolled back.

Original interface

CosTransactions::Terminator Interface

IDL syntax

```
void rollback();
```

Remarks

When a transaction is rolled back, all changes to transactional objects made in the scope of this transaction (including changes made by descendant transactions) are rolled back. All resources locked by the transaction are made available to other transactions as appropriate to the degree of isolation enforced by the resources.

Note that the suspend operation of the CosTransactions::Current Interface must be used to suspend an active transaction before the rollback operation of the Terminator interface is used to rollback the transaction.

Example

```
/* C++ example */
{
   try
   {
     CosTransactions::Control_var control;
     CosTransactions::Terminator_var term;

     .....

     control = current->suspend();
     term = control->get_terminator();
     term->rollback();
   }
   catch ( .........
}
```

TransactionalObject Interface

The TransactionalObject interface is used by an object to indicate that it is transactional. By inheriting from the TransactionalObject interface, an object indicates that it wants the transaction context associated with the client thread to be propagated on requests to the object.

The TransactionalObject interface defines no operations. It is simply a marker.

File stem

CosTransactions

IDL syntax

```
interface TransactionalObject{};
```

TransactionFactory Interface

This interface is not part of the programming model and should not be directly invoked or overridden.

Chapter 17. IBOIMExtLocal in the Managed Object Framework

The other modules in the Managed Object Framework are:

- CBSeriesGlobal
- IBOIMExtLocalToServer
- IBOIMInstanceManagerFriendQOS
- IBOIMManagedObjectFriendQOS
- IBOIMServiceFriendQOS
- ICollectionsBase
- IManagedAdvancedClient
- IManagedAdvancedServer
- IManagedClient
- IManagedCollections
- IManagedLocal
- IManagedServer
- IRDBIMExtLocalToServer

IBOIMExtLocal Module

Provides the interfaces used for managed objects that are based on UUIDs as their primary keys.

Note: These interfaces are not available in the OS/390 Component Broker client environment. Nor are the keys produced by these interfaces in a Component Broker workstation environment usable in the OS/390 Component Broker server environment. Uniqueness of the generated UUID-based keys can only be guaranteed if the keys are generated on the platform in which they will be used (in other words, workstation keys on workstation platforms and OS/390 keys on an OS/390 platform). For these reasons, the recommended use of these interfaces should be within a ″create″ method on a specialized home.

The IManagedAdvancedServer UUID support should be used in a specialized home to provide a solution that is portable on both OS/390 Component Broker and Component Broker workstation.

File name

IBOIMExtLocal.idl

Intended usage

The IBOIMExtLocal module contains the interfaces needed to have managed objects based on the UUID key.

Interfaces

IBOIMExtLocal::“IUUIDPrimaryKey Interface” on page 526

IBOIMExtLocal::“IUUIDCopyHelperBase Interface” on page 526

IUUIDCopyHelperBase Interface

This interface is intended to be a base class for the copy helpers of those managed objects that are based on the UUID (are transient).

File name

IBOIMExtLocal.idl

Intended usage

This interface is intended to be a base class for the copy helpers of those managed objects that are based on the UUID (are transient). Initializing the UUID portion of this interface is expected to occur during the construction of the object.

IDL syntax

```
interface IUUIDCopyHelperBase :  IManagedLocal::INonManageable
{
       ByteString getUuid();
};
```

Supported operations

IUUIDCopyHelperBase::getUuid

IUUIDCopyHelperBase::getUuid

This operation returns the value of the unique identifier of the primary key.

Return value

This return value is a byte string that can be used as the key attributes in the data object. The return value of the operation is not valid to be used on the fromString call on a primary key.

IUUIDPrimaryKey Interface

This is a default for primary key of managed objects that are transient.

File name

IBOIMExtLocal.idl

Intended usage

An new initializer is provided for the primary key. The primary key must be initialized by either the fromString method (supporting the inherited interface) or with the new initializer method “generate”. Also a way to get at the one attribute that is the primary key is provided.

To guarantee uniqueness across Component Broker platforms, the use of this interface should be within a “create” method on a specialized home.

IDL syntax

```
interface IUUIDPrimaryKey : IManagedLocal::IPrimaryKey
   {
    ByteString getUuid();
    void generate()
                   raises(IBOIMException::IUuidSetFailed);
   };
```

Supported operations

IUUIDPrimaryKey::generate
IUUIDPrimaryKey::getUuid

IUUIDPrimaryKey::generate

This operation is an initializer of the primary key. If a new primary key is needed to create an object then the generate method is used to initialize the primary key with a new unique value.

Exceptions

IBOIMException::IUuidSetFailed

This exception is raised if a unique identifier cannot be generated. The causes for this type of failure are the primary key had already been initialized or the Component Broker runtime has had some problem. The activity log should include any information if the problem is in the Component Broker runtime.

IUUIDPrimaryKey::getUuid

This operation returns the value of the unique identifier of the primary key.

Return value

This return value is a byte string that can be used as the key attributes in the data object. The return value of the operation is not valid to be used on the fromString call.

Chapter 18. IBOIMExtLocalToServer in the Managed Object Framework

The other modules in the Managed Object Framework are:

- CBSeriesGlobal
- IBOIMExtLocal
- IBOIMInstanceManagerFriendQOS
- IBOIMManagedObjectFriendQOS
- IBOIMServiceFriendQOS
- ICollectionsBase
- IManagedAdvancedClient
- IManagedAdvancedServer
- IManagedClient
- IManagedCollections
- IManagedLocal
- IManagedServer
- IRDBIMExtLocalToServer

IBOIMExtLocalToServer Module

Provides the interfaces that are required to be supported by components that are configured to BOIM application adaptor containers.

Note: The interface names and file names for OS/390 Component Broker are different from those on a Component Broker workstation. On OS/390 the names are IBOIM390ExtLocalToServer, on a workstation they are IBOIMExtLocalToServer. For simplicity the interface is described using the Component Broker workstation name.

File name

IBOIMExtLocalToServer.idl

Intended usage

The IBOIMExtLocalToServer module contains the interfaces needed to have managed objects configured to a BOIM application adaptor container.

Interfaces

IBOIMExtLocalToServer::IDataObject Interface
IBOIMExtLocalToServer::IHomeAwareDataObject Interface
IBOIMExtLocalToServer::IQueryableDataObject Interface

IDataObject Interface

The dataobject interface is an extension of the IManagedServer::IDataObject interface. This object is used to communicate to backing stores (which are also known as two level stores).

File name

IBOIMExtLocalToServer.idl

Supported operations

IDataObject::IDataObject::del
IDataObject::IDataObject::externalizeKeyAttributes
IDataObject::IDataObject::insert
IDataObject::IDataObject::internalizeKeyAttributes
IDataObject::IDataObject::retrieve
IDataObject::IDataObject::update
IDataObject::IDataObject::verifyKey

IDataObject::del

This operation is used as a part of the synchronization of the essential state of a data object with the backend. The implementation of this method is provided as a part of the application supplied code. This implementation should remove an already existing record from the backend.

Original interface

IDataObject Interface

IDL syntax

```
void del()
   raises(IBOIMException::IDataKeyNotFound,
          IBOIMException::IDataObjectFailed);
```

Exceptions

IDataKeyNotFound
This exception is raised if the data to be removed from the backend does not exist.

IDataObjectFailed
This exception is raised if the backend has been unsuccessful in deleting its data for any reason other than the data not existing.

IDataObject::externalizeKeyAttributes

This operation is used by the data object to supply its key attributes. The key attributes of a data object is the information required by a data object to locate the essential state in the backend. The implementation of this method is provided as a part of the application supplied code. This implementation should put into the KeyComponent any information needed to locate the essential state in the backend.

Original interface

IDataObject Interface

IDL syntax

```
void externalizeKeyAttributes(inout IIMFLocalToServer::IKeyComponent keyComp)
   raises(IBOIMException::IExternalizeKeyAttributesFailed);
```

Parameters

KeyComp
The object to be used by the data object to externalize the key attibutes.

Exceptions

IExternalizeKeyAttributesFailed
This exception is raised to indicate that the KeyComponent does not contain the key attributes and the problem cannot be expressed in terms of a CORBA system exception.

IDataObject::insert

This operation is used as a part of the synchronization of the essential state of a data object with the backend. The implementation of this method is provided as a part of the application supplied code. The implementation should insert the data into the backend if a record of the same key does not already exist.

Original interface

IDataObject Interface

IDL syntax

```
void insert()
   raises(IBOIMException::IDataKeyAlreadyExists,
          IBOIMException::IDataObjectFailed);
```

Exceptions

IDataKeyAlreadyExists
This exception is raised if some data already exists with the same key of the data being inserted into the backend.

IDataObjectFailed
This exception is raised if the backend has been unsuccessful in inserting its data for any reason other than the data already existing.

IDataObject::internalizeKeyAttributes

This operation is used by the data object to obtain its key attributes. The key attributes made available in the KeyComponent are those that were supplied in the externalizeKeyAttributes method. This method is the initialization method of the data object when the object is being reactivated because a client has used an object reference. The implementation of this method is provided as a part of the application supplied code. This implementation should retrieve from the KeyComponent the information needed to locate the essential state in the backend

Original interface

IDataObject Interface

IDL syntax

```
void internalizeKeyAttributes(in IIMFLocalToServer::IKeyComponent keyComp)
   raises(IBOIMException::IInternalizeKeyAttribututesFailed,
          IBOIMException::IDataKeyNotFound,
          IBOIMException::IDataObjectFailed);
```

Parameters

KeyComp
The object to be used by the data object to retrieve the key attibutes.

Exceptions

IDataKeyNotFound
This exception is raised to indicate that an attempt was made to locate the essential state and it was not found.

IDataObjectFailed
This exception is raised to indicate that an attempt was made to locate the essential state in the backend and the attempt was unsuccessful for any reason other than the data not existing.

IInternalizeKeyAttribututesFailed
This exception is raised to indicate that the data object was either unsuccessful or the data object expects that it will be unsuccessful in being able to use the attributes in the KeyComponent to locate the essential state.

IDataObject::retrieve

This operation is used as a part of the synchronization of the essential state of a data object with the backend. The implementation of this method is provided as a part of the application supplied code. This implementation should retrieve the data from the back end.

Original interface

IDataObject Interface

IDL syntax

```
void retrieve()
   raises(IBOIMException::IDataKeyNotFound,
   IBOIMException::IDataObjectFailed);
```

Exceptions

IDataKeyNotFound
This exception is raised if the data cannot be located in the backend.

IDataObjectFailed
This exception is raised if the backend has been unsuccessful at retrieving its data for any reason other than the data not existing.

IDataObject::update

This operation is used as a part of the synchronization of the essential state of a data object with the backend. The implementation of this method is provided as a part of the application supplied code. This implementation should update the data in the backend with that which is held by the objects.

Original interface

IDataObject Interface

IDL syntax

```
void update()
   raises(IBOIMException::IDataKeyNotFound,
          IBOIMException::IDataObjectFailed);
```

Exceptions

IDataKeyNotFound
This exception is raised if the data to be updated cannot be located in the backend.

IDataObjectFailed
This exception is raised if the backend has been unsuccessful at updating its data for any reason other than the data not existing.

IDataObject::verifyKey

This operation is used to describes the current state of the key attributes. This method is used by the framework to determine if the data object has been initialized sufficiently to be able to locate the persistent state. The implementation of this method is provided as a part of the application supplied code. This implementation should determine the state of the key and report its condition (valid or not valid).

Original interface

IDataObject Interface

IDL syntax

```
boolean verifyKey();
```

Return value

boolean
This is returned true if the key attributes describe the location of the essential state. This is returned false if the key attributes are not initialized sufficiently to allow the data object to find the essential state.

IHomeAwareDataObject Interface

The IHomeAwareDataObject is an extension of the IBOIMExtLocalToServer::IDataObjectBase interface. It provides an additional interface which is required in Component Broker to add quality of services to the managed object. The customer code does not call or implement any methods on this interface.

File name

IBOIMExtLocalToServer.idl

Supported operations

IHomeAwareDataObject::IHomeAwareDataObject::setHome

IHomeAwareDataObject::setHome

This operation is used as a part of the of the initialization of a data object that needs to be able to communicate with its home. The implementation of this method is provided as a part of the application supplied code. This implementation should keep track of the home provided to be able to use it when it is needed by other methods.

Original interface

IHomeAwareDataObject Interface

IDL syntax

```
void setHome (in IManagedClient::IHome theHome)
   raises(IBOIMException::IInvalidHomeType);
```

Parameters

theHome
This object is the home of the managed object that will use the data object.

Exceptions

IInvalidHomeType
This exception is raised to indicate that the type of the home provided to the data object was not of the type expected.

IQueryableDataObject Interface

If a managed object is to participate in query, the data object associated with the managed object must also support the IQueryableDataObject interface.

File name

IBOIMExtLocalToServer.idl

Supported operations

IQueryableDataObject::IQueryableDataObject::internalizeData

IQueryableDataObject::internalizeData

This operation is used as a part of the synchronization of the essential state of a data object with the backend. The implementation of this method is provided as a part of the application supplied code. This method is used as an alternative to the retrieve method to the data object. However, contrary to the retrieve method, the data for the object has been obtained through some other means external to the data object (like the query service). This implementation should take the data from the dataSequence and treat it as though it had been retrieved by the data object.

Original interface

IQueryableDataObject Interface

IDL syntax

```
void internalizeData(in IBOIMLocalToServerMetadata::dataSequence dataSeq);
```

Parameters

dataSeq

This object contains the essential state of the data object that has been obtained through some means other than the data object.

Chapter 19. IBOIMInstanceManagerFriendQOS in the Managed Object Framework

The other modules in the Managed Object Framework are:

- CBSeriesGlobal
- IBOIMExtLocal
- IBOIMExtLocalToServer
- IBOIMManagedObjectFriendQOS
- IBOIMServiceFriendQOS
- ICollectionsBase
- IManagedAdvancedClient
- IManagedAdvancedServer
- IManagedClient
- IManagedCollections
- IManagedLocal
- IManagedServer
- IRDBIMExtLocalToServer

IBOIMInstanceManagerFriendQOS Module

This module defines the interfaces that are required to be supported by the managed objects that are configured to BOIM application adaptor containers. These interfaces are required in order for the BOIM application adaptor to provide the quality of service it has claimed. Some definitions of quality of service define the interface that is supported, while others define the behavior of the managed objects. The interfaces defined in this module are divided according to these definitions; that is, some interfaces are required because a quality of service defines some set of interfaces that are supported, while other interfaces are required because of behavior that is required. In addition to the interfaces which deal with quality of service, there are also some interfaces which are used by the application adaptor to help manage the managed object assembly.

Note: The interface names and file names are different on OS/390 Component Broker and a Component Broker workstation. On OS/390 the names are IBOIM390InstanceManagerFriendQOS, on a workstation they are IBOIMInstanceManagerFriendQOS. For simplicity the interface is described using the Component Broker workstation name.

File name

IBOIMInstanceManagerFriendQOS.idl

Intended usage

These interfaces are used by the BOIM application adaptor and object services to communicate with the managed object. The managed object provides an implementation for them.

Interfaces

IBOIMInstanceManagerFriendQOS::IMForMixinAndMO Interface
IBOIMInstanceManagerFriendQOS::IMLocalFriend Interface
IBOIMInstanceManagerFriendQOS::IMMixin Interface
IBOIMInstanceManagerFriendQOS::IMOnlyForMO Interface

IMForMixinAndMO Interface

This interface defines a set of methods that both the MO and the mixin support. The MO supports the interface so that clients and object services can use them. The mixin supports the interface so that the MO can delegate the implementation to the mixin.

File name

IBOIMInstanceManagerFriendQOS.idl

Intended usage

This interface is intended to help define the quality of service which is supported by a managed object that is configured to a BOIM container. This quality of service is provided when each managed object supports the interface with an implementation which delegates the implementation of the function to the mixin. Each managed object is expected to provide an implemenation for all methods defined on the interface. The implementation should delegate to the mixin.

IDL syntax

```
interface IMForMixinAndMO :
     IBOIMManagedObjectQOS::IMMixin,
     IBOIMServiceFriendQOS::IMServiceMixin,
     IBOIMManagedObjectFriendQOS::IMMixin,
     IMLocalFriend

  {
  };
```

IMLocalFriend Interface

This interface defines a set of interfaces that the managed objects need to support if they are configured to a BOIM container. The interfaces which combine to make this interface are needed by the BOIM application adaptor so that it can provide the QOS behavior defined to its containers. There is no explanation of how they are used to provide the behavior.

File name

IBOIMInstanceManagerFriendQOS.idl

Intended usage

This interface is intended to be restricted to developers of the Component Broker product. Each managed object is expected to provide an implemenation for all the methods defined on the interface. The implementation should delegate to the mixin.

IMMixin Interface

This interface is defined to provide an argument to be used with the IMOnlyForMO interface.

File name

IBOIMInstanceManagerFriendQOS.idl

Intended usage

This interface is intended to be a base class so that the MO can be told about its mixin.

IMOnlyForMO Interface

These are BOIM specific methods which must be implemented directly in each managed object and cannot be delegated to an instance manager supplied implementation. These methods allow the managed object to function within the Managed Object Framework and are pertinent to allowing the managed object to delegate managed object methods to the instance manager.

File name

IBOIMInstanceManagerFriendQOS.idl

Intended usage

This interface is intended to be called by the application adaptor and implemented by the MO. The methods are called as part of the assembly process of the managed object assembly.

IDL syntax

```
   interface IMOnlyForMO :
      IIMFInstanceManagerFriendQOS::IMOnlyForMO
   {

    void setMixin (in IIMFLocalToServer::IMMixinForDelegatingMO mixin) ;

//    IMixin getMixin ();    <-- this method is for RELEASE 1.3
   };
```

Supported operations

IMOnlyForMO::IMOnlyForMO::setMixin

IMOnlyForMO::setMixin

This operation is used as a part of the initialization process of the managed object assembly. The mixin provided should be used by the managed object to provide an implementation for many of the methods of the managed object. The implementation of this method must make the mixin provided in the argument available to the other methods on the managed object.

Chapter 20. IBOIMManagedObjectFriendQOS in the Managed Object Framework

The other modules in the Managed Object Framework are:

- CBSeriesGlobal
- IBOIMExtLocal
- IBOIMExtLocalToServer
- IBOIMInstanceManagerFriendQOS
- IBOIMServiceFriendQOS
- ICollectionsBase
- IManagedAdvancedClient
- IManagedAdvancedServer
- IManagedClient
- IManagedCollections
- IManagedLocal
- IManagedServer
- IRDBIMExtLocalToServer

IBOIMManagedObjectFriendQOS Module

The interfaces in this module are used by the IMForMixinAndMO interface in the IBOIMInstanceManagerFriendQOS module. For more information about usage and implementation see the description for that interface.

Note: The interface names and file names are different on OS/390 Component Broker and a Component Broker workstation. On OS/390 the names are IBOIM390ManagedObjectFriendQOS, on a workstation they are IBOIMManagedObjectFriendQOS. For simplicity the interface is described using the Component Broker workstation name.

File name

IBOIMManagedObjectFriendQOS.idl

Intended usage

See IBOIMInstanceManagerFriendQOS::IMForMixinAndMO interface.

Chapter 21. IBOIMServiceFriendQOS in the Managed Object Framework

The other modules in the Managed Object Framework are:

- CBSeriesGlobal
- IBOIMExtLocal
- IBOIMExtLocalToServer
- IBOIMInstanceManagerFriendQOS
- IBOIMManagedObjectFriendQOS
- ICollectionsBase
- IManagedAdvancedClient
- IManagedAdvancedServer
- IManagedClient
- IManagedCollections
- IManagedLocal
- IManagedServer
- IRDBIMExtLocalToServer

IBOIMServiceFriendQOS Module

The interfaces in this module are used by the IMForMixinAndMO interface in the IBOIMInstanceManagerFriendQOS module. For more information about usage and implementation see the description for that interface.

Note: The interface names and file names are different on OS/390 Component Broker and a Component Broker workstation. On OS/390 the names are IBOIM390ServiceFriendQOS, on a workstation they are IBOIMServiceFriendQOS. For simplicity the interface is described using the Component Broker workstation name.

File name

IBOIMServiceFriendQOS.idl

Intended usage

See IBOIMInstanceManagerFriendQOS::IMForMixinAndMO interface.

Chapter 22. ICollectionsBase in the Managed Object Framework

The other modules in the Managed Object Framework are:

- CBSeriesGlobal
- IBOIMExtLocal
- IBOIMExtLocalToServer
- IBOIMInstanceManagerFriendQOS
- IBOIMManagedObjectFriendQOS
- IBOIMServiceFriendQOS
- IManagedAdvancedClient
- IManagedAdvancedServer
- IManagedClient
- IManagedCollections
- IManagedLocal
- IManagedServer
- IRDBIMExtLocalToServer

ICollectionsBase Module

Defines the operations that are common to all collections that can be iterated through (supports the creation of iterators).

File name

ICollectionsBase.idl

Types

```
typedef CosQuery::ParameterList ParameterList;
typedef sequence<any> DataArray;
typedef sequence<DataArray> DataArrayList;
```

Interfaces

ICollectionsBase::IIterator Interface
ICollectionsBase::IMIterable Interface
ICollectionsBase::IMIterableUpdate Interface
ICollectionsBase::IMKeyable Interface
ICollectionsBase::IMKeyedUpdate Interface
ICollectionsBase::IMNonKeyedUpdate Interface
ICollectionsBase::IMNonUniqueKeyable Interface
ICollectionsBase::IMOrderedIterableAdd Interface
ICollectionsBase::IMOrderedNonKeyedAdd Interface
ICollectionsBase::IMQueryable Interface
ICollectionsBase::IMTyped Interface
ICollectionsBase::ITwoWayIterator Interface

Exceptions

IElementNotFound
The specified element is nil or not in the collection.

IInvalidElement
The element is nil, or its type does not match the collection type.

IInvalidIterator
The iterator is pointing before the first element or after the last element, or the iterator is pointing at a collection which does not support modification during iteration and elements have been added or removed.

IInvalidKeyString
The specified byte string is not a key of the correct type.

INoElementForKey
No element with the specified key is in the collection.

IIterator Interface

Defines the operations that are common to all iterators.

File name

ICollectionsBase.idl

Intended usage

Derived from:
IManageable Interface

Supported operations

IIterator::current
IIterator::hasMoreElements
IIterator::more
IIterator::next
IIterator::nextElement
IIterator::nextN
IIterator::nextS
IIterator::nextOne
IIterator::reset

IIterator::current

Retrieves the current object pointed to by the iterator for the collection.

Original interface

ICollectionsBase::IIterator

IDL syntax

```
IManagedClient::IManageable current();
```

Return value

The corresponding managed object.

IIterator::hasMoreElements

Determines if there are more elements left to iterate through for the collection (this is the same as more(), and is here for compatibility with other collection interfaces).

Original interface

ICollectionsBase::IIterator

IDL syntax

```
boolean hasMoreElements();
```

Return value

Returns true (1) if there are more elements, or false (0) if there are not more elements to iterate through for the collection.

IIterator::more

Determines if there are more elements left to iterate through for the collection.

Original interface

ICollectionsBase::IIterator

IDL syntax

```
boolean more();
```

Return value

Returns true (1) if there are more elements, or false (0) if there are not more elements to iterate through for the collection.

IIterator::next

Retrieves the next element from the collection.

Original interface

ICollectionsBase::IIterator

IDL syntax

```
IManagedClient::IManageable next();
```

Return value

The next managed object from the collection, or IManagedClient::IManageable::_nil() if the iterator is currently on the last element (not past the end yet).

Exceptions

IInvalidIterator

IIterator::nextElement

Retrieves the next element from the collection (this is the same as next(), and is here for compatibility with other collection interfaces).

Original interface

ICollectionsBase::IIterator

IDL syntax

```
IManagedClient::IManageable nextElement();
```

Return value

The next managed object from the collection, or IManagedClient::IManageable::_nil() if the iterator is currently on the last element (not past the end yet).

Exceptions

IInvalidIterator

IIterator::nextN

Retrieves the next N elements from the collection.

Original interface

ICollectionsBase::IIterator

IDL syntax

```
boolean nextN(in unsigned log howMany
              out MemberList collectionMembers);
```

Parameters

howMany
An unsigned long value for the number of elements to be retrieved into the output parameter.

collectionMembers
This list is filled with the requested elements (if the iterator was not invalidated).

Return value

Returns true (1) if the requested number of elements were placed in the output parameter; or false (0) if the iterator is invalid or less than the requested number of elements that were placed in the output parameter.

IIterator::nextOne

Retrieves the next element from the collection (this is the same as the next() method, but is more consistent with the syntax of the nextN() method).

Original interface

ICollectionsBase::IIterator

IDL syntax

```
boolean nextOne(out IManagedClient::IManageable theElements);
```

Parameters

theElements
This is set to the next element pointed to by the iterator for the collection (if the iterator was not invalidated).

Return value

Returns true (1) if an element was placed in the output parameter; or false (0) if the iterator is invalid.

IIterator::nextS

Retrieves the next howMany elements from the collection.

Original interface

ICollectionsBase::IIterator

IDL syntax

```
boolean nextS(in unsigned log howMany
              out MemberList collectionMembers);
```

Parameters

howMany
An unsigned long value for the number of elements to be retrieved into the output parameter.

collectionMembers
This list is filled with the requested elements (if the iterator was not invalidated).

Return value

Returns true (1) if the requested number of elements were placed in the output parameter; or false (0) if less than the requested number of elements that were placed in the output parameter.

Exceptions

IInvalidIterator
The iterator is invalid. An exception code, substitution variables and a string variable providing text explanation of the error are contained in the exception.

IIterator::reset

Resets the iterator so that it points before the first element in the collection.

Original interface

ICollectionsBase::IIterator

IDL syntax

```
void reset();
```

IMIterable Interface

defines the operations that are common to all collections that can be iterated through (supports the creation of iterators).

File name

ICollectionsBase.idl

Supported operations

IMIterable::createIterator

IMIterable::createIterator

Manufactures a new iterator for the collection.

Original interface

ICollectionsBase::IMIterable

IDL syntax

```
IIterator createIterator();
```

Return value

A new iterator for the collection.

IMIterableUpdate Interface

Defines the update operations that are common to all collections that can be iterated over (support the creation of iterators).

File name

ICollectionsBase.idl

Supported operations

IMIterableUpdate::addAllElements
IMIterableUpdate::removeElementAt
IMIterableUpdate::replaceElementAt

IMIterableUpdate::addAllElements

Adds all the elements from the specified iterable collection to this collection (if this collection is element type constrained, then the elements in the from collection must be of that type).

Original interface

ICollectionsBase::IMIterableUpdate

IDL syntax

```
void addAllElements(in IMIterable fromCollection);
```

Parameters

fromCollection
An iterable collection from which all the elements are to be copied.

Exceptions

IInvalidElement

IMIterableUpdate::removeElementAt

Removes the element from the collection that is pointed to by the specified iterator.

Original interface

ICollectionsBase::IMIterable

IDL syntax

```
void removeElementAt(in IIterator position);
```

Parameters

position
An iterator that points to the element to be removed from the collection.

Exceptions

IInvalidIterator

IMIterableUpdate::replaceElementAt

Replaces the element from the collection that is pointed to by the specified iterator with the new specified element (if the element is of the correct type for the collection).

Original interface

ICollectionsBase::IMIterableUpdate

IDL syntax

```
void replaceElementAt(in IManagedClient::IManageable newElement,
                      in IIterator position);
```

Parameters

newElement
The managed object that is to replace the element specified by the iterator for the collection.

position
An iterator that points to the element to be replaced in the collection.

Exceptions

IInvalidElement
IInvalidIterator

IMKeyable Interface

Defines the operations that are common to all collections that can access their elements using keys.

File name

ICollectionsBase.idl

Supported operations

IMKeyable::containsKeyString
IMKeyable::getElementByString
IMKeyable::getElementKeyString

IMKeyable::containsKeyString

Determines if there is an element in the collection with the specified key string.

Original interface

ICollectionsBase::IMKeyable

IDL syntax

```
boolean containsKeyString(in ::ByteString keyString);
```

Parameters

keyString
A byte string that contains the key value to search for in the collection.

Return value

Returns true (1) if there is an element in the collection with the specified key string, or false (0) if there is not an element in the collection with the specified key string.

Exceptions

IInvalidKeyString

IMKeyable::getElementByString

Retrieves the element from the collection that matches the specified key string (if any element does).

Original interface

ICollectionsBase::IMKeyable

IDL syntax

```
IManagedClient::IManageable getElementByString(in ::ByteString keyString);
```

Parameters

keyString
A byte string that contains the key value to search for in the collection.

Return value

The managed object (if the key string is valid and a matching element is found).

Exceptions

IInvalidKeyString
INoElementForKey

IMKeyable::getElementKeyString

Retrieves the key string for the specified element in the collection.

Original interface

ICollectionsBase::IMKeyable

IDL syntax

```
::ByteString getElementKeyString(in IManagedClient::IManageable element);
```

Parameters

element
The managed object to search for in the collection.

Return value

A byte string which contains the key value for the specified element (if the element is in the collection).

Exceptions

IElementNotFound

IMKeyedUpdate Interface

Defines the update operations that are common to all collections that support access by keys.

File name

ICollectionsBase.idl

Supported operations

IMKeyedUpdate::addElementByString
IMKeyedUpdate::removeElementByString
IMKeyedUpdate::replaceElementByString

IMKeyedUpdate::addElementByString

Adds the specified element to the collection with the specified corresponding key string (if this collection is element type constrained, then the element must be of that type).

Original interface

ICollectionsBase::IMKeyedUpdate

IDL syntax

```
void addElementByString(in IManagedClient::IManageable element,
                        in ::ByteString keyString);
```

Parameters

element
A managed object to be added to the collection.

keyString
A byte string which contains a key value to be associated with the managed object to be added to the collection.

Exceptions

IInvalidElement
IInvalidKeyString

IMKeyedUpdate::removeElementByString

Removes all instances of the element that match the specified key string from the collection (if there are any matches).

Original interface

ICollectionsBase::IMKeyedUpdate

IDL syntax

```
void removeElementByString(in ::ByteString keyString);
```

Parameters

keyString
A byte string which contains a key value to be used to find the element(s) to be removed from the collection.

Exceptions

IInvalidKeyString

IMKeyedUpdate::replaceElementByString

Replaces all instances of the element in the collection that match the specified key string with the specified new element (if the element is of the correct type for the collection, and the key string matches an element in the collection).

Original interface

ICollectionsBase::IMKeyedUpdate

IDL syntax

```
void replaceElementByString(in IManagedClient::IManageable newElement,
                            in ::ByteString keyString);
```

Parameters

keyString
A byte string which contains a key value to be used to find the element(s) to be replaced in the collection.

newElement
The managed object that is to replace the element(s) in the collection that match the specified key string.

Exceptions

IInvalidElement
IInvalidKeyString
INoElementForKey

IMNonKeyedUpdate Interface

Defines the update operations that are common to all collections that do not support access by keys.

File name

ICollectionsBase.idl

Supported operations

IMNonKeyedUpdate::addElement
IMNonKeyedUpdate::removeElement
IMNonKeyedUpdate::replaceElement

IMNonKeyedUpdate::addElement

Adds the specified element to the collection (if this collection is element type constrained, then the element must be of that type).

Original interface

ICollectionsBase::IMNonKeyedUpdate

IDL syntax

```
void addElement(in IManagedClient::IManageable element);
```

Parameters

element
A managed object to be added to the collection.

Exceptions

IInvalidElement

IMNonKeyedUpdate::removeElement

Removes all instances of the specified element from the collection (if the element is in the collection).

Original interface

ICollectionsBase::IMNonKeyedUpdate

IDL syntax

```
void removeElement(in IManagedClient::IManageable element);
```

Parameters

element
A managed object to be added to the collection.

Exceptions

IElementNotFound

IMNonKeyedUpdate::replaceElement

Replaces all instances of the specified element in the collection with the specified new element (if the element is of the correct type for the collection, and the element to be replaced is in the collection).

Original interface

ICollectionsBase::IMIterableUpdate

IDL syntax

```
void replaceElement(in IManagedClient::IManageable newElement,
                    in IManagedClient::IManageable originalElement);
```

Parameters

newElement
The managed object that is to replace the specified element in the collection.

originalElement
The managed object that is to be replaced in the collection.

Exceptions

IElementNotFound
IInvalidElement

IMNonUniqueKeyable Interface

Defines the operations that are common to all collections that can access their elements via non-unique keys.

File name

ICollectionsBase.idl

Supported operations

IMNonUniqueKeyable::getAllElementsByString

IMNonUniqueKeyable::getAllElementsByString

Manufactures an iterator that can iterate through the elements that matched the specified key string (if any elements do).

Original interface

ICollectionsBase::IMNonUniqueKeyable

IDL syntax

```
IIterator getAllElementsByString(in ::ByteString keyString);
```

Parameters

keyString
A byte string that contains the key value to search for in the collection.

Return value

An iterator object (if the key string is valid).

Exceptions

IInvalidKeyString

IMOrderedIterableAdd Interface

Defines the add operations that are common to all collections that can be iterated over and support ordering of the elements.

File name

ICollectionsBase.idl

Supported operations

IMOrderedIterableAdd::insertElementAt

IMOrderedIterableAdd::insertElementAt

Inserts the element into the collection after the element that is pointed to by the specified iterator (if the element is of the correct type for the collection, and the iterator has not yet been invalidated).

Original interface

ICollectionsBase::IMOrderedIterableAdd

IDL syntax

```
void insertElementAt(in IManagedClient::IManageable element,
                     in IIterator position);
```

Parameters

element

The managed object that is to be inserted after the element specified by the iterator for the collection.

position

an iterator that points to the element after which the element is to be added into the collection.

Exceptions

IInvalidElement
IInvalidIterator

IMOrderedNonKeyedAdd Interface

Defines the add operations that are common to all collections that do not support access by keys but do support ordering of elements.

File name

ICollectionsBase.idl

Supported operations

IMOrderedNonKeyedAdd::insertElementAfter
IMOrderedNonKeyedAdd::insertElementBefore

IMOrderedNonKeyedAdd::insertElementAfter

Inserts the specified element into the collection after the specified element (if the element is of the correct type for the collection, and the element to be inserted after is in the collection).

Original interface

ICollectionsBase::IMOrderedNonKeyedAdd

IDL syntax

```
void insertElementAfter(in IManagedClient::IManageable element,
                        in IManagedClient::IManageable afterElement);
```

Parameters

element
The managed object that is to be inserted into the collection.

afterElement
The managed object after which the specified element is to be inserted into the collection.

Exceptions

IElementNotFound
InvalidElement

IMOrderedNonKeyedAdd::insertElementBefore

Inserts the specified element into the collection before the specified element (if the element is of the correct type for the collection, and the element to be inserted before is in the collection).

Original interface

ICollectionsBase::IMOrderedNonKeyedAdd

IDL syntax

```
void insertElementBefore(in IManagedClient::IManageable element,
                         in IManagedClient::IManageable beforeElement);
```

Parameters

element
The managed object that is to be inserted into the collection.

beforeElement
The managed object before which the specified element is to be inserted into the collection.

Exceptions

IElementNotFound
InvalidElement

IMQueryable Interface

Defines the operations that are common to all collections that can be queried for a subset of their elements.

File name

ICollectionsBase.idl

Intended usage

The IMQueryable interfaces are used to submit OOSQL queries to the Component Broker server. Results are returned in the form of iterators that range over the results of a query. The extendedEvaluateToDataArray is the most general query operation. It accepts a full OOSQL query and returns a DataArrayIterator as result. The extendedEvaluate operation accepts a restricted form of OOSQL query. The projection list of an OOSQL query is restricted to a single element of object type. The result of this interface is an iterator that returns objects, instead of the more complex data arrays. The evaluate is a simplified query interface to the collection. Note that this interface is abstract; other collection interfaces will provide concrete implementations of these interfaces.

Supported operations

IMQueryable::evaluate
IMQueryable::extendedEvaluate
IMQueryable::extendedEvaluateToDataArray

IMQueryable::evaluate

Manufactures an iterator that will iterate through all the elements in the collection that match the specified query.

Original interface

ICollectionsBase::IMQueryable

IDL syntax

```
IIterator evaluate(in string queryPredicate);
```

Parameters

queryPredicate

A string that contains the query statement to be evaluated against the collection.

```
select a from policyHome a where policyNo > 3000.
This will select  policy numbers > 3000 from policyHome.
```

For details on the syntax of OOSQL query please see section “Object Oriented Structured Query Language” in *Component Broker Advanced Programming Guide*.

Exceptions

ICollectionsBase::QueryInvalid
ICollectionsBase::QueryProcessingError
ICollectionsBase::QueryTypeInvalid

IMQueryable::extendedEvaluate

Produces an iterator over a set of objects resulting from an object query. This operation provides a keyword ″thisCollection″ that is used to identify the collection that this method was executed against. and allows a query to be coded such as:

```
select e from thisCollection e where e.empid>6;
```

AIX WIN Supported on AIX and Windows NT only

Original interface

ICollectionsBase::IMQueryable Interface

IDL syntax

```
IManagedCollections::IIterator extendedEvaluate(
  in string query,
  in ParameterList collection_names,
  in ParameterList params,
  in unsigned long how_many,
  out MemberList members)
raises(ICollectionsBase::IQueryProcessingError,
       ICollectionsBase::IQueryInvalid,
       ICollectionsBase::IQueryTypeInvalid);
```

Remarks

The extendedEvaluate operation is used to produce an iterator over a set of objects resulting from an object query. The type of the iterator is IManagedCollections::IIterator. The interface is designed to closely resemble the CosQuery::QueryEvaluator::evaluate operation. The extendedEvaluate returns an iterator and optional member list as out arguments, thus making it perhaps more convenient to use than the standard evaluate

method. If how_many is set greater than zero, then that number of members are returned in the member list (fewer if how_many exceeds the total number of members resulting from the query) and the remainder are returned in the iterator. If how_many is set to zero then members will be empty and all of the resulting members are returned in the iterator. If all of the members are returned in the member list then the iterator is a NULL.

The iterator returned is of the type IManagedCollection::IIterator. However, the real run time type of the iterator is a subtype of IQueryManagedCollections::IIterator, which itself is a subtype of IManagedCollections::IIterator. This means that from an interface point of view the returned iterator can be treated as an IManagedCollections::IIterator but the implementation behavior such as persistency, home of the iterator, and so on, may be different from other implementations of IManagedCollections::IIterator in Component Broker such as the iterator implementation provided by the reference collection component.

When a non-NULL iterator is returned it must be removed and released by the client when it is through using it.

IMQueryable::extendedEvaluateToDataArray

Used to produce an iterator over a set of DataArrays resulting from an object query. This operation provides a keyword ″thisCollection″ that is used to identify the collection that this method was executed against. and allows a query to be coded such as:

```
select e from thisCollection e where e.empid>6;
```

Supported on AIX and Windows NT only

Original interface

ICollectionsBase::IMQueryable Interface

IDL syntax

```
Object extendedEvaluateToDataArray(
  in string query,
  in ParameterList collection_names,
  in ParameterList params,
  in unsigned long how_many,
  out DataArrayList aggregate_members)
raises(ICollectionsBase::IQueryProcessingError,
       ICollectionsBase::IQueryInvalid,
       ICollectionsBase::IQueryTypeInvalid);
```

Remarks

The extendedEvaluateToDataArray method is used to produce an iterator over a set of DataArrays resulting from an object query. A DataArray is essentially a row in a relational database table or a view projected on a set of columns. A DataArray can

represent partial data of an object or combination of data from different objects. A DataArray is a sequence of anys. If an element of a DataArray is a null value then the any corresponding to that element has the value any.type()= tk_null and any.value()= undefined. The original data type of the column corresponding to the NULL value can be obtained from the method get_field_type on DataArrayIterator returned from an extendedEvaluateToDataArray method call. The extendedEvaluateToDataArray returns an iterator and optional member list as out-arguments, thus making it perhaps more convenient to use than the standard evaluate method.

If how_many is set greater than zero, then that number of DataArray are returned in the aggregate_members (fewer if how_many exceeds the total number of DataArrays resulting from the query) and the remainder are returned in the iterator. If how_many is set to zero then aggregate_members will be empty and all resulting DataArrays are returned in the iterator. If all of the DataArrays are returned in the aggregate_members then the iterator is a NULL.

When a non-NULL iterator is returned it must be removed and released by the client when it is through using it.

This interfaces returns an Object. It is neccesary to _narrow() the returned object to an IExtendedQuery::DataArrayIterator in order to access the returned data.

IMTyped Interface

Defines the operations that are common to all collections that can be constrained to hold a single type of element.

File name

ICollectionsBase.idl

Supported operations

IMTyped::getElementClassName
IMTyped::getElementInterface

IMTyped::getElementClassName

Retrieves the name of the class (interface) for the type of element stored in the collection (this is the same as calling `getElementInterface() -> absolute_name()`).

Original interface

ICollectionsBase::IMTyped

IDL syntax

```
string getElementClassName();
```

Return value

A string containing the element class name.

IMTyped::getElementInterface

Retrieves the CORBA interface definition for the type of element stored in the collection.

Original interface

ICollectionsBase::IMTyped

IDL syntax

```
CORBA::InterfaceDef getElementInterface();
```

Return value

A CORBA interface definition for the element type.

ITwoWayIterator Interface

Defines the operations that are common to all bi-directional iterators.

File name

ICollectionsBase.idl

Intended usage

Derived from:
IIterator Interface

Supported operations

ITwoWayIterator::isFirst
ITwoWayIterator::isLast
ITwoWayIterator::positionOn
ITwoWayIterator::previous
ITwoWayIterator::resetAfter
ITwoWayIterator::resetBefore

ITwoWayIterator::isFirst

Determines if the iterator is currently positioned over the first element in the collection.

Original interface

ICollectionsBase::ITwoWayIterator

IDL syntax

```
boolean isFirst();
```

Return value

Returns true (1) if the iterator is positioned over the first element, or false (0) if the iterator is not positioned over the first element.

ITwoWayIterator::isLast

Determines if the iterator is currently positioned over the last element in the collection.

Original interface

ICollectionsBase::ITwoWayIterator

IDL syntax

```
boolean isLast()
```

Return value

Returns true (1) if the iterator is positioned over the last element, or false (0) if the iterator is not positioned over the last element.

ITwoWayIterator::positionOn

Position the iterator over the specified element in the collection.

Original interface

ICollectionsBase::ITwoWayIterator

IDL syntax

```
void positionOn(in IManagedClient::IManageable element);
```

Parameters

element
Object in the collection over which the iterator is to be positioned.

Exceptions

IElementNotFound

ITwoWayIterator::previous

Retrieves the previous element from the collection.

Original interface

ICollectionsBase::ITwoWayIterator

IDL syntax

```
IManagedClient::IManageable previous();
```

Return value

The previous managed object from the collection, or IManagedClient::IManageable::_nil() if the iterator is currently on the first element (not before the start yet).

Exceptions

IInvalidIterator

ITwoWayIterator::resetAfter

Resets the iterator so that it points after the last element in the collection.

Original interface

ICollectionsBase::ITwoWayIterator

IDL syntax

```
void resetAfter();
```

ITwoWayIterator::resetBefore

Resets the iterator so that it points before the first element in the collection (this is the same as reset(), but is included for consistency with resetAfter()).

Original interface

ICollectionsBase::ITwoWayIterator

IDL syntax

```
void resetBefore();
```

Chapter 23. IExtendedEventChannelAdmin in the Event Service

The other modules in the Event Service are:
- CosEventChannelAdmin
- CosEventComm

Note: The Event Service is not supported by OS/390 Component Broker.

IExtendedEventChannelAdmin Module

Allows applications to create or find an EventChannel object on the server.

Not supported by OS/390 Component Broker

File name

IExtendedEventChannelAdmin.idl

Intended usage

The IExtendedEventChannelAdmin module is an abstract class from which actual EventChannel home implementation inherits. The EventChannel home object is implemented by the Event Service in the Component Broker. It is not intended to be implemented or modified by any customers.

Interfaces

IExtendedEventChannelAdmin::IEventChannelHome Interface

IEventChannelHome Interface

Defines three operations: create EventChannel, find EventChannel, and create a visible EventChannel in the CDS.

File name

IExtendedEventChannelAdmin.idl

Intended usage

An abstract base class from which actual create or find EventChannel implementation inherits.

IDL syntax

```
interface IEventChannelHome : IManagedClient::IHome
{
    IEventChannelAdminManagedClient::EventChannel
       createEventChannel(out ByteString key);
    IEventChannelAdminManagedClient::EventChannel
       findEventChannel(in ByteString key);
    IEventChannelAdminManagedClient::EventChannel
       createVisibleEventChannel(out ByteString key,
                                  in string relativeName,
                                  in boolean visibleInCellNameTree,
                                  in boolean visibleInHostNameTree,
                                  in boolean visibleInWorkGroupNameTree);
};
```

Supported operations

IEventChannelHome::createEventChannel
IEventChannelHome::createVisibleEventChannel
IEventChannelHome::findEventChannel

IEventChannelHome::createEventChannel

Returns both an EventChannel object and a ByteString as key which can be used to find an EventChannel object.

Original interface

IExtendedEventChannelAdmin::IEventChannelHome Interface

IDL syntax

```
IEventChannelAdminManagedClient::EventChannel
   createEventChannel(out ByteString key);
```

Parameters

key A ByteString used as a key for the created EventChannel object.

Return value

IEventChannelAdminManagedClient::EventChannel
A new EventChannel object.

Remarks

This method is called by the clients and implemented by the Event Server in the Component Broker. It is not intended to be implemented or modified by any customers.

Example

```
CORBA::Object_var intermediateObject;
IExtendedLifeCycle::FactoryFinder_var hostScopeFF;
IExtendedEventChannelAdmin::IEventChannelHome_var ecHome;
ByteString_var key;
CosEventChannelAdmin::EventChannel_var ec;

                    // initialize the CBSeries environment
CBSeriesGlobal::Initialize();
                    // obtain the default factory finder with a host scope
intermediateObject =
   CBSeriesGlobal::nameService()->resolve_with_string(
                              "host/resources/factory-finders/host-scope");
                    // narrow to a factory finder
hostScopeFF =
   IExtendedLifeCycle::FactoryFinder::_narrow(intermediateObject);
                    // find the event channel factory
intermediateObject = hostScopeFF->find_factory_from_string(
   "IEventChannelAdminManagedClient::EventChannel.object interface");
                     // narrow to the event channel home
ecHome = IExtendedEventChannelAdmin::IEventChannelHome::_narrow(
                                                        intermediateObject);
                     // use the factory in the event channel home to create
                     // a new event channl
ec = ecHome->createEventChannel(key);
```

IEventChannelHome::createVisibleEventChannel

Returns both an EventChannel object and a ByteString as key which can be used to find an EventChannel object. The user can decide whether or not the created EventChannel object should be visible in the CDS under the cell name tree, host name tree, or workgroup name tree.

Original interface

IExtendedEventChannelAdmin::IEventChannelHome Interface

IDL syntax

```
IEventChannelAdminManagedClient::EventChannel createVisibleEventChannel(
                                      out ByteString key,
                                      in string relativeName,
                                      in boolean visibleInCellNameTree,
                                      in boolean visibleInHostNameTree,
                                      in boolean visibleInWorkGroupNameTree);
```

Parameters

key Indicates a ByteString for the created EventChannel object.

relativeName
A string name to be used under the name tree.

visibleInCellNameTree
A boolean value indicates whether or not the relativeName should appear under the cell name tree.

visibleInHostNameTree
A boolean value indicates whether or not the relativeName should appear under the host name tree.

visibleInWorkGroupNameTree
A boolean value indicates whether or not the relativeName should appear under the workgroup name tree.

Return value

IEventChannelAdminManagedClient::EventChannel
A new EventChannel object.

Remarks

This method is called by the clients and implemented by the Event Server in the Component Broker. It is not intended to be implemented or modified by any customers.

Example

```
CORBA::Object_var intermediateObject;
IExtendedLifeCycle::FactoryFinder_var hostScopeFF;
IExtendedEventChannelAdmin::IEventChannelHome_var ecHome;
ByteString_var key;
CosEventChannelAdmin::EventChannel_var ec;

                    // initialize the CBSeries environment
CBSeriesGlobal::Initialize();
                    // obtain the default factory finder with a host scope
intermediateObject = CBSeriesGlobal::nameService()->resolve_with_string(
                              "host/resources/factory-finders/host-scope");
                    // narrow to a factory finder
hostScopeFF = IExtendedLifeCycle::FactoryFinder::_narrow(intermediateObject);
                    // find the event channel factory
intermediateObject = hostScopeFF->find_factory_from_string(
         "IEventChannelAdminManagedClient::EventChannel.object interface");
                    // narrow to the event channel home
ecHome = IExtendedEventChannelAdmin::IEventChannelHome::_narrow(
                                                        intermediateObject);
                    // use the factory in the event channel home to create
                    // a new event channl and bind in the system name space
ec = ecHome->createVisibleEventChannel(key, "AgentActions", 0, 1, 0);
```

IEventChannelHome::findEventChannel

Finds an EventChannel object on the server from a ByteString key.

Original interface

IExtendedEventChannelAdmin::IEventChannelHome Interface

IDL syntax

```
IEventChannelAdminManagedClient::EventChannel
                                        findEventChannel(in ByteString key);
```

Parameters

key A ByteString used as the key of the EventChannel object.

Return value

IEventChannelAdminManagedClient::EventChannel
A new EventChannel object.

Remarks

This operation is called by the clients and implemented by the Event Server in the Component Broker. It is not intended to be implemented or modified by you, the developer.

Example

```
CORBA::Object_var intermediateObject;
IExtendedLifeCycle::FactoryFinder_var hostScopeFF;
IExtendedEventChannelAdmin::IEventChannelHome_var ecHome;
ByteString_var key;
CosEventChannelAdmin::EventChannel_var ec, ec1;

                    // initialization the CBSeries environment
CBSeriesGlobal::Initialize();
                    // obtain the default factory finder with a host scope
intermediateObject = CBSeriesGlobal::nameService()->resolve_with_string(
                             "host/resources/factory-finders/host-scope");
                    // narrow to a factory finder
hostScopeFF = IExtendedLifeCycle::FactoryFinder::_narrow(intermediateObject);
                    // find the event channel factory
intermediateObject = hostScopeFF->find_factory_from_string(
         "IEventChannelAdminManagedClient::EventChannel.object interface");
                    // narrow to the event channel home
ecHome = IExtendedEventChannelAdmin::IEventChannelHome::_narrow(
                                                        intermediateObject);
                    // use the factory in the event channel home to create
                    // a new event channl
ec = ecHome->createEventChannel(key);
...
                    // use the key to find the event channel where ec1
                    // should be identical to the ec created before
ec1 = ecHome->findEventChannel(key);
```

Chapter 24. IExtendedLifeCycle in the LifeCycle Service

The other modules in the LifeCycle Service are:
- CosLifeCycle
- ILifeCycleLocalObjectImpl
- ILifeCycleManagedClient

IExtendedLifeCycle Module

Introduces the complete set of operational interfaces for the Lifecycle service.

File name

IExtendedLifeCycle.idl

Types

AIX WIN For AIX and Windows NT:

```
struct Scope {
    CosNaming::Istring cell;
    CosNaming::Istring workgroup;
    CosNaming::Istring host;
    CosNaming::Istring server;
    CosNaming::Istring container;
    CosNaming::Istring home;
};
typedef sequence <Scope> OrderedScopes;
typedef CosNaming::IString::NameString ScopeString;
typedef CosNaming::IString::NameString FactoryKeyString;
typedef sequence<ScopeString> OrderedScopeStrings;
typedef sequence<Location> SequenceOfLocations
```

390 SOLARIS For OS/390 and Solaris:

```
struct Scope {
    CosNaming::Istring cell;
    CosNaming::Istring workgroup;
    CosNaming::Istring host;
    CosNaming::Istring server;
    CosNaming::Istring container;
    CosNaming::Istring home;
};
typedef sequence <Scope> OrderedScopes;
typedef NamingStringSyntax::NameString ScopeString;
typedef NamingStringSyntax::NameString FactoryKeyString;
typedef sequence<ScopeString> OrderedScopeStrings;
typedef sequence<Location> SequenceOfLocations
```

Exceptions

CORBA standard exceptions and the following user exceptions:
IExtendedLifeCycle::InvalidScope
IExtendedLifeCycle::UnrecognizedScopeElement
IExtendedLifeCycle::UnMatchedQuote
IExtendedLifeCycle::IllegalStringSyntax

Interfaces

IExtendedLifeCycle::FactoryFinder Interface
IExtendedLifeCycle::Location Interface
IExtendedLifeCycle::OrderedLocation Interface
IExtendedLifeCycle::ScopeManipulator Interface
IExtendedLifeCycle::SingleLocation Interface
IExtendedLifeCycle::FactoryFinderHome Interface
IExtendedLifeCycle::OrderedLocationHome Interface
IExtendedLifeCycle::SingleLocationHome Interface

FactoryFinder Interface

Provides extensions to the “FactoryFinder Interface” on page 339. These extensions are intended to make the interface easier to use and to improve performance.

Supported operations

FactoryFinder::find_factories_from_string
FactoryFinder::find_factory
FactoryFinder::find_factory_from_string
FactoryFinder::get_location

FactoryFinder::find_factories_from_string

Finds all factories that meet the constraints specified in the factory_key string and the rules configured into the FactoryFinder itself.

Original interface

IExtendedLifeCycle::FactoryFinder

IDL syntax

```
CosLifeCycle::Factories find_factories_from_string(
      in FactoryKeyString factory_key)
   raises(CosLifeCycle::NoFactory,
          IllegalStringSyntax,
          UnMatchedQuote);
```

Parameters

factory_key

String form of a factory key. See the sections entitled “Factory Keys” and “Factory Key Structures and Strings”, in the *Component Broker Advanced Programming Guide*, for complete information on the meanings and syntax of these values.

Return value

factories

Sequence of object references to factories found.

Exceptions

CORBA standard exceptions and the following user exceptions:

IExtendedLifeCycle::IllegalStringSyntax - indicates that the factory_key string has a syntax error.

CosLifeCycle::NoFactory - The factory finder could not find any factories that met both the criteria specified in the Key and the scope configured into the factory finder itself. A minor code of LCERR_FAC_FIND_INVALID_KEY (0x49420201) indicates that the key was not in the correct form. See Factory Keys. Check that the key used in the find factory operation is correct. Try using a factory_finder with a less restrictive scope. Check the activity log to verify that the registration of the expected factories did not result in any errors.

IExtendedLifeCycle::UnMatchedQuote - indicates that there is an unmatched quote in the factory_key string. Ensure that the name contains all ending and beginning quotes paired.

Remarks

This operation is very similar to the CosLifeCycle::find_factories operation except that it allows the user to specify the key in string form instead of as a CosLifeCycle::Key. It uses the NamingStringSyntax module to convert the key string to a CosLifeCycle::Key as part of its execution.

The find_factories_from_string operation will return only after finding all factories that meet the caller’s request. Since the caller’s request may be very general, it is possible that Lifecycle may have to check a large number of factories before returning the (potentially very large) list of factories.

Once a factory has been found, it can be used to create objects with the interface type specfied in the factory_key.

Example

```
// C ++ example
// Note: for code clarity, some standard client setup, exception handling,
// error checking and cleanup have been ommitted from this example.
```

```
   extern IExtendedNaming::NamingContext_ptr root_context; // previously
                                                           // initialized

   extern CosLifeCycle::FactoryFinder_var  finder; // previously
                                                   // initialized

// find the factories which supports the "MyTest" interface

   cout << "find MyTest factories \n";
   cout.flush();

   CosLifeCycle::Factories_var temp_seq =
      finder->find_factories_from_string("MyTest.object interface");
```

FactoryFinder::find_factory

Finds a factory that meets the constraints specified in the factory_key and the rules configured into the FactoryFinder.

Original interface

IExtendedLifeCycle::FactoryFinder

IDL syntax

```
CosLifeCycle::Factory find_factory(in CosLifeCycle::Key factory_key)
   raises(CosLifeCycle::NoFactory);
```

Parameters

factory_key
String form of the key. See the sections entitled “Factory Keys” and “Factory Key Structures and Strings”, in the *Component Broker Advanced Programming Guide*, for complete information on the meanings and syntax of these values.

Return value

factory Object reference to the factory found.

Exceptions

CORBA standard exceptions and the following user exceptions:

IExtendedLifeCycle::NoFactories - the factory finder could not find any factories that met both the criteria specified in the Key and the scope configured into the factory finder itself. A minor code of LCERR_FAC_FIND_INVALID_KEY (0x49420201) indicates that the key was not in the correct form. See Factory Keys. Check that the key used in the find factory operation is correct. Try using a factory_finder with a less restrictive scope. Verify that the registration of the expected factories did not result in any errors by checking the activity log.

Remarks

This operation is similar to the CosLifeCycle::find_factories operation except that it only returns one factory. If more than one such factory exists, the factory returned is simply the first one found in the LifeCycle factory repository. Because this operation returns once the first factory is found, its performance is expected to be better than that of find_factories which continues searching until all factories have been found.

Once a factory has been found, it can be used to create objects with the interface type specfied in the factory_key.

FactoryFinder::find_factory_from_string

Find a factory that meets the constraints specified in the factory_key string and the rules configured into the FactoryFinder itself.

Original interface

IExtendedLifeCycle::FactoryFinder

IDL syntax

```
CosLifeCycle::Factories
      find_factory_from_string(in FactoryKeyString factory_key)
   raises(CosLifeCycle::NoFactory,
          IllegalStringSyntax,
          UnMatchedQuote);
```

Parameters

factory_key
String form of the key. See the sections entitled “Factory Keys” and “Factory Key Structures and Strings”, in the *Component Broker Advanced Programming Guide*, for complete information on the meanings and syntax of these values.

Return value

factory Object reference to the factory found.

Exceptions

CORBA standard exceptions and the following user exceptions:

IExtendedLifeCycle::IllegalStringSyntax - indicates that the factory_key string has a syntax error.

IExtendedLifeCycle::NoFactories The factory finder could not find any factories that met both the criteria specified in the Key and the scope configured into the factory finder itself. A minor code of LCERR_FAC_FIND_INVALID_KEY (0x49420201) indicates that the key was not in the correct form. See Factory Keys. Check that the key used in the find factory operation is correct. Try using a factory_finder with a

less restrictive scope. Verify that the registration of the expected factories did not result in any errors by checking the event log.

IExtendedLifeCycle::UnMatchedQuote - indicates that there is an unmatched quote in the factory_key string.

Remarks

Find a factory that meets the constraints specified in the factory_key string and the rules configured into the FactoryFinder itself.

If more than one such factory exists, the factory returned is simply the first one found in the LifeCycle factory repository. Because this operation returns once the first factory is found, its performance is expected to be better than that of find_factories which continues searching until all factories have been found.

This operation is very similar to the IExtendedLifeCycle::find_factory operation except that it allows the user to specify the key in string form instead of as a CosLifeCycle::Key. It uses the NamingStringSyntax module to convert the key string to a CosLifeCycle::Key as part of its execution. Once a factory has been found, it can be used to create objects with the interface type specfied in the factory_key.

Example

```
// C ++ example
// Note: For code clarity, some exception handling, error checking and
// cleanup have been omitted from this example.
   extern IExtendedNaming::NamingContext_ptr root_context; // previously
                                                           // initialized
   extern CosLifeCycle::FactoryFinder_var  finder; // previously initialized
// find a factory which supports the "MyTest" interface

   cout << "find MyTest factory \n";
   cout.flush();

   CosLifeCycle::Factory_var temp_fac =
                finder->find_factory_from_string("MyTest.object interface");
```

FactoryFinder::get_location

Returns a reference to the Location object used by the factory finder.

Original interface

IExtendedLifeCycle::FactoryFinder

IDL syntax

```
Location get_location();
```

Return value

Location
Object reference of the location.

Remarks

Returns a reference to the Location object used by the factory finder. The Location may be an IExtendedLifeCycle::SingleLocation, IExtendedLifeCycle::OrderedLocation or a user implementated location. The location my be useful for retrieving the scope information which controls the searching behavior of the factory finder. See IExtendedLifeCycle::Location.

Example

See the example in IExtendedLifeCycle::Location get_scopes Operation.

FactoryFinderHome Interface

Provides a means for clients to create managed factory finders configured with their own location parameter.

Ancestor interfaces

IManagedClient::IHome

Supported operations

FactoryFinderHome::createWithLocation

FactoryFinderHome::createWithLocation

Creates a managed factory finder object and configures it with the input location.

Original interface

IExtendedLifeCycle::FactoryFinderHome

IDL syntax

```
IExtendedLifeCycle::FactoryFinder createWithLocation(
                        in Location location,
                        in CosNaming::Istring relativeName,
                        in boolean visibleInCellNameTree,
                        in boolean visisbleInHostNameTree,
                        in boolean visibleInWorkgroupNameTree
                        );
```

Parameters

location
A valid managed IExtendedLifeCycle::Location object. The object must inherit from IManagedClient::IManageable to be valid.

relativeName
The name under which the created factory finder is to be registered. If the name specified is an empty string, the factory finder will not be registered in the name space.

visibleInCellNameTree
If TRUE, the factory finder will be registered in the cell name tree.

visibleInHostNameTree
If TRUE, the factory finder will be registered in the host name tree.

visibleInWorkgroupNameTree
If TRUE, the factory finder will be registered in the Workgroup name tree.

Return value

FactoryFinder
The newly created factory finder object.

Exceptions

CORBA standard exceptions.

Example

```
//*************************************************************************
//  This example uses Naming to find the default managed factory finder for
//  this host.  It then uses the factory finder to find the specialized homes
//  for factory finders and single locations.  Next it uses the single
//  location home to create a new single location.  This is then used with
//  the factory finder factory to create a new factory finder.
//*************************************************************************

   extern IExtendedNaming::NamingContext_ptr root_context; // previously
                                                           // initialized
   ......

// get default factory

   cout << " get default factory finder \n";
   CORBA::Object_var temp_var = root_context->resolve_with_string(
                  "host/resources/factory-finders/host-scope");

   IExtendedLifeCycle::FactoryFinder_var  finder;
   finder = IExtendedLifeCycle::FactoryFinder::_narrow(temp_var);
   if (!finder)
   {
        cout << "Couldn't get default factory finder \n";
        exit(-1);
```

```
   }

// find factory finder factory

   cout << "get managed factory finder factory \n";
   CORBA::Object_var temp_obj = finder->find_factory_from_string(
                  "ILifeCycleManagedClient::FactoryFinder.object interface");
   ILifeCycleManagedClient::FactoryFinderHome_var
      managed_factory_finder_factory;
   managed_factory_finder_factory =
      ILifeCycleManagedClient::FactoryFinderHome::_narrow temp_obj);
   if (!managed_factory_finder_factory)
   {
        cout << "Couldn't get managed factory finder factory \n";
        exit(-1);
   }


// Now get single location factory


   cout << "get managed single location factory \n";
   CORBA::Object_var temp_obj2 = finder->find_factory_from_string(
                  "ILifeCycleManagedClient::SingleLocation.object interface");
   ILifeCycleManagedClient::SingleLocationHome_var  managed_single_loc;
   managed_single_loc = ILifeCycleManagedClient
                        ::SingleLocationHome::_narrow(temp_obj2);
   if (!managed_single_loc)
   {
        cout << "Couldn't get managed single loc factory \n";
        exit(-1);
   }

// Create up your very own single location
// The scope will specify my server named "MyTest."
// This will cause only factories on my server to be found.

   cout << "create new locfactory \n";
   IExtendedLifeCycle::SingleLocation_var  new_single_loc;
   new_single_loc = managed_single_loc->createWithScopeString(
        "MyTest.server",            // The new scope for this location.
        "MyTest_server_location",   // The name to register in the name space.
        0,                          // Do NOT register in cell name tree
        1,                          // Register in host name tree.
        0);                         // Do NOT register in workgroup name tree

   IExtendedLifeCycle::Location_var  new_loc;
   new_loc = IExtendedLifeCycle::Location::_narrow(new_single_loc);
   if (!new_loc)
   {
        cout << "Couldn't get new locfactory \n";
        exit(-1);
   }

// Use new location to create new factory finder
```

```
cout << "create new factory finder \n";
IExtendedLifeCycle::FactoryFinder_var  new_fac_find;
new_fac_find = managed_factory_finder_factory->createWithLocation(
     new_loc,                  // The location created above.
     "MyTest_server_factory",  // The name to register in the name space.
     0,                        // Do NOT register in cell name tree
     1,                        // Register in host name tree.
     0);                       // Do NOT register in workgroup name tree

if (!new_fac_find)
{
     cout << "Couldn't get new fac_find \n";
     exit(-1);
}

// Now new_fac_find can be used to find factories on MyTest server.
```

Location Interface

A FactoryFinder object encapsulates a Location object. The Location object is used to define the boundaries which the FactoryFinder is to limit itself to when locating factories. This information is contained in a sequence of scopes within the location object. See “Defining scope of location” in the *Component Broker Advanced Programming Guide* for more information. No direct implementation is supplied for the Location Interface. However, implementation for IExtendedLifeCycle::SingleLocation and IExtendedLifeCycle::OrderedLocation, which are specializations of Location, are supplied.

Supported operations

Location::get_scopes

Location::get_scopes

Returns the sequence of scopes contained in the Location object.

Original interface

IExtendedLifeCycle::Location

IDL syntax

```
OrderedScopes get_scopes();
```

Return value

scopes

Sequence of scopes.

Remarks

The Location object is used to define the boundaries which the FactoryFinder is to limit itself to when locating factories. This information is contained in a series of scopes within the location object. See “Defining scope of location” in the *Component Broker Advanced Programming Guide* for more information. It may be useful to examine the scopes to determine the extent of the FactoryFinder’s search area.

Example

```
// C ++ example
// Note: for code clarity, some exception standard client setup, handling,
//  error checking and cleanup have been omitted from this example.

   extern IExtendedNaming::NamingContext_ptr root_context; // previously
                                                           // initialized

   extern CosLifeCycle::FactoryFinder_var  finder; // previously initialized

// get the location from the factory finder

   IExtendedLifeCycle::Location_var     loc;

   loc = finder->get_location();

// Now get the scopes from the location.

   IExtendedLifeCycle::OrderedScopes_var        scopes;

   scopes = loc->get_scopes();


// Print out the number of scopes.

   cout << "Found " << scopes->length() << " scopes
      in factory finder\n";

// Print out each scope

   for (int i=0;i < scopes->length(); i++)
   {
      cout << "Content of scope " << i << endl;

      cout << "      Cell : " << (*scopes)[i]].cell << endl;
      cout << " Workgroup : " << (*scopes)[i].workgroup << endl;
      cout << "      Host : " << (*scopes)[i].host << endl;
      cout << "    Server : " << (*scopes)[i].server << endl;
      cout << " Container : " << (*scopes)[i].container << endl;
      cout << "      Home : " << (*scopes)[i].home << endl;
   }

}
```

OrderedLocation Interface

A FactoryFinder object encapsulates a Location object. The Location object is used to define the boundaries which the FactoryFinder is to limit itself to when locating factories. This information is contained in a sequence of scopes within the location object. OrderedLocation is a specialization of the IExtendedLifeCycle::Location interface. An OrderedLocation contains a sequence of location object references. These references my be to SingleLocations, other OrderedLocations, or to user-implemented specializations of the IExtendedLifeCycle::Location interface. A get_scopes operation will return a concatenated sequence of all scopes for each of the locations. A find factory operation on a factory finder configured with an ordered location performs its search using the first scope, then the second scope, etc. It will only throw a NoFactory exception if it cannot find a factory using ANY of the scopes.

Note: CB/390 has a current limitation of five location objects within a given ordered location.

Supported operations

OrderedLocation::get_locations

OrderedLocation::get_locations

The OrderedLocation object is used to define the boundaries which the FactoryFinder is to limit itself to when locating factories.

Original interface

IExtendedLifeCycle::OrderedLocation

IDL syntax

```
Scope get_locations();
```

Return value

locations

A sequence of location references.

Remarks

The OrderedLocation object is used to define the boundaries which the FactoryFinder is to limit itself to when locating factories. This information is contained in a sequence of locations within the OrderedLocation object. This operation returns the sequence of locations.

In general it is not necessary to examine individual members of the OrderedLocation sequence. However, this operation may be useful for creating additional OrderedLocation objects using the individual location components of an already existing

OrderedLocation. This can be done with the IExtendeLifeCycle::OrderedLocationHome::createWithLocations operation.

Example

```
// C ++ example
// Note: for code clarity, some standard client setup, exception handling,
//  error checking and cleanup have been omitted from this example.

   extern IExtendedNaming::NamingContext_ptr root_context; // previously
                                                           // initialized

   extern IExtendedLifeCycle::Location_var loc; // previously initialized

   loc = finder->get_location();

// Narrow to a OrderedLocation

   IExtendedLifeCycle::OrderedLocation_var       ordered_loc;
   ordered_loc = IExtendedLifeCycle::OrderedLocation::_narrow(loc);

// Now get the locations from the ordered location.

   IExtendedLifeCycle::SequenceOfLocations        *locations;

   locations = ordered_loc->get_locations();
```

OrderedLocationHome Interface

Provides a means for clients to create managed ordered locations configured with their own sequence of scopes.

Ancestor interfaces

IManagedClient::IHome

Supported operations

OrderedLocationHome::createWithLocations
OrderedLocationHome::createWithScopes
OrderedLocationHome::createWithScopeStrings

OrderedLocationHome::createWithScopes

Creates a managed ordered location object and configures it with the input scope sequence.

Original interface

IExtendedLifeCycle::OrderedLocationHome

IDL syntax

```
IExtendedLifeCycle::OrderedLocation createWithScopes(
                        in OrderedScopes scopes,
                        in CosNaming::Istring relativeName,
                        in boolean visibleInCellNameTree,
                        in boolean visisbleInHostNameTree,
                        in boolean visibleInWorkgroupNameTree);
   raises (IExtendedLifeCycle::InvalidScope);
```

Parameters

scopes
An ordered sequence of scope structures which will define the search order of the location.

relativeName
The name under which the created ordered location is to be registered. If the name specified is an empty string, the ordered location will not be registered in the name space.

visibleInCellNameTree
If TRUE, the ordered location will be registered in the cell name tree.

visibleInHostNameTree
If TRUE, the ordered location will be registered in the host name tree.

visibleInWorkgroupNameTree
If TRUE, the ordered location will be registered in the Workgroup name tree.

Return value

OrderedLocation
The newly created order location object.

Exceptions

CORBA standard exceptions and the following user exception:
IExtendedLifeCycle::InvalidScope.

Example

```
// C ++ example
// Note: for code clarity, some standard client setup, exception handling,
//  error checking and cleanup have been omitted from this example.


//**********************************************************************
//  This example uses Naming to find the default managed factory finder
//  for this host. It then uses the factory finder to find the specialized
//  homes for factory finders and ordered locations.  Next it uses the
//  ordered locatin home to create a new ordered location.  This is then
//  used with the factory finder factory to create a new factory finder.
//**********************************************************************
```

```
   extern IExtendedNaming::NamingContext_ptr root_context; // previously
                                                           // initialized
   ......

// get default factory

   cout << " get default factory finder \n";
   CORBA::Object_var temp_var = root_context->resolve_with_string(
                  "host/resources/factory-finders/host-scope");

   IExtendedLifeCycle::FactoryFinder_var  finder;
   finder = IExtendedLifeCycle::FactoryFinder::_narrow(temp_var);
   if (!finder)
   {
        cout << "Couldn't get default factory finder \n";
        exit(-1);
   }

// find factory finder factory

   cout << "get managed factory finder factory \n";
   CORBA::Object_var temp_obj = finder->find_factory_from_string(
      "ILifeCycleManagedClient::FactoryFinder.object interface");
   ILifeCycleManagedClient::FactoryFinderHome_var
      managed_factory_finder_factory;
   managed_factory_finder_factory =
      ILifeCycleManagedClient::FactoryFinderHome::_narrow(temp_obj  );
   if (!managed_factory_finder_factory)
   {
        cout << "Couldn't get managed factory finder factory \n";
        exit(-1);
   }


// Now get ordered location factory


   cout << "get managed ordered location factory \n";
   CORBA::Object_var temp_obj2 = finder->find_factory_from_string(
      "ILifeCycleManagedClient::OrderedLocation.object interface");
   ILifeCycleManagedClient::OrderedLocationHome_var  managed_ordered_loc;
   managed_ordered_loc =
      ILifeCycleManagedClient::OrderedLocationHome::_narrow(temp_obj2);
   if (!managed_ordered_loc)
   {
        cout << "Couldn't get managed ordered loc factory \n";
        exit(-1);
   }

// Create up your very own ordered location
// The scopes will specify server1, followed by server2
// This will cause only factories on either server to be found.

// create and initialize a orderedscope sequence
```

```
    IExtendedLifeCycle::OrderedScopes_var    scopes;
    scopes = new IExtendedLifeCycle::OrderedScopes(2);
    scopes->length(2);

    // first scope is for server1
    (*scopes)[0].cell = CORBA::string_dup("*LOCAL");
    (*scopes)[0].workgroup = CORBA::string_dup("*LOCAL");
    (*scopes)[0].host = CORBA::string_dup("*LOCAL");
    // specify server1 name
    (*scopes)[0].server = CORBA::string_dup("server1");
    (*scopes)[0].container = CORBA::string_dup("*ANY");
    (*scopes)[0].home = CORBA::string_dup("*ANY");

    // second scope is for server2
    (*scopes)[1].cell = CORBA::string_dup("*LOCAL");
    (*scopes)[1].workgroup = CORBA::string_dup("*LOCAL");
    (*scopes)[1].host = CORBA::string_dup("*LOCAL");
    // specify server2 name
    (*scopes)[1].server = CORBA::string_dup("server2");
    (*scopes)[1].container = CORBA::string_dup("*ANY");
    (*scopes)[1].home = CORBA::string_dup("*ANY");

    cout << "create new orderedlocation \n";
    IExtendedLifeCycle::OrderedLocation_var  new_ordered_loc;
    new_ordered_loc = managed_ordered_loc->createWithScopes(
         scopes,                     // The new scopes for this location.
         "MyTest_server_location",   // The name to register in the name
                                        space.
         0,                          // Do NOT register in cell name tree
         1,                          // Register in host name tree.
         0);                         // Do NOT register in workgroup name
                                        tree

    IExtendedLifeCycle::Location_var  new_loc;
    new_loc = IExtendedLifeCycle::Location::_narrow(new_ordered_loc);
    if (!new_loc)
    {
         cout << "Couldn't get new locfactory \n";
         exit(-1);
    }

// Use new location to create new factory finder

    cout << "create new factory finder \n";
    IExtendedLifeCycle::FactoryFinder_var  new_fac_find;
    new_fac_find = managed_factory_finder_factory->createWithLocation(
         new_loc,                  // The location created above.
         "MyTest_server_factory",  // The name to register in the name
                                      space.
         0,                        // Do NOT register in cell name tree
         1,                        // Register in host name tree.
         0);                       // Do NOT register in workgroup name
                                      tree

    if (!new_fac_find)
```

```
    {
         cout << "Couldn't get new fac_find \n";
         exit(-1);
    }

// Now new_fac_find can be used to find factories.
```

OrderedLocationHome::createWithScopeStrings

Creates a managed ordered location object and configures it with the scopes specified by the input scopestrings.

Original interface

IExtendedLifeCycle::OrderedLocationHome

IDL syntax

```
IExtendedLifeCycle::OrderedLocation createWithScopeStrings(
                        in OrderedScopeStrings scope_strings,
                        in CosNaming::Istring relativeName,
                        in boolean visibleInCellNameTree,
                        in boolean visisbleInHostNameTree,
                        in boolean visibleInWorkgroupNameTree);
   raises (IExtendedLifeCycle::InvalidScope,
           IExtendedLifeCycle::UnrecognizedScopeElement,
           IExtendedLifeCycle::IllegalStringSyntax,
           IExtendedLifeCycle::UnMatchedQuote);
```

Parameters

scopestrings
An ordered sequence of strings, each of which represents a acope structure, which will define the search order of for the location.

relativeName
The name under which the created ordered location is to be registered. If the name specified is an empty string, the ordered location will not be registered in the name space.

visibleInCellNameTree
If TRUE, the ordered location will be registered in the cell name tree.

visibleInHostNameTree
If TRUE, the ordered location will be registered in the host name tree.

visibleInWorkgroupNameTree
If TRUE, the ordered location will be registered in the Workgroup name tree.

Return value

OrderedLocation
The newly created order location object.

Exceptions

CORBA standard exceptions and the following user exceptions:
IExtendedLifeCycle::InvalidScope
IExtendedLifeCycle::Unrecognized ScopeElement
IExtendedLifeCycle::IlegalStringSyntax
IExtendedLifeCycle::UnmatchedQuote

Example

See example in “OrderedLocationHome::createWithScopes” on page 587.

OrderedLocationHome::createWithLocations

Creates a managed ordered location object and configures it with the input Location sequence.

Original interface

IExtendedLifeCycle::OrderedLocationHome

IDL syntax

```
IExtendedLifeCycle::OrderedLocation createWithLocations(
                        in SequenceOfLocations locations,
                        in CosNaming::Istring relativeName,
                        in boolean visibleInCellNameTree,
                        in boolean visisbleInHostNameTree,
                        in boolean visibleInWorkgroupNameTree);
```

Parameters

locations
An ordered sequence of location references which will define the search order of for the ordered location.

relativeName
The name under which the created ordered location is to be registered. If the name specified is an empty string, the ordered location will not be registered in the name space.

visibleInCellNameTree
If TRUE, the ordered location will be registered in the cell name tree.

visibleInHostNameTree
If TRUE, the ordered location will be registered in the host name tree.

visibleInWorkgroupNameTree
If TRUE, the ordered location will be registered in the Workgroup name tree.

Return value

OrderedLocation
The newly created ordered location object.

Remarks

CORBA standard exceptions.

Example

See example in “OrderedLocationHome::createWithScopes” on page 587.

ScopeManipulator Interface

This interface defines the operations used to convert between Scope structures and ScopeString structures.

Supported operations

ScopeManipulator::scope_to_string
ScopeManipulator::string_to_scope

ScopeManipulator::scope_to_string

Converts an IExtendedLifeCycle::Scope to string form.

Original interface

IExtendedLifeCycle::ScopeManipulator

IDL syntax

```
ScopeString scope_to_string(in Scope scope)
   raises(InvalidScope);
```

Parameters

IExtendedLifeCycle::Scope

Return value

IExtendedLifeCycle::ScopeString

Exceptions

CORBA standard exceptions and the following user exception:

IExtendedLifeCycle::InvalidScope The Scope parameter did not obey the rules for valid scopes. See “Defining scope of location” in the *Component Broker Advanced Programming Guide* for more information.

Example

```
// C ++ example
// Note: for code clarity, some exception handling, error checking and
//  cleanup have been ommitted from this example.

// create a local only ScopeManipulator

   ILifeCycleLocalObjectImpl::ScopeManipulator_var      scopeman;
   scopeman = ILifeCycleLocalObjectImpl::ScopeManipulator::_create();

// create and initialize a scope structure to specify MyServer server

   IExtendedLifeCycle::Scope    scope;
   scope.cell = CORBA::string_dup("*LOCAL");
   scope.workgroup = CORBA::string_dup("*LOCAL");
   scope.host = CORBA::string_dup("*LOCAL");
   scope.server = CORBA::string_dup("MyServer");  // specify my server name
   scope.container = CORBA::string_dup("*ANY");
   scope.home = CORBA::string_dup("*ANY");

// use the scope manipulator to convert the scope to a string and print it

   cout << " The scope is " << scopeman->scope_to_string(scope) << endl;
```

ScopeManipulator::string_to_scope

Converts a properly formed IExtendedLifeCycle::ScopeString to a scope. See “Defining scope of location” in the *Component Broker Advanced Programming Guide* for more information.

Original interface

IExtendedLifeCycle::ScopeManipulator

IDL syntax

```
Scope string_to_scope(in ScopeString scope_string)
   raises(InvalidScope,
          UnrecognizedScopeElement,
          IllegalStringSyntax,
          UnMatchedQuote);
```

Parameters

IExtendedLifeCycle::ScopeString

Return value

IExtendedLifeCycle::Scope

Exceptions

CORBA standard exceptions and the following user exceptions:

IExtendedLifeCycle::InvalidScope - the Scope parameter did not obey the rules for valid scopes. See “Defining scope of location” in the *Component Broker Advanced Programming Guide* for more information.

IExtendedLifeCycle::UnrecognizedScopeElement - the ScopeString parameter contains an element other than cell, workgroup, host, server, container, or home. See “Defining scope of location” in the *Component Broker Advanced Programming Guide* for more information.

IExtendedLifeCycle::IllegalStringSyntax - indicates that the ScopeString has a syntax error.

IExtendedLifeCycle::UnmatchedQuote - indicates that there is an unmatched quote in the ScopeString.

Example

```
// C ++ example
// Note: For code clarity, some exception handling, error checking and
// cleanup have been ommitted from this example.

#include <ILifeCycleLocalObjectImpl.hh>
#include <io.h>
#include <stdio.h>
#include <fstream.h>
#include <fcntl.h>
#include <sys/stat.h>
#include <orb.h>
#include <assert.h>
#include <stdlib.h>

int main(int argc, char * argv[])
{

// create a local only ScopeManipulator

   ILifeCycleLocalObjectImpl::ScopeManipulator_var      scopeman;
   scopeman = ILifeCycleLocalObjectImpl::ScopeManipulator::_create();

// use scope manipulator to convert string to scope

   IExtendedLifeCycle::Scope_var        scope;
   scope = scopeman->string_to_scope("MyServer.server/MyHost.host");

// examine the resulting scope

   cout << "Content of scope " << endl;
```

```
   cout << "      Cell : " << scope->cell << endl;
   cout << " Workgroup : " << scope->workgroup << endl;
   cout << "      Host : " << scope->host << endl;
   cout << "    Server : " << scope->server << endl;
   cout << " Container : " << scope->container << endl;
   cout << "      Home : " << scope->home << endl;

// expected result is
//      Cell : *LOCAL
// Workgroup : *LOCAL
//      Host : MyHost
//    Server : MyServer
// Container : *ANY
//      Home : *ANY

   cout << "GoodBye! "<< endl;

}
```

SingleLocation Interface

A FactoryFinder object encapsulates a Location object. The Location object is used to define the boundaries which the FactoryFinder is to limit itself to when locating factories. This information is contained in a sequence of scopes within the location object. SingleLocation is a specialization of the IExtendedLifeCycle::Location interface. A SingleLocation contains only a single scope, while a generic Location is allowed to contain multiple scopes. See “Defining scope of location” in the *Component Broker Advanced Programming Guide* for more information.

Supported operations

SingleLocation::get_scope

SingleLocation::get_scope

The SingleLocation object is used to define the boundaries which the FactoryFinder is to limit itself to when locating factories.

Original interface

IExtendedLifeCycle::SingleLocation

IDL syntax

```
Scope get_scope();
```

Return value

scope The SingleLocation’s scope.

Remarks

The SingleLocation object is used to define the boundaries which the FactoryFinder is to limit itself to when locating factories. This information is contained in a single scope within the SingleLocation object. See “Defining scope of location” in the *Component Broker Advanced Programming Guide* for more information. This operation returns the scope contained in the SingleLocation.

It may be useful to examine the scope to determine the extent of the FactoryFinder’s search area.

Example

```
// C ++ example
// Note: for code clarity, some standard client setup, exception handling,
//  error checking and cleanup have been ommitted from this example.

   extern IExtendedNaming::NamingContext_ptr root_context; // previously
                                                           // initialized

   extern CosLifeCycle::FactoryFinder_var  finder // previously initialized;

// get the location from the factory finder

   IExtendedLifeCycle::Location_var     loc;

   loc = finder->get_location();

// Narrow to a SingleLocation

   IExtendedLifeCycle::SingleLocation_var       single_loc;
   single_loc = IExtendedLifeCycle::SingleLocation::_narrow(loc);

// Now get the scope from the single location.

   IExtendedLifeCycle::Scope_var        scope;

   scope = single_loc->get_scope();

// Print out the scope

   cout << "Content of scope " << endl;

   cout << "      Cell : " << scope->cell << endl;
   cout << " Workgroup : " << scope->workgroup << endl;
   cout << "      Host : " << scope->host << endl;
   cout << "    Server : " << scope->server << endl;
   cout << " Container : " << scope->container << endl;
   cout << "      Home : " << scope->home << endl;
```

SingleLocationHome Interface

Provides a means for clients to create managed single locations configured with their own scope parameter.

Ancestor interfaces

IManagedClient::IHome

Supported operations

SingleLocationHome::createWithScope
SingleLocationHome::createWithScopeString

SingleLocationHome::createWithScope

Creates a managed single location object and configures it with the input Scope.

Original interface

IExtendedLifeCycle::SingleLocationHome

IDL syntax

```
IExtendedLifeCycle::SingleLocation createWithScope(
          in Scope scope,
          in CosNaming::Istring relativeName,
          in boolean visibleInCellNameTree,
          in boolean visisbleInHostNameTree,
          in boolean visibleInWorkgroupNameTree
          );
```

Parameters

scope Defines the boundaries fo the location's search area.

relativeName
The name under which the created single location is to be registered. If the name specified is an empty string, the single location will not be registered in the name space.

visibleInCellNameTree
If TRUE, the single location will be registered in the cell name tree.

visibleInHostNameTree
If TRUE, the single location will be registered in the host name tree.

visibleInWorkgroupNameTree
If TRUE, the single location will be registered in the Workgroup name tree.

Return value

SingleLocation
The newly created single location object.

Exceptions

CORBA standard exceptions.

Example

See “FactoryFinderHome::createWithLocation” on page 581.

SingleLocationHome::createWithScopeString

Creates a managed single location object and configures it with the scope specified by the input scopestring.

Original interface

IExtendedLifeCycle::SingleLocationHome

IDL syntax

```
IExtendedLifeCycle::SingleLocation createWithScopeString(
                    in ScopeString scope_string,
                    in CosNaming::Istring relativeName,
                    in boolean visibleInCellNameTree,
                    in boolean visisbleInHostNameTree,
                    in boolean visibleInWorkgroupNameTree
   );
```

Parameters

scopeString
The string form of a scope.

relativeName
The name under which the created single location is to be registered. If the name specified is an empty string, the single location will not be registered in the name space.

visibleInCellNameTree
If TRUE, the single location will be registered in the cell name tree.

visibleInHostNameTree
If TRUE, the single location will be registered in the host name tree.

visibleInWorkgroupNameTree
If TRUE, the single location will be registered in the Workgroup name tree.

Return value

SingleLocation
The newly created single location object.

Exceptions

CORBA standard exceptions.

Example

See “FactoryFinderHome::createWithLocation” on page 581.

Chapter 25. IExtendedNaming in the Naming Service

The other modules in the Naming Service are:
- CosNaming
- IExtendedNamingStringSyntax
- NamingStringSyntax

IExtendedNaming Module

Supports methods that allow the assigning of a name to an object (that is, creating an object-name binding in a context) and methods that find this object using the assigned name. Provides a more simple and familiar name format than the CORBA::Name structure.

File name

IExtendedNaming.idl

Intended usage

The IExtendedNaming::NamingContext Interface is very similar to the CosNaming::NamingContext Interface. That is, the behavior of the operations in both interfaces is the same. The only differences are in the names of the operations and in the type of the Name parameter in each operation. In the IExtendedNaming::NamingContext interface the type of the Name parameter is IExtendedNaming::Name. This is an IBM-defined interface.

Typically, without this module, a user of a naming context first converts a name from a string format to a CosNaming::Name format using the StandardSyntaxModel object (see “StandardSyntaxModel Interface” on page 769). Then, the user invokes an operation on the naming context using the converted name. The StandardSyntaxModel object needs to reside on the client side.

The IExtendedNaming Module combines the above two steps into a single step, and simplifies both client programming and client enablement. The IExtendedNaming::“NamingContext Interface” on page 607 is used in conjunction with the “Chapter 43. NamingStringSyntax in the Naming Service” on page 769 to simplify the use of a name when invoking operations on a naming context. The StandardSyntaxModel object becomes part of the server code making the task of client enablement easier.

Types

```
typedef CosNaming::Istring NameString;

struct BindingString {
  NameString binding_name;
  CosNaming::BindingType binding_type;
```

```
};

typedef sequence <BindingString> BindingStringList;
```

Interfaces

IExtendedNaming::BindingStringIterator Interface
IExtendedNaming::NamingContext Interface

BindingStringIterator Interface

The IExtendedNaming::BindingStringIterator Interface provides support for iterating through the list of bindings in a naming context.

File name

IExtendedNaming.idl

Intended usage

Operations in this interface behave in a similar way to the operations in the CosNaming::BindingIterator Interface. The only difference is in the type of binding and binding list. In this interface the binding is of type BindingString and the binding list is of type BindingStringList.

This interface is instantiated and returned as an out parameter in the IExtendedNaming::NamingContext::list_with_string Operation if the targeted naming context contains more name-object bindings than requested.

Types

```
typedef CosNaming::Istring NameString;

struct BindingString {
  NameString binding_name;
  CosNaming::BindingType binding_type;
};
typedef sequence <BindingString> BindingStringList;
```

IDL syntax

```
interface BindingStringIterator {
  void destroy();
  boolean next_one (out IExtendedNaming::BindingString binding);
  boolean next_n (in unsigned long how_many,
      out IExtendedNaming::BindingStringList blist);
};
```

Supported operations

BindingStringIterator::destroy
BindingStringIterator::next_n
BindingStringIterator::next_one

BindingStringIterator::destroy

Destroys the iterator and frees allocated memory.

Original interface

IExtendedNaming::BindingStringIterator Interface

IDL syntax

```
void destroy();
```

Exceptions

CORBA standard exceptions.

Remarks

This operation is intended to be used by client applications. It is not typically overridden.

Example

The following example demonstrates the usage of the IExtendedNaming Module.

```
// An IExtendedNaming usage example.
// For simplicity, error and exception checking and cleanup are ommitted.

  #include <IExtendedNaming.hh>
  #include <stdlib.h>
  #include <fstream.h>
  #define filename1 "NMUTST1.OUT"
  #define filename2 "NMUTST1.OUT"


  // Create a new naming context vehiculesNamingContext and bind it to
  // the root rootNamingContext with the name "vehicules"
  IExtendedNaming::NamingContext_ptr vehiculesNamingContext =
    rootNamingContext->bind_new_context_with_string("vehicules");


  // create a new naming context largeVehiculeNamingContext and bind it
  // to the naming context vehiculesNamingContext with the name
  // "vehicules.large"
  IExtendedNaming::NamingContext_ptr largeVehiculeNamingContext =
    vehiculesNamingContext->bind_new_context_with_string(
    "vehicules.large");
```

```
// create a new naming context vansNamingContext and bind it to the
// naming context vehiculesNamingContext with the name "vans"
IExtendedNaming::NamingContext_ptr vansNamingContext =
  vehiculesNamingContext->bind_new_context_with_string("vans");


// create a new naming context trucksNamingContext and bind it to the
// naming context vehiculesNamingContext with the name "trucks"
IExtendedNaming::NamingContext_ptr trucksNamingContext =
  vehiculesNamingContext->bind_new_context_with_string("trucks");


// Create an object aVehiculeObject
ifstream strm1(filename1);
char refStr[2048];
memset(refStr, 2048, '\0');
strm1 >> refStr;
CORBA::Object_ptr aVehiculeObject = orb_p->string_to_object(refStr);


// Bind the object aVehiculeObject to the naming context
// vansNamingContext with the name "chrysler"
vansNamingContext->bind_with_string("chrysler", aVehiculeObject);


// Create another object anotherVehiculeObject
ifstream strm2(filename2);
memset(refStr, 2048, '\0');
strm2 >> refStr;
CORBA::Object_ptr anotherVehiculeObject = orb_p->string_to_object(refStr);


// Rebind the object anotherVehiculeObject to the naming context
// vansNamingContext with the name "chrysler"
vansNamingContext->rebind_with_string("chrysler", anotherVehiculeObject);


// Bind the context vansNamingContext to the context
// vehiculesNamingContext with the name "vans"
largeVehiculeNamingContext->bind_context_with_string("vans",
  vansNamingContext);


// Rebind the context vansNamingContext to the context
// vehiculesNamingContext with the name "vans.mini"
largeVehiculeNamingContext->rebind_context_with_string("vans.mini",
  vansNamingContext);


// Unbind the object bound to vehiculesNamingContext with the name
// "vans"
largeVehiculeNamingContext->unbind_with_string("vans");
```

```
// Unbind the object bound to vehiculesNamingContext with the name
// "vans.mini"
largeVehiculeNamingContext->unbind_with_string("vans.mini");


// Resolve the name from the root naming context rootNamingContext
aVehiculeObject =
  rootNamingContext->resolve_with_string("vehicules/vans/chrysler");


// list only one binding in the naming root context rootNamingContext
// The remaining bindings can be retrieved from the binding iterator bi.
IExtendedNaming::BindingStringList_var bl;
IExtendedNaming::BindingStringIterator_var bi;
vehiculesNamingContext->list_with_string(1, bl, bi);


// Retrieve the next binding from the binding iterator
IExtendedNaming::BindingString_var b;
bi->next_one(b);


// Retrieve the next 2 bindings from the binding iterator
IExtendedNaming::BindingStringList_var bl1;
bi->next_n(2, bl1);


// Destroy the naming context vehiculesNamingContext
largeVehiculeNamingContext->destroy();


// Destroy the binding iterator bi
bi->destroy();
```

BindingStringIterator::next_n

Retrieves at most the specified number of name-object bindings. This operation allows clients to iterate through the bindings in the iterator.

Original interface

IExtendedNaming::BindingStringIterator Interface

IDL syntax

```
boolean next_n (
  in unsigned long how_many,
  out IExtendedNaming::BindingStringList blist);
```

Parameters

how_many
The maximum number of bindings to be returned.

blist The returned BindingStringList.

Return value

TRUE Indicates more bindings exist.

FALSE Indicates that there are no more bindings.

Exceptions

CORBA standard exceptions.

Remarks

This operation is intended to be used by client applications. It is not typically overridden.

The next_n operation returns the maximum number of bindings in the blist parameter. Use the next_n operation to iterate through the bindings. This operation will return fewer bindings if less than how_many bindings remain in the iterator.

Example

See the IExtendedNaming usage example for “BindingStringIterator::destroy” on page 603.

BindingStringIterator::next_one

Retrieves the next name-object binding.

Original interface

IExtendedNaming::BindingStringIterator Interface

IDL syntax

```
boolean next_one (out IExtendedNaming::BindingString binding);
```

Parameters

binding
The returned BindingString.

Return value

TRUE Indicates that bindings exist.

FALSE Indicates that there are no more bindings.

Exceptions

CORBA standard exceptions.

Remarks

This operation is intended to be used by client applications. It is not typically overridden.

Example

See the IExtendedNaming usage example for “BindingStringIterator::destroy” on page 603.

NamingContext Interface

Provides the operations necessary to create and manipulate a system naming tree, to bind a name to an object in a naming context, to retrieve an object from a naming context using the object name, and to list the bindings in a naming context.

This is an IBM-defined interface.

File name

IExtendedNaming.idl

Intended usage

The behavior of operations in the IExtendedNaming::NamingContext Interface is similar to the behavior of operations in the CosNaming::NamingContext Interface. The IExtendedNaming::NamingContext interface, however, simplifies the use of names by defining a name to be of a string type.

Ancestor interfaces

CosNaming::NamingContext Interface

Exceptions

CORBA standard exceptions and the following user exceptions:

CosNaming::NamingContext::AlreadyBound - raised to indicate that an object is already bound to the name. Re-binding operations unbind the name, then rebinds the name without raising this exception.

CosNaming::NamingContext::CannotProceed{NamingContext ctx; Name rest_of_name;}; - raised to indicate that the implementation has given up for some reason. The client may be able to continue the operation using the returned naming context.

CosNaming::NamingContext::InvalidName - raised to indicate that the name is invalid. A name with a length of zero is invalid. (This exception may be raised upon further implementation restrictions.)

CosNaming::NamingContext::NotFound{NotFoundReason why; Name rest_of_name;}; - raised to indicate that the name does not identify a binding. If a compound name is passed as an argument for the bind operation, it traverses multiple contexts. A NotFound exception is raised if any of the intermediate contexts cannot be resolved.

Supported operations

NamingContext::bind_with_string
NamingContext::bind_context_with_string
NamingContext::bind_new_context_with_string
NamingContext::list_with_string
NamingContext::rebind_with_string
NamingContext::rebind_context_with_string
NamingContext::resolve_with_string
NamingContext::unbind_with_string

NamingContext::bind_with_string

Creates a binding in a naming context.

Original interface

IExtendedNaming::NamingContext Interface

IDL syntax

```
void bind_with_string(
  in IExtendedNaming::NameString name,
  in Object obj);
```

Parameters

name The name for the binding.

obj The Object to be bound.

Exceptions

CORBA standard exceptions and the following user exceptions:

CosNaming::NamingContext::NotFound - raised to indicate that the name does not identify a binding. If a compound name is passed as an argument for the

bind_with_string operation, it traverses multiple contexts. A NotFound exception is raised if any of the intermediate contexts cannot be resolved.

CosNaming::NamingContext::CannotProceed - raised to indicate that the implementation has given up for some reason. The client may be able to continue the operation using the returned naming context.

CosNaming::NamingContext::InvalidName - raised to indicate that the name is invalid. A name with a length of zero is invalid. (This exception may be raised upon further implementation restrictions.)

CosNaming::NamingContext::AlreadyBound - raised to indicate that an object is already bound to the name. Re-binding operations unbind the name, then rebinds the name without raising this exception.

Remarks

This operation is intended to be used by client applications. It is not typically overridden.

Creates a binding of a name to an object in a naming context. Binding a name to an object in a naming context creates a name-object association relative to the target naming context. Once an object is bound, it can be found through the resolve_with_string operation. Naming contexts that are bound using bind_with_string do not participate in name resolution when compound names are resolved - bind_context_with_string should be used to bind naming context objects. This operation runs resolve_with_string to traverse a compound name. An object can be bound to multiple names in a context or across multiple contexts. Within a context, names of an object must be unique. That is, only one object can be bound to a particular name in a naming context.

Example

See the IExtendedNaming usage example for “BindingStringIterator::destroy” on page 603.

NamingContext::bind_context_with_string

Creates a naming context binding.

Original interface

IExtendedNaming::NamingContext Interface

IDL syntax

```
void bind_context_with_string(
  in IExtendedNaming::NameString name,
  in IExtendedNaming::NamingContext naming_context);
```

Parameters

name The name for the binding.

naming_context
The naming context object to be bound.

Exceptions

CORBA standard exceptions and the following user exceptions:

CosNaming::NamingContext::NotFound - raised to indicate that the name does not identify a binding. If a compound name is passed as an argument for the bind operation, it traverses multiple contexts. A NotFound exception is raised if any of the intermediate contexts cannot be resolved.

CosNaming::NamingContext::CannotProceed - raised to indicate that the implementation has given up for some reason. The client may be able to continue the operation using the returned naming context.

CosNaming::NamingContext::InvalidName - raised to indicate that the name is invalid. A name with a length of zero is invalid. (This exception may be raised upon further implementation restrictions.)

CosNaming::NamingContext::AlreadyBound - raised to indicate that an object is already bound to the name. Re-binding operations unbind the name, then rebind the name without raising this exception.

Remarks

This operation is intended to be used by client applications. It is not typically overridden.

This operation creates a naming context binding. Binding a name and a naming context object into a naming context creates a name-object association relative to the target naming context. Naming contexts that are bound using bind_context_with_string participate in name resolution when compound names are resolved. This operation is used to extend the naming tree by binding sub-contexts to contexts. Like an object, a naming context can be bound, using bind_context_with_string, to multiple names in a context or across multiple contexts. Within a context, the names bound to a context must be unique. That is, only one context can be bound to a particular name in a naming context.

Example

See the IExtendedNaming usage example for “BindingStringIterator::destroy” on page 603.

NamingContext::bind_new_context_with_string

Creates a new naming context in the same server as the target naming context on which the operation was invoked and binds it to a supplied name.

Original interface

IExtendedNaming::NamingContext Interface

IDL syntax

```
IExtendedNaming::NamingContext bind_new_context(
   in IExtendedNaming::NameString name);
```

Parameters

name The name for the naming context object binding.

Return value

IExtendedNaming::NamingContext
New context bound to the supplied name.

Exceptions

CORBA standard exceptions and the following user exceptions:

CosNaming::NamingContext::NotFound - raised to indicate that the name does not identify a binding. If a compound name is passed as an argument for the bind_with_context operation, it traverses multiple contexts. A NotFound exception is raised if any of the intermediate contexts cannot be resolved.

CosNaming::NamingContext::CannotProceed - raised to indicate that the implementation has given up for some reason. The client may be able to continue the operation using the returned naming context.

CosNaming::NamingContext::InvalidName - raised to indicate that the name is invalid. A name with a length of zero is invalid. (This exception may be raised upon further implementation restrictions.)

CosNaming::NamingContext::AlreadyBound - raised to indicate that an object is already bound to the name.

Remarks

This operation is intended to be used by client applications. It is not typically overridden.

This operation creates a new naming context in the same process as the target naming context on which the operation was invoked and binds it to a supplied name. The new naming context has the same implementation as the target naming context to which it is bound. This new context is created in the same process as that of the target naming context. Note that the target naming context in which the new context is bound is denoted by a name that is equivalent to the name name excluding the last name component of name.

Example

See the IExtendedNaming usage example for “BindingStringIterator::destroy” on page 603.

NamingContext::list_with_string

Retrieves the bindings from a naming context.

Original interface

IExtendedNaming::NamingContext Interface

IDL syntax

```
void list_with_string(
  in unsigned long how_many,
  out IExtendedNaming::BindingStringList blist,
  out IExtendedNaming::BindingStringIterator biterator);
```

Parameters

how_many
The maximum number bindings to install into the BindingStringList.

blist The returned BindingStringList.

biterator
The returned BindingStringIterator.

Exceptions

CORBA standard exceptions.

Remarks

This operation is intended to be used by client applications. It is not typically overridden.

This operation retrieves bindings from a naming context. The operation returns the number of bindings; equal, at most, to how_many in blist. If the naming context contains additional bindings, an IExtendedNaming::BindingStringIterator is returned, and the calling program can iterate through the remaining bindings. If the naming context does not contain additional bindings, the IExtendedNaming::BindingStringIterator is a NIL object reference.

The value of how_many should be less than or equal to a maximum of 1000.

The returned binding list is of type IExtendedNaming::BindingStringList which contains a list of bindings. Each element in the list is of type IExtendedNaming::BindingString.

IExtendedNaming::BindingString consists of two fields: binding_name which is the name part of the binding and binding_type which is the type of the object part of the binding. A binding type is either an object (nobject) or a naming context (ncontext).

Example

See the IExtendedNaming usage example for “BindingStringIterator::destroy” on page 603.

NamingContext::rebind_with_string

Recreates a binding in a naming context even if the name is already bound in the naming context.

Original interface

IExtendedNaming::NamingContext Interface

IDL syntax

```
void rebind_with_string(
  in IExtendedNaming::NameString name,
  in Object obj);
```

Parameters

name The name to be re-bound.

obj The Object to be re-bound.

Exceptions

CORBA standard exceptions and the following user exceptions:

CosNaming::NamingContext::NotFound - raised to indicate that the name does not identify a binding. If a compound name is passed as an argument for the bind operation, it traverses multiple contexts. A NotFound exception is raised if any of the intermediate contexts cannot be resolved.

CosNaming::NamingContext::CannotProceed - raised to indicate that the implementation has given up for some reason. The client may be able to continue the operation using the returned naming context.

CosNaming::NamingContext::InvalidName - raised to indicate that the name is invalid. A name with a length of zero is invalid. (This exception may be raised upon further implementation restrictions.)

Remarks

This operation is intended to be used by client applications. It is not typically overridden.

This operation recreates a name binding in a naming context, even if the name is already bound in the naming context. Rebinding a name and object into a naming context recreates a name-object association relative to the target naming context. Naming contexts that are bound using rebind_with_string do not participate in the name resolution process when compound names are resolved.

If an object is already bound with the same name, the bound object is replaced by the passed argument obj. If the name-object binding does not exist, the rebind_with_string method behaves like the bind_with_string method.

As a developer, you can use the rebind_with_string method to replace an existing binding. Use this operation instead of the unbind_with_string and bind_with_string methods.

Example

See the IExtendedNaming usage example for “BindingStringIterator::destroy” on page 603.

NamingContext::rebind_context_with_string

Recreates a binding to a naming context, even if the name is already bound in the naming context.

Original interface

IExtendedNaming::NamingContext Interface

IDL syntax

```
void rebind_context_with_string(
  in IExtendedNaming::NameString name,
  in IExtendedNaming::NamingContext naming_context);
```

Parameters

name The name to be re-bound.

naming_context
The NamingContext object to be re-bound to the name.

Exceptions

CORBA standard exceptions and the following user exceptions:

CosNaming::NamingContext::NotFound - raised to indicate that the name cannot be resolved into a naming context to perform binding. If a compound name is passed as an argument for the bind operation, it traverses multiple contexts. A NotFound exception is raised if any of the intermediate contexts cannot be resolved.

CosNaming::NamingContext::CannotProceed - raised to indicate that the implementation has given up for some reason. The client may be able to continue the operation using the returned naming context.

CosNaming::NamingContext::InvalidName - raised to indicate that the name is invalid. A name with a length of zero is invalid. (This exception may be raised upon further implementation restrictions.)

Remarks

This operation is intended to be used by client applications. It is not typically overridden.

This operation recreates a binding to a naming context, even if the name is already bound in the naming context. Re-binding a name and a naming context object into a naming context recreates a name-object association relative to the target naming context. Naming contexts that are bound using rebind_context_with_string participate in name resolution when compound names are resolved.

This operation is used to bind or replace a subcontext. If a context is already bound in a context, the bind_with_string operation raises the AlreadyBound exception. However, the rebind_with_string method replaces the bound object with the passed object.

Example

See the IExtendedNaming usage example for “BindingStringIterator::destroy” on page 603.

NamingContext::resolve_with_string

Retrieves an Object bound to a name.

Original interface

IExtendedNaming::NamingContext Interface

IDL syntax

```
Object resolve_with_string(in IExtendedNaming::NameString name);
```

Parameters

name The name for the name-object binding.

Return value

Object The Object bound to the supplied name.

Exceptions

CORBA standard exceptions and the following user exceptions:

CosNaming::NamingContext::NotFound - raised to indicate that the name does not identify a binding. If a compound name is passed as an argument for the bind operation, it traverses multiple contexts. A NotFound exception is raised if any of the intermediate contexts cannot be resolved.

CosNaming::NamingContext::CannotProceed - raised to indicate that the implementation has given up for some reason. The client may be able to continue the operation using the returned naming context.

CosNaming::NamingContext::InvalidName - raised to indicate that the name is invalid. A name with a length of zero is invalid. (This exception may be raised upon further implementation restrictions.)

Remarks

This operation is intended to be used by client applications. It is not typically overridden.

This operation retrieves the object bound to name *name* in the target naming context. The name *name* could be a simple name. In that case *name* should match exactly the name bound to the object in the context. Or, the name *name* could be a compound name that spans multiple contexts. In this case, name resolution traverses multiple contexts. At each context traversed, the *name* bound to this context, in its super context, should match exactly the name component corresponding to this traversed context. The last name component of the compound name should match exactly the name bound to the object in the last traversed naming context.

The type of the returned object object is not provided. Clients are responsible for narrowing the object to the appropriate type. Clients typically cast the returned object to a more specialized interface

Example

See the IExtendedNaming usage example for “BindingStringIterator::destroy” on page 603.

NamingContext::unbind_with_string

Removes a name-object binding.

Original interface

IExtendedNaming::NamingContext Interface

IDL syntax

```
void unbind_with_string(in IExtendedNaming::NameString name);
```

Parameters

name The name for the name-object binding.

Exceptions

CORBA standard exceptions and the following user exceptions:

CosNaming::NamingContext::NotFound - raised to indicate that the name does not identify a binding. If a compound name is passed as an argument for the bind_with_string operation, it traverses multiple contexts. A NotFound exception is raised if any of the intermediate contexts cannot be resolved.

CosNaming::NamingContext::CannotProceed - raised to indicate that the implementation has given up for some reason. The client may be able to continue the operation using the returned naming context.

CosNaming::NamingContext::InvalidName - raised to indicate that the name is invalid. A name with a length of zero is invalid. (This exception may be raised upon further implementation restrictions.)

Remarks

This operation is intended to be used by client applications. It is not typically overridden.

This operation removes a binding from a context. This operation unbinds the name *name* from the context. It is used to unregister the name *name* with the Naming Service.

This operation can also be used to unbind a naming context. If the naming context was originally bound using bind_context, rebind_context, bind, or rebind, the operation will be allowed to proceed. However, if this context was originally bound using bind_new_context, then a CORBA::PERSIST_STORE exception will be thrown since this request would result in an orphaned name context (which is not supported). In the case of the CORBA::PERSIST_STORE exception, the user is required to call the destroy() method to unbind the name context.

Example

See the IExtendedNaming usage example for “BindingStringIterator::destroy” on page 603.

Chapter 26. IExtendedNamingStringSyntax in the Naming Service

The other modules in the Naming Service are:
- CosNaming
- IExtendedNaming
- NamingStringSyntax

IExtendedNamingStringSyntax Module

Defines the local-only object implementation of the StandardSyntaxModel helper object.

Supported on AIX and Windows NT only

File name

IExtendedNamingStringSyntax.idl

Intended usage

This module defines the implementation interface of the local-only StandardSyntaxModel helper object whose interface is defined in the NamingStringSyntax module.

Interfaces

IExtendedNamingStringSyntax::StandardSyntaxModel Interface

StandardSyntaxModel Interface

This interface introduces new operations which create and initialize loca-only instances of StandardSyntaxModel objects. These instances are non-remotable and non-managed. They can be created from C++ client applications and by C++ and Java business objects.

Local-only

True

Supported operations

StandardSyntaxModel::_create

StandardSyntaxModel::_create

Creates a local-only standard syntax model object and initializes its attributes to the default syntax model.

Original interface

IExtendedNamingStringSyntax::StandardSyntaxModel Interface

C++ syntax

```
_create();
```

Return value

IExtendedNamingStringSyntax::StandardSyntaxModel

Remarks

The standard syntax model attributes are set to the values which define the default syntax model, which are:

```
syntax_direction = kLeftToRight
syntax_absolute_prefix = "/", "\"
syntax_reserved_names = ".:", "..."
syntax_delimiter = "/", "\"
syntax_separator = "."
syntax_escape = "\"
syntax_begin_quote = """(double quote), """(single quote)
syntax_end_quote = """(double quote), """(single quote)
syntax_code_set = kISOLatin1
syntax_locale_info = kUS_ENG
```

Example

```
IExtendedNamingStringSyntax::StandardSyntaxModel_var anSSM;
anSSM = IExtendedNamingStringSyntax::StandardSyntaxModel::_create();
```

Chapter 27. IExtendedQuery in the Query Service

The other modules in the Query Service are:
- CosQuery
- CosQueryCollection

IExtendedQuery Module

Defines the IDL for query types related to the Query service.

File name

IExtendedQuery.idl

Types

```
typedef sequence<IManagedClient::IManageable> MemberList;
typedef sequence<any> DataArray;
typedef sequence<DataArray> DataArrayList;
```

Interfaces

IExtendedQuery::DataArrayDescriptor Interface
IExtendedQuery::DataArrayIterator Interface
IExtendedQuery::ParameterListBuilder Interface
IExtendedQuery::QueryEvaluator Interface

DataArrayDescriptor Interface

The DataArrayDescriptor is an abstract interface supported by the DataArrayIterator Interface.

File name

IExtendedQuery.idl

Intended usage

The DataArrayDescriptor is used to retrieve metadata about the contents of the DataArray contained in a query result; specifically, the names and types of fields in the DataArray. A DataArray is a sequence of anys.

This interface is separated from the DataArrayIterator to allow it to be used with other types of iterators where appropriate.

IDL syntax

```
interface DataArrayDescriptor {

     unsigned long get_number_of_fields();
     string get_field_name(
            in unsigned long position)
        raises(PositionInvalid);
     CORBA::TCKind get_field_type(
            in unsigned long position)
        raises(PositionInvalid);
     string get_field_class_name(
            in unsigned long position)
        raises(PositionInvalid, FieldNotObjectType);
};
```

Supported operations

DataArrayDescriptor::get_number_of_fields
DataArrayDescriptor::get_field_name
DataArrayDescriptor::get_field_type
DataArrayDescriptor::get_field_class_name

DataArrayDescriptor::get_number_of_fields

Returns the number of fields in the DataArray structure. DataArray is the data type contained in a query result generated by QueryEvaluator::evaluate_to_data_array operation.

Original interface

IExtendedQuery::DataArrayDescriptor Interface

IDL syntax

```
unsigned long get_number_of_fields()
```

Return value

long The number of fields in the DataArray structure.

DataArrayDescriptor::get_field_name

Returns the string name of a DataArray field.

Original interface

IExtendedQuery::DataArrayDescriptor Interface

IDL syntax

```
string get_field_name(in unsigned long position)
     raises(PositionInvalid)
```

Parameters

position
The position parameter must be 0 <= position < number_of_fields.

Return value

string Each position in a DataArray is called a field, and each field has a string name. get_field_name returns the string name of the field.

Exceptions

IExtendedQuery::PositionInvalid - The position parameter is invalid. The position parameter must be 0 <= position < number_of_fields.

DataArrayDescriptor::get_field_type

Returns the type of the field.

Original interface

IExtendedQuery::DataArrayDescriptor Interface

IDL syntax

```
TCKind get_field_type(in unsigned long position)
   raises(PositionInvalid)
```

Parameters

position
The position parameter must be 0 <= position < number_of_fields.

Return value

TCKind
Each position in a DataArray is called a field. get_field_type returns the type of the field.

Exceptions

IExtendedQuery::PositionInvalid - The position parameter is invalid. The position parameter must be 0 <= position < number_of_fields.

DataArrayDescriptor::get_field_class_name

Returns the class name of the field if the field is an object type.

Original interface

IExtendedQuery::DataArrayDescriptor Interface

IDL syntax

```
string get_field_class_name(in unsigned long position)
   raises(PositionInvalid, FieldNotObjectType)
```

Parameters

position
The position parameter must be 0 <= position < number_of_fields.

Return value

string Each position in a DataArray is called a field. get_field_class_name returns the class name of the field if the field is an object type.

Exceptions

IExtendedQuery::PositionInvalid - The position parameter is invalid. The position parameter must be 0 <= position < number_of_fields.
IExtendedQuery::FieldNotObjectType - The field is not an object type.

DataArrayIterator Interface

The DataArrayIterator interface defines the return type of evaluate_to_data_array method. Result of a query can be accessed using methods defined by this interface.

Note: The collating sequence depends on NLS settings on the Component Broker Server system.

File name

IExtendedQuery.idl

Intended usage

The DataArrayIterator interface is used to access the result of a full OOSQL query submitted using the evaluate_to_data_array interface. Similarly to other iterators, each successive element in a data array is accessed using the next_one or next_n operation. Each element returned by the iterator is a sequence of CORBA::Any's that correspond to the projection list of the full OOSQL query. The DataArrayIterator Interface allows users to retrieve the name and data type of a data array column, aside from its value.

IDL syntax

```
interface DataArrayIterator :
      DataArrayDescriptor {
```

```
    boolean next_one(out DataArray next_row);
    boolean nextOne(out DataArray next_row);
    DataArray next()
      raises (IInvalidIterator);
    boolean next_n(in unsigned long how_many,
                   out DataArrayList data_array_rows);
    boolean nextN(in unsigned long how_many,
                  out DataArrayList data_array_rows);
    boolean nextS(in unsigned long how_many,
                  out DataArrayList data_array_rows);
      raises (IInvalidIterator);
};
```

Supported operations

DataArrayIterator::next
DataArrayIterator::nextN
DataArrayIterator::next_n
DataArrayIterator::nextOne
DataArrayIterator::next_one
DataArrayIterator::nextS

DataArrayIterator::next

Retrieves the next DataArray in the query result.

Original interface

IExtendedQuery::DataArrayIterator Interface

IDL syntax

```
boolean next(out DataArray next_row)
  raises (IInvalidIterator);
```

Parameters

next_row
Returns the next data array of type DataArray from the query result.

Return value

Returns true (1) if a data array was placed in the output parameter or false (0) if the iterator is currently on the last element (not past the end yet).

Exceptions

IInvalidIterator
The iterator is invalid. An exception code, substitution variables and a string variable providing text explanation of the error are contained in the exception.

DataArrayIterator::nextN

Retrieves at most the next how_many DataArrays from the query result.

Original interface

IExtendedQuery::DataArrayIterator Interface

IDL syntax

```
boolean next_N(in unsigned long how_many,
               out DataArrayList data_array_rows);
```

Parameters

how_many
An unsigned long value for the number of elements to be retrieved into the output parameter.

data_array_rows
This list of type DataArrayList is filled with the requested data array elements of DataArray.

Return value

Returns true (1) if the requested number of elements were placed in the output parameter, or false (0) if the iterator is invalid or less than the requested number of elements were placed in the output parameter.

DataArrayIterator::next_n

Retrieves at most the next how_many DataArrays from the query result.

Original interface

IExtendedQuery::DataArrayIterator Interface

IDL syntax

```
boolean next_n(in unsigned long how_many,
          out DataArrayList data_array_rows);
```

Parameters

how_many
An unsigned long value for the number of elements to be retrieved into the output parameter

data_array_rows
This list of type DataArrayList is filled with the requested data array elements of DataArray.

Return value

Returns true (1) if the requested number of elements were placed in the output parameter, or false (0) if the iterator is invalid or less than the requested number of elements were placed in the output parameter.

DataArrayIterator::nextOne

Retrieves the next DataArray in the query result.

Original interface

IExtendedQuery::DataArrayIterator Interface

IDL syntax

```
boolean nextOne(out DataArray next_row);
```

Parameters

next_row
Returns the next data array of type DataArray from the query result.

Return value

Returns true (1) if an element was placed in the output parameter or false (0) if the iterator is invalid.

DataArrayIterator::next_one

Retrieves the next DataArray in the query result.

Original interface

IExtendedQuery::DataArrayIterator Interface

IDL syntax

```
boolean next_one(out DataArray next_row);
```

Parameters

next_row
Returns the next data array of type DataArray from the query result.

Return value

Returns true (1) if an element was placed in the output parameter or false (0) if the iterator is invalid.

DataArrayIterator::nextS

Retrieves at most the next how_many DataArrays from the query result.

Original interface

IExtendedQuery::DataArrayIterator Interface

IDL syntax

```
boolean nextS(in unsigned long how_many,
              out DataArrayList data_array_rows);
```

Parameters

how_many
An unsigned long value for the number of elements to be retrieved into the output parameter.

data_array_rows
This list of type DataArrayList is filled with the requested data array elements of DataArray.

Return value

Returns true (1) if the requested number of elements were placed in the output parameter, or false (0) if the iterator is invalid or less than the requested number of elements were placed in the output parameter.

Exceptions

IInvalidIterator
The iterator is invalid. An exception code, substitution variables and a string variable providing text explanation of the error are contained in the exception.

ParameterListBuilder Interface

Used as a helper object for constructing IExtendedQuery::ParameterList. The ParameterListBuilder interface provides convenience operations for assigning type-specific data values to named parameters in the list.

File name

IExtendedQuery.idl

Intended usage

The ParameterListBuilder is used to construct a parameter list of collection names with collection references. The parameter list is submitted with an OOSQL query. The collection names that are in the parameter list can appear in the ″from″ clause of an OOSQL query. Collection names are first matched against the parameter list, then against the names space managed by the naming service. The ParameterListBuilder can also be used to construct a parameter list of host variables. This parameter list is also submitted along with an OOSQL. Host variables appearing in the query are matched against this parameter list.

IDL syntax

```
interface ParameterListBuilder {

      void clear_parm_list();
      void add_string_parm(
            in string parm_name,
            in string parm_value);
      void add_long_parm(
            in string parm_name,
            in long parm_value);
      void add_float_parm(
            in string parm_name,
            in float parm_value);
      void add_double_parm(
            in string parm_name,
            in double parm_value);
      void add_object(
            in string parm_name,
            in Object parm_value);
      void remove_parm(
            in string parm_name)
         raises(ParameterNotFound);
      CosQuery::ParameterList get_parm_list();
};
```

Supported operations

ParameterListBuilder::add_double_parm
ParameterListBuilder::add_float_parm
ParameterListBuilder::add_long_parm
ParameterListBuilder::add_object_parm
ParameterListBuilder::add_string_parm
ParameterListBuilder::clear_parm_list
ParameterListBuilder::get_parm_list
ParameterListBuilder::remove_parm

ParameterListBuilder::add_double_parm

Appends an NVpair at the end of the ParameterList. The name in the NVpair is set to parm_name and the value of the NVpair is set to an any value constructed by setting any._type = double and any._value = parm_value.

Original interface

IExtendedQuery::ParameterListBuilder Interface

IDL syntax

```
void add_double_parm(in string parm_name, in double parm_value)
```

Parameters

parm_name
The name in the NVpair.

parm_value
Used to construct the value of the NVpair. The value of the NVpair is an any constructed by setting any._type = double and any._value = parm_value

ParameterListBuilder::add_float_parm

Appends an NVpair at the end of the ParameterList. The name in the NVpair is set to parm_name and the value of the NVpair is set to an any value constructed by setting any._type = float and any._value = parm_value.

Original interface

IExtendedQuery::ParameterListBuilder Interface

IDL syntax

```
void add_float_parm(in string parm_name, in float parm_value)
```

Parameters

parm_name
The name in the NVpair.

parm_value
Used to construct the value of the NVpair. The value of the NVpair is an any constructed by setting any._type = float and any._value = parm_value

ParameterListBuilder::add_long_parm

Appends an NVpair at the end of the ParameterList. The name in the NVpair is set to parm_name and the value of the NVpair is set to an any value constructed by setting any._type = long and any._value = parm_value.

Original interface

IExtendedQuery::ParameterListBuilder Interface

IDL syntax

```
void add_long_parm(in string parm_name, in long parm_value)
```

Parameters

parm_name
The name in the NVpair.

parm_value
Used to construct the value of the NVpair. The value of the NVpair is an any constructed by setting any._type = long and any._value = parm_value

ParameterListBuilder::add_object_parm

Appends an NVpair at the end of the ParameterList. The name in the NVpair is set to parm_name and the value of the NVpair is set to an any value, constructed by setting any._type = string and any._value = object_to_string(parm_value). The reason for storing stringified object reference instead of object reference itself is for passing the object reference from client to server.

Original interface

IExtendedQuery::ParameterListBuilder Interface

IDL syntax

```
void add_object_parm(in string parm_name, in Object parm_value)
```

Parameters

parm_name
The name in the NVpair.

parm_value
Used to construct the value of the NVpair. The value of the NVpair is an any constructed by setting any._type = string and any._value = parm_value

ParameterListBuilder::add_string_parm

Appends an NVpair at the end of the ParameterList. The name in the NVpair is set to parm_name and the value of the NVpair is set to an any value constructed by setting any._type = string and any._value = parm_value.

Original interface

IExtendedQuery::ParameterListBuilder Interface

IDL syntax

```
void add_string_parm(in string parm_name, in string parm_value)
```

Parameters

parm_name
The name in the NVpair.

parm_value
Used to construct the value of the NVpair. The value of the NVpair is an any constructed by setting any._type = string and any._value = parm_value

ParameterListBuilder::clear_parm_list

Empties the ParameterList.

Original interface

IExtendedQuery::ParameterListBuilder Interface

IDL syntax

```
void clear_parm_list()
```

ParameterListBuilder::get_parm_list

Returns the ParameterList.

Original interface

IExtendedQuery::ParameterListBuilder Interface

IDL syntax

```
CosQueryCollection::ParameterList get_parm_list()
```

Return value

CosQueryCollection::ParameterList

ParameterListBuilder::remove_parm

Removes the NVpair with the name equal to parm_name from the ParameterList.

Original interface

IExtendedQuery::ParameterListBuilder Interface

IDL syntax

```
void remove_parm(in string parm_name) raises(ParameterNotFound)
```

Parameters

parm_name
The name of the parameter in the ParameterList.

Exceptions

IExtendedQuery::ParameterNotFound - The NVpair with name equal to parm_name is not find in the ParameterList.

QueryEvaluator Interface

Introduces distinct methods for producing an iterator over a collection of objects, an iterator over a collection of data arrays, and an iterator over a collection of objects starting from a queryable collection, respectively.

Note: The collating sequence depends on NLS settings on the Component Broker Server system.

File name

IExtendedQuery.idl

Intended usage

The QueryEvaluator interfaces are used to submit OOSQL queries to the CBC server. Results are returned in the form of iterators that range over the results of a query. The evaluate_to_data_array is the most general query operation. It accepts a full OOSQL query and returns a DataArrayIterator as result.

The evaluate_to_iterator operation accepts a restricted form of OOSQL query. The projection list of an OOSQL query is restricted to a single element of object type. The result of this interface is an iterator that returns objects, instead of the more complex data arrays.

The evaluate_collection operation is a helper operation for the evaluate method on collections. The evaluate_collection interface accepts a query fragment that corresponds to the ″where″ clause of an OOSQL query. For example ″sal > 10000″ can be used against the ″Emp″ collection to retrieve employees with a salary greater than 10000. This form of query fragment is a syntactic convenience as opposed to a new query syntax. These predicates are converted into OOSQL queries by prefixing them with the string ″select x from aCollection x where″ (e.g., select x from aCollection x where sal > 10000). The name ″aCollection″ is bound the collection against which the predicate is to be evaluated using the ParameterListBuilder Interface. The evaluate_to_iterator operation is then used to submit the query for evaluation.

IDL syntax

```
interface QueryEvaluator :
            CosQuery::QueryEvaluator {

  void evaluate_to_iterator(in string query, in QLType ql_type,
    in ParameterList collection_names,
    in ParameterList params,
    in unsigned long how_many,
    out MemberList members,
    out IManagedCollections::IIterator members_iterator)
 raises(IExQueryProcessingError, IExQueryInvalid, IExQueryTypeInvalid);

 void evaluate_to_data_array(in string query, in QLType ql_type,
    in ParameterList collection_names,
    in ParameterList params,
    in unsigned long how_many,
    out DataArrayList aggregate_members,
    out DataArrayIterator aggregate_iterator)
 raises(IExQueryProcessingError, IExQueryInvalid, IExQueryTypeInvalid);

 void evaluate_collection(
    in IManagedCollections::IMIterable collection,
    in string query,
    in QLType ql_type,
    in ParameterList params,
    in unsigned long how_many,
    out MemberList collection_members,
    out IManagedCollections::IIterator collection_iterator)
 raises(IExQueryProcessingError, IExQueryInvalid, IExQueryTypeInvalid);
 };
```

Supported operations

QueryEvaluator::evaluate_collection
QueryEvaluator::evaluate_to_iterator
QueryEvaluator::evaluate_to_data_array

QueryEvaluator::evaluate_collection

Supports evaluate method on a collection (“queryableCollection”).

Original interface

IExtendedQuery::QueryEvaluator Interface

IDL syntax

```
void evaluate_collection(
      in IManagedCollections::IMIterable collection,
      in string query,
      in QLType ql_type,
      in ParameterList params,
      in unsigned long how_many,
      out MemberList collection_members,
      out IManagedCollections::IIterator collection_iterator)
   raises(IExtendedQuery::IExQueryProcessingError,
          IExtendedQuery::IExQueryInvalid,
          IExtendedQuery::IExQueryTypeInvalid);
```

Exceptions

IExtendedQuery::IExQueryProcessingError - Any error is encountered during query processing. An exception code, substitution variables and a string variable providing text explanation of the error are contained in the exception.

IExtendedQuery::IExQueryInvalid - Query string syntax or semantics is incorrect or the input parameter list is incorrect. An exception code, substitution variables and a string variable providing text explanation of the error are contained in the exception.

IExtendedQuery::IExQueryTypeInvalid - Query language is not supported by the QueryEvaluator. An exception code, substitution variables and a string variable providing text explanation of the error are contained in the exception.

Remarks

The evaluate_collection method is designed for supporting evaluate method on a collection (“queryableCollection”). It is intended for use by Component Broker internal components only. The evaluate method on a collection is supported by delegating the method to evaluate_collection with the collection as the first parameter passed in. The evaluate_collection method produces an iterator over a set of objects.

The syntax and semantics of evaluate_collection method are the same as evaluate_to_iterator method with the following two exceptions:

- The evaluate_collection has an extra in parameter collection.
- The syntax of the query string passed to evaluate_collection method requires that it contains only the predicate portion of a full select statement (must not contain “select”, “from” clauses and the key word where). The predicate is expanded to a full

query by the implementation of the method to “select x from collection x where YOUR_PREDICATE” so the predicate either can not contain quantifiers or the quantifier must be x.

Since the target of the query and the object type in the result collection are implicitly implied, the “select” and “from” clauses are redundant.

QueryEvaluator::evaluate_to_iterator

Produces an iterator over a set of objects resulting from an object query.

Original interface

IExtendedQuery::QueryEvaluator Interface

IDL syntax

```
void evaluate_to_iterator(
         in string query,
         in QLType ql_type,
         in ParameterList collection_names,
         in ParameterList params,
         in unsigned long how_many,
         out MemberList members,
         out IManagedCollections::IIterator members_iterator)
  raises(IExtendedQuery::IExQueryProcessingError,
         IExtendedQuery::IExQueryInvalid,
         IExtendedQuery::IExQueryTypeInvalid);
```

Exceptions

IExtendedQuery::IExQueryProcessingError - Any error is encountered during query processing. An exception code, substitution variables and a string variable providing text explanation of the error are contained in the exception.

IExtendedQuery::IExQueryInvalid - Query string syntax or semantics is incorrect or the input parameter list is incorrect. An exception code, substitution variables and a string variable providing text explanation of the error are contained in the exception.

IExtendedQuery::IExQueryTypeInvalid - Query language is not supported by the QueryEvaluator. An exception code, substitution variables and a string variable providing text explanation of the error are contained in the exception.

Remarks

The evaluate_to_iterator operation is used to produce an iterator over a set of objects resulting from an object query. The type of the iterator is IManagedCollections::IIterator. The interface is designed to closely resemble the CosQuery::QueryEvaluator::evaluate operation. Generally the implementation of the evaluate method can be mapped to the evaluate_to_iterator method when the ql_type is OOSQL to preserve compliance with the CORBA standard. In this case, the resulting iterator is returned in the CORBA any return value from the evaluate method.

The evaluate_to_iterator returns an iterator and optional member list as out arguments, thus making it perhaps more convenient to use than the standard evaluate method.

If how_many is set greater than zero, then that number of members are returned in the member list (fewer if how_many exceeds the total number of members resulting from the query) and the remainder are returned in the iterator. If how_many is set to zero then members will be empty and all of the resulting members are returned in the iterator. If all of the members are returned in the member list then the iterator is a NULL.

The iterator returned is of the type IManagedCollection::IIterator. However, the real run time type of the iterator is a subtype of IQueryManagedCollections::IIterator, which itself is a subtype of IManagedCollections::IIterator. This means that from an interface point of view the returned iterator can be treated as an IManagedCollections::IIterator but the implementation behavior such as persistency, home of the iterator, etc., may be different from other implementations of IManagedCollections::IIterator in Component Broker such as the iterator implementation provided by the reference collection component. When a non-NULL iterator is returned it must be removed and released by the client when it is through using it.

QueryEvaluator::evaluate_to_data_array

Used to produce an iterator over a set of DataArrays resulting from an object query.

Original interface

IExtendedQuery::QueryEvaluator Interface

IDL syntax

```
void evaluate_to_data_array(
         in string query,
         in QLType ql_type,
         in ParameterList collection_names,
         in ParameterList params,
         in unsigned long how_many,
         out DataArrayList aggregate_members,
         out DataArrayIterator aggregate_iterator)
      raises(IExtendedQuery::IExQueryProcessingError,
             IExtendedQuery::IExQueryInvalid,
             IExtendedQuery::IExQueryTypeInvalid);
```

Exceptions

IExtendedQuery::IExQueryProcessingError - Any error is encountered during query processing. An exception code, substitution variables and a string variable providing text explanation of the error are contained in the exception.

IExtendedQuery::IExQueryInvalid - Query string syntax or semantics is incorrect or the input parameter list is incorrect. An exception code, substitution variables and a string variable providing text explanation of the error are contained in the exception.

IExtendedQuery::IExQueryTypeInvalid - Query language is not supported by the QueryEvaluator. An exception code, substitution variables and a string variable providing text explanation of the error are contained in the exception.

Remarks

The evaluate_to_data_array method is used to produce an iterator over a set of DataArrays resulting from an object query. A DataArray is essentially a row in a relational database table or a view projected on a set of columns. A DataArray can represent partial data of an object or combination of data from different objects. A DataArray is a sequence of anys. If an element of a DataArray is a null value then the any corresponding to that element has the value any.type()= tk_null and any.value()= undefined. The original data type of the column corresponding to the NULL value can be obtained from the method get_field_type on DataArrayIterator returned from an evaluate_to_data_array method call.

The evaluate_to_data_array returns an iterator and optional member list as out-arguments, thus making it perhaps more convenient to use than the standard evaluate method.

If how_many is set greater than zero, then that number of DataArray are returned in the aggregate_members (fewer if how_many exceeds the total number of DataArrays resulting from the query) and the remainder are returned in the iterator. If how_many is set to zero then aggregate_members will be empty and all resulting DataArrays are returned in the iterator. If all of the DataArrays are returned in the aggregate_members then the iterator is a NULL.

When a non-NULL iterator is returned it must be removed and released by the client when it is through using it.

Chapter 28. IExtendedSecurity in the Security Service

The other modules in the Security Service are:
- IExtendedSecurityClient
- Security
- SecurityLevel1
- SecurityLevel2

IExtendedSecurity Module

The IExtendedSecurity module contains IBM extensions to the standard Object Management Group (OMG) defined interfaces and also new interfaces introduced by IBM. In particular, the CORBA::Principal interface is extended to contain two new attributes: security_name and host_name, and a new interface, LoginHelper, is introduced.

Note: OS/390 Component Broker does not support the IExtendedSecurity module.

File name

IExtendedSecurity.idl

Constants

```
const unsigned short IBM_BOSS_FAMILY_DEFINER = 8;

 // Component Broker Extensions to Security Attributes

 // Credential attributes: family definer = 8, family = 2
 const Security::SecurityAttributeType CredAttrSecName         = 1;
 const Security::SecurityAttributeType CredAttrHostName        = 2;
 const Security::SecurityAttributeType CredAttrInitSecObjGroup = 3;

 // Authorization attributes:

 //defining authority = 8, family = 0

 const Security::SecurityAttributeType PrivilegeUser         = 1;
 const Security::SecurityAttributeType PrivilegeGroup        = 2;
 const Security::SecurityAttributeType PrivilegeOther        = 3;
 const Security::SecurityAttributeType PrivilegeForeignUser  = 4;
 const Security::SecurityAttributeType PrivilegeForeignOther = 5;
 const Security::SecurityAttributeType PrivilegeAnyOther     = 6;
 const Security::SecurityAttributeType PrivilegeOwnerUser    = 7;
 const Security::SecurityAttributeType PrivilegeOwnerGroup   = 8;
```

Interfaces

IExtendedSecurity::“Current Interface” on page 640

IExtendedSecurity::Credentials Interface

IExtendedSecurity::LoginHelper Interface
IExtendedSecurity::Principal Interface

Credentials Interface

Derives from SecurityLevel2::Credentials Interface. Provides no additional functionality for this release.

File name

IExtendedSecurity.idl

Local-only

True

Ancestor interfaces

SecurityLevel2::Credentials Interface

IDL syntax

```
interface Credentials : SecurityLevel2::Credentials {
};
```

Current Interface

Derives from SecurityLevel2::Current Interface.

File name

IExtendedSecurity.idl

Local-only

True

Ancestor interfaces

SecurityLevel2::Current Interface

IDL syntax

```
interface Current : SecurityLevel2::Current {
};
```

LoginHelper Interface

Provides the request_login operation which is used by the Security Service to get login information from the Client (or Server) if the required Credentials are not available.

This is an IBM-defined interface, which extends the CORBA 2.0 specifications provided by the Object Management Group (OMG).

File name

IExtendedSecurity.idl

Intended usage

The login helper is an object that can be used to obtain user information with which to perform a login. The login helper is responsible for producing a credential for a user principal. Normally this is implemented to present a login pop-up. An instance of the LoginHelper object can be created at any time. The security service can provide different implementations of this object for different conditions. However the actual implementation class used by the security service is hard-coded into the service to prevent tampering.

Local-only

True

IDL syntax

```
interface LoginHelper {

   SecurityLevel2::Credentials request_login(
     in  Istring               security_name,
     in  Istring               realm_name,
     in  Istring               password,
     out  SecurityLevel2::Credentials               creds,
     out  Security::Opaque               auth_specific_data,
   )
   raises (SecurityLevel2::LoginFailed);
 };
```

Supported operations

LoginHelper::request_login

LoginHelper::request_login

The Security service or an application can use request_login to establish a client principal's identity and log the principal into DCE. This is done when the principal has not logged-in or the login has expired. At this time this interface is Workstation specific with a DCE based security mechanism.

Original interface

IExtendedSecurity::LoginHelper Interface

IDL syntax

```
// authenticating a user within a DCE realm
SecurityLevel2::Credentials request_login(
  in  Istring                security_name,
  in  Istring                realm_name,
  in  Istring                password,
  out  SecurityLevel2::Credentials               creds,
  out  Security::Opaque               auth_specific_data
)
   raises (SecurityLevel2::LoginFailed);
```

Parameters

security_name
The DCE login-name of the principal to be logged in.Note that the security_name should not contain the cell name which should be specified with the realm_name.

realm_name
The DCE cell into which the principal is to be logged.

password
The principal's DCE password. The password is held for as short a period as possible, and it's not transmitted on the network.

creds Credentials created as a result of a successful login. No credentials are returned if the login attempt fails.

auth_specific_data
Any other security information. Nothing will be returned in this release.

Return value

The credentials created if the login is successful. This is the same object as returned in the *creds* parameter.

Exceptions

SecurityLevel2::LoginFailed

Remarks

The required information (security_name, realm_name and password) is typically obtained by means of a ″login pop-up″ in a client and a keytab file in a server. There are other options and which option is used depends on the client's (or server's as the case may be) configuration information in the CDS (Common Data Store). Normally, this is done under the covers by the Security Service, but Client applications can also invoke this operation to perform explicit authentication.

Example

```
IExtendedSecurity::LoginHelper_ptr pLH =
   IExtendedSecurity::LoginHelper::_create();

char* security_name = "MyDCEUser";
char* realm_name = "MyDCECellName";
char* password = "MyDCEPassword";

SecurityLevel2::Credentials_ptr      pReturnCred;
SecurityLevel2::Credentials_ptr      pCred;
Security::Opaque*                    auth_specific_data;

pReturnCred = pLH->request_login( security_name,
                                  realm_name,
                                  password,
                                  pCred,
                                  auth_specific_data );
```

Principal Interface

This interface is not part of the programming model and should not be directly invoked or overridden.

Chapter 29. IExtendedSecurityClient in the Security Service

The other modules in the Security Service are:

- IExtendedSecurity
- Security
- SecurityLevel1
- SecurityLevel2

IExtendedSecurityClient Module

Contains the SecurableObject interface, intended as a mixin interface.

Note: OS/390 Component Broker does not support the IExtended SecurityClient module.

File name

IExtendedSecurityClient.idl

Interfaces

IExtendedSecurityClient::SecurableObject Interface

SecurableObject Interface

Provides a mixin for managed objects that need to be secure, that is, objects that demand controlled access. The interface is not implemented for this release of the Component Broker.

File name

IExtendedSecurityClient.idl

Intended usage

The SecurableObject interface is the base-class for all secure objects. It is used as a marker to prevent clients from making external requests on un-secure objects; that is, objects that are not derived from SecurableObject.

Implementations of the SecurableObject interface are required to provide a mechanism by which method requests invoked on the secure object are intercepted and tested for whether access to the object should be allowed for the requesting principal.

IDL syntax

```
interface SecurableObject{
};
```

Chapter 30. IExtendedStream in the Externalization Service

The other module in the Externalization Service is:
- CosStream

IExtendedStream Module

This module provides extensions to the CosStream::StreamIO that support the streaming of data objects of non-primitive types.

Supported on AIX and Windows NT only

File name

IExtendedStream.idl

Interfaces

IExtendedStream::StreamIO Interface

Intended usage

This module provides interfaces that allow an object to place its essential state, which may contain complex data types, into a sequential memory buffer.

StreamIO Interface

Defines methods which allow the streaming of complex constructed CORBA types.

File name

IExtendedStream.idl

Intended usage

This interface is intended to allow a user to read/write object state information from/to a StreamIO object, providing extended capability to stream objects of complex constructed CORBA types.

Local-only

True - This interface itself is not local only, but all implementations of this interface in Component Broker are local-only.

Ancestor interfaces

CosStream::StreamIO

Supported operations

StreamIO::read_any
StreamIO::write_any

StreamIO::read_any

This method reads and returns an any value from the StreamIO.

Original interface

IExtendedStream::StreamIO

IDL syntax

```
any read_any () raises (CosStream::StreamDataFormatError);
```

Return value

An any value read from the StreamIO

Exceptions

CosStream::StreamDataFormatError

StreamIO::write_any

This method writes an any value to the StreamIO.

Original interface

IExtendedStream::StreamIO

IDL syntax

```
void write_any(in any anAny);
```

Parameters

An any value to be written to the StreamIO

Chapter 31. ILifeCycleLocalObjectImpl in the LifeCycle Service

The other modules in the LifeCycle Service are:
- CosLifeCycle
- IExtendedLifeCycle
- ILifeCycleManagedClient

ILifeCycleLocalObjectImpl Module

Defines the local only object implementation of the Lifecycle service.

Intended usage

This module defines the implementation interfaces used with the local only object versions of LifeCycle objects. A local only object is sometimes also referred to as a non-managed object. The non-managed objects in this service will all follow the local only pattern as defined for Component Broker Connector (CBConnector).

Local only objects are non-remotable and non-managed. Instances of these objects can exist on servers or on clients. These interfaces do not introduce any new methods except those needed for creating and initializing instances of these local only objects.

Exceptions

ILifeCycleLocalObjectImpl::AlreadyInitialized

Interfaces

FactoryFinder Interface
ILocalOnly Interface
OrderedLocation Interface
ScopeManipulator Interface
SingleLocation Interface

FactoryFinder Interface

This interface introduces new operations which create and initialize local only instances of factory finder objects. These instances are non-remotable and non-managed.

Local-only

True

Supported methods

FactoryFinder::_create
FactoryFinder::_create(Location)
FactoryFinder::_create(Scope)
FactoryFinder::_create(ScopeString)

FactoryFinder::_create(SequenceOfLocations)
FactoryFinder::_create(OrderedScopes)
FactoryFinder::_create(OrderedScopeStrings)

FactoryFinder::_create

Creates a local-only factory finder and initializes it using the default SingleLocation object.

Original interface

ILifeCycleLocalObjectImpl::FactoryFinder

C++ syntax

```
_create();
```

Return value

ILifeCycleLocalObjectImpl::FactoryFinder

Remarks

The default SingleLocation object is in turn initialized with the default scope. The default scope is defined as follows:

Cell = *LOCAL
Workgroup = *LOCAL
Host = *LOCAL
Server = *ANY
Container = *ANY
Home = *ANY

See “Defining scope of location” in the *Component Broker Advanced Programming Guide* for more information. The factory finder returned by this call can be used for finding factories capable of creating objects in a very broad area. If a narrower scope is desired, use another variants of the _create function which allows a particular scope to be specified.

Example

```
// C ++ example
// Note: for code clarity, some exception handling, error checking and
//  cleanup have been ommitted from this example.

   CosLifeCycle::Factory_var    retfac;

// create a default local-only factory finder.

   ILifeCycleLocalObjectImpl::FactoryFinder_var factory_finder;
   factory_finder = ILifeCycleLocalObjectImpl::FactoryFinder::_create();
```

FactoryFinder::_create(Location)

Creates a local-only factoryfinder and initializes it with the given location.

Original interface

ILifeCycleLocalObjectImpl::FactoryFinder

C++ syntax

```
_create(IExtendedLifeCycle::Location);
```

Parameters

Location
An IExtendedLifeCycle::Location object

Return value

ILifeCycleLocalObjectImpl::FactoryFinder

Remarks

The Location parameter defines the scope of the resulting factory finder. See “Defining scope of location” in the *Component Broker Advanced Programming Guide* for more information on scopes. The location parameter may be a SingleLocation, OrderedLocation or a user implemented location.

Note that in most cases, if a SingleLocation or OrderedLocation object is desired for initializing the factory finder, other _create variations can be used to create the desired factory finder directly. This would eliminate the need to write code for creating a the Location object.

However, if a user-implemeted specialization of the Location interface is available, this _create variant provides the means to use it to initialize a factory finder. The factory finder returned by this call can be used for finding factories capable of creating objects in the area defined by the location parameter.

Example

```
// C ++ example
// Note: For code clarity, some exception handling, error checking and
//  cleanup have been ommitted from this example.

   CosLifeCycle::Factory_var    retfac;

// create a default local-only singlelocation.
   ILifeCycleLocalObjectImpl::SingleLocation_var single_location;
   single_location = ILifeCycleLocalObjectImpl::SingleLocation::_create();
```

```
// narrow the single_location to a location.
   IExtendedLifeCycle::Location_var location;
   location = IExtendedLifeCycle::Location::_narrow(single_location);

// create a local-only factory finder specifying the location.
// Note that since this example creates the default single location object
// and uses it to initialize the factory finder, the result is the same as
// it would have simply created the default factory finder.

   ILifeCycleLocalObjectImpl::FactoryFinder_var factory_finder;
   factory_finder =
      ILifeCycleLocalObjectImpl::FactoryFinder::_create(location);
```

FactoryFinder::_create(OrderedScopes)

Creates a local-only OrderedLocation object which contains the given scopes. Creates a local-only FactoryFinder and initializes it with this OrderedLocation object.

Original interface

ILifeCycleLocalObjectImpl::FactoryFinder

C++ syntax

```
_create(in IExtendedLifeCycle::OrderedScopes scopes)
   raises (ExtendedLifeCycle::InvalidScope);
```

Parameters

IExtendedLifeCycle::OrderedScopes

Return value

ILifeCycleLocalObjectImpl::FactoryFinder

Exceptions

IExtendedLifeCycle::InvalidScope
A scope in the OrderedScopes parameter did not obey the rules for valid scopes. See “Defining scope of location” in the *Component Broker Advanced Programming Guide* for more information.

Remarks

The factory finder returned by this call can be used for finding factories capable of creating objects in the area defined by the OrderedScopes parameter. It will first search using the first scope, then the second, etc. For information on supplying the scope parameter, please see “Defining scope of location” in the *Component Broker Advanced Programming Guide*.

Example

```
// C ++ example
// Note: for code clarity, some exception handling, error checking and
//  cleanup have been omitted from this example.

   CosLifeCycle::Factory_var    retfac;

// create and initialize a orderedscope sequence

   IExtendedLifeCycle::OrderedScopes_var    scopes;
   scopes = new IExtendedLifeCycle::OrderedScopes(2);
   scopes->length(2);

   // first scope is for server1
   (*scopes)[0].cell = CORBA::string_dup("*LOCAL");
   (*scopes)[0].workgroup = CORBA::string_dup("*LOCAL");
   (*scopes)[0].host = CORBA::string_dup("*LOCAL");
   // specify server1 name
   (*scopes)[0].server = CORBA::string_dup("server1");
   (*scopes)[0].container = CORBA::string_dup("*ANY");
   (*scopes)[0].home = CORBA::string_dup("*ANY");

   // second scope is for server2
   (*scopes)[1].cell = CORBA::string_dup("*LOCAL");
   (*scopes)[1].workgroup = CORBA::string_dup("*LOCAL");
   (*scopes)[1].host = CORBA::string_dup("*LOCAL");
   // specify server2 name
   (*scopes)[1].server = CORBA::string_dup("server2");
   (*scopes)[1].container = CORBA::string_dup("*ANY");
   (*scopes)[1].home = CORBA::string_dup("*ANY");

// create a local-only factory finder specifying the scope.

   ILifeCycleLocalObjectImpl::FactoryFinder_var factory_finder;
   factory_finder =
      ILifeCycleLocalObjectImpl::FactoryFinder::_create(scopes);
```

FactoryFinder::_create(OrderedScopeStrings)

Creates a local-only OrderedLocation object which contains the scopes specified in the given OrderedScopeStrings. Creates a local-only FactoryFinder and initializes it with this OrderedLocation object.

Original interface

ILifeCycleLocalObjectImpl::FactoryFinder

C++ syntax

```
_create (in IExtendedLifeCycle::OrderedScopeStrings scope)
   raises (IExtendedLifeCycle::InvalidScope,
           IExtendedLifeCycle::UnrecognizedScopeElement,
           IExtendedLifeCycle::IllegalStringSyntax,
           IExtendedLifeCycle::UnMatchedQuote);
```

Parameters

IExtendedLifeCycle::OrderedScopeStrings

Return value

IExtendedLifeCycle::FactoryFinder

Exceptions

CORBA standard exceptions and the following user exceptions:

InvalidScope - an element of the OrderedScopeStrings parameter did not obey the rules for valid scopes. See “Defining scope of location” in the *Component Broker Advanced Programming Guide* for more information.

UnrecognizedScopeElement - an element of the OrderedScopeStrings parameter contains an element other than cell, workgroup, host, server, container, or home. See “Defining scope of location” in the *Component Broker Advanced Programming Guide* for more information.

ExtendedLifeCycle::IllegalStringSyntax - indicates that the ScopeString has a syntax error.

UnmatchedQuote - indicates that there is an unmatched quote in the ScopeString.

Remarks

The factory finder returned by this call can be used for finding factories capable of creating objects in the area defined by the OrderedScopeStrings parameter. It will first search using the first scope, then the second, etc. For information on supplying the scope parameter, please see “Defining scope of location” in the *Component Broker Advanced Programming Guide*.

Example

```
// C ++ example
// Note: for code clarity, some exception handling, error checking and
//  cleanup have been omitted from this example.

   CosLifeCycle::Factory_var    retfac;

// create and initialize a orderedscopestring sequence

   IExtendedLifeCycle::OrderedScopeStrings_var    scopestrings;
   scopestrings = new IExtendedLifeCycle::OrderedScopeStrings(2);
   scopestrings->length(2);
```

```
// first scope is for server1
(*scopesstrings)[0] = CORBA::string_dup("server1.server");
// second scope is for server2
(*scopesstrings)[1] = CORBA::string_dup("server2.server");

ILifeCycleLocalObjectImpl::FactoryFinder_var factory_finder;
factory_finder = ILifeCycleLocalObjectImpl::FactoryFinder
   ::_create(scopestrings);
```

FactoryFinder::_create(Scope)

Creates a local-only SingleLocation object which contains the given scope. Creates a local-only FactoryFinder and initializes it with this SingleLocation object.

Original interface

ILifeCycleLocalObjectImpl::FactoryFinder

C++ syntax

```
_create(in IExtendedLifeCycle::Scope scope)
   raises (IExtendedLifeCycle::InvalidScope);
```

Parameters

IExtendedLifeCycle::Scope

Return value

ILifeCycleLocalObjectImpl::FactoryFinder

Exceptions

CORBA standard exceptions and the following user exception:

IExtendedLifeCycle::InvalidScope -The Scope parameter did not obey the rules for valid scopes. See “Defining scope of location” in the *Component Broker Advanced Programming Guide* for more information.

Remarks

The factory finder returned by this call can be used for finding factories capable of creating objects in the area defined by the scope parameter. For information on supplying the scope parameter, please see “Defining scope of location” in the *Component Broker Advanced Programming Guide*.

Example

```
// C ++ example
// Note: for code clarity, some exception handling, error checking and
//  cleanup have been ommitted from this example.

   CosLifeCycle::Factory_var    retfac;

// create and initialize a scope structure to specify MyServer server

   IExtendedLifeCycle::Scope    scope;
   scope.cell = CORBA::string_dup("*LOCAL");
   scope.workgroup = CORBA::string_dup("*LOCAL");
   scope.host = CORBA::string_dup("*LOCAL");
   scope.server = CORBA::string_dup("MyServer");  // specify my server name
   scope.container = CORBA::string_dup("*ANY");
   scope.home = CORBA::string_dup("*ANY");

// create a local-only factory finder specifying the scope.

   ILifeCycleLocalObjectImpl::FactoryFinder_var factory_finder;
   factory_finder = ILifeCycleLocalObjectImpl::FactoryFinder::_create(scope);
```

FactoryFinder::_create(ScopeString)

Creates a local-only SingleLocation object which contains the scope specified in the given ScopeString. Creates a local-only FactoryFinder and initializes it with this SingleLocation object.

Original interface

ILifeCycleLocalObjectImpl::FactoryFinder

C++ syntax

```
_create (in IExtendedLifeCycle::ScopeString scope)
   raises (IExtendedLifeCycle::InvalidScope,
           IExtendedLifeCycle::UnrecognizedScopeElement,
           IExtendedLifeCycle::IllegalStringSyntax,
           IExtendedLifeCycle::UnMatchedQuote);
```

Parameters

IExtendedLifeCycle::ScopeString

Return value

IExtendedLifeCycle::FactoryFinder

Exceptions

CORBA standard exceptions and the following user exceptions:

InvalidScope - the Scope parameter did not obey the rules for valid scopes. See "Defining scope of location" in the *Component Broker Advanced Programming Guide* for more information.

UnrecognizedScopeElement - the ScopeString parameter contains an element other than cell, workgroup, host, server, container, or home. See "Defining scope of location" in the *Component Broker Advanced Programming Guide* for more information.

ExtendedLifeCycle::IllegalStringSyntax - indicates that the ScopeString has a syntax error.

UnmatchedQuote - indicates that there is an unmatched quote in the ScopeString.

Remarks

The factory finder returned by this call can be used for finding factories capable of creating objects in the area defined by the ScopeString parameter. For information on supplying the scope parameter, please see "Defining scope of location" in the *Component Broker Advanced Programming Guide*.

Example

```
// C ++ example
// Note: for code clarity, some exception handling, error checking and
//  cleanup have been ommitted from this example.

   CosLifeCycle::Factory_var    retfac;

// create a local-only factory finder specifying my server name in the scope
//  string

   ILifeCycleLocalObjectImpl::FactoryFinder_var factory_finder;
   factory_finder =
      ILifeCycleLocalObjectImpl::FactoryFinder::_create("MyServer.server");
```

FactoryFinder::_create(SequenceOfLocations)

Creates a local-only OrderedLocation object which contains the given locations. Creates a local-only factoryfinder and initializes it with the ordered location.

Original interface

ILifeCycleLocalObjectImpl::FactoryFinder

C++ syntax

```
_create(in IExtendedLifeCycle::SequenceOfLocations locations)
  raises(IExtendedLifeCycle::InvalidScope);
```

Parameters

Location
An IExtendedLifeCycle::SequenceOfLocations structure.

Return value

ILifeCycleLocalObjectImpl::FactoryFinder

Exceptions

CORBA standard exceptions and the following user exceptions:

InvalidScope
The Scope parameter did not obey the rules for valid scopes. See “Defining scope of location” in the *Component Broker Advanced Programming Guide* for more information.

Remarks

The SequenceOfLocations parameter defines the scope of the resulting factory finder. See “Defining scope of location” in the *Component Broker Advanced Programming Guide* for more information on scopes.

This _create variant may be useful if it is desirable to have several factory finders which use the same scopes (represented by Location objects) in different orders. For example, if there is a server1 scope and a server2 scope, a client could create one Location for each server. Then it could create one factory finder which will first look on server1 followed by server2, and another factory finder which does the opposite, just by manipulating the order of the Locations within the sequence.

Example

```
// C ++ example
// Note: for code clarity, some exception handling, error checking and
//  cleanup have been omitted from this example.

   CosLifeCycle::Factory_var    retfac;

// create a server1 local-only singlelocation.
   ILifeCycleLocalObjectImpl::SingleLocation_var single_location1;
   single_location1 = ILifeCycleLocalObjectImpl::SingleLocation
                      ::_create("server1.server");

// narrow the single_location to a location.
   IExtendedLifeCycle::Location_var location1;
   location1 = IExtendedLifeCycle::Location::_narrow(single_location1);

// create a server2 local-only singlelocation.
   ILifeCycleLocalObjectImpl::SingleLocation_var single_location2;
   single_location2 = ILifeCycleLocalObjectImpl::SingleLocation
                      ::_create("server2.server");
```

```
// narrow the single_location to a location.
   IExtendedLifeCycle::Location_var location2;
   location2 = IExtendedLifeCycle::Location::_narrow(single_location2);

// put the locations in a sequence such that the search order will be
// server1 followed by server2

   IExtendedLifeCycle::SequenceOfLocations    *locations;
   locations = new IExtendedLifeCycle::SequenceOfLocations(2);
   locations->length(2);
   (*locations)[0] = location1;
   (*locations)[1] = location2;

// create a local-only factory finder specifying the location.
// Note that since this example creates a sequence of SINGLE location
// objects and uses it to initialize the factory finder, the result is
// the same as if it would have simply created the factory finder using
// a sequence of scopes or scopestrings.

   ILifeCycleLocalObjectImpl::FactoryFinder_var factory_finder;
   factory_finder =
      ILifeCycleLocalObjectImpl::FactoryFinder::_create(locations);
```

ILocalOnly Interface

Provides a local only implementation for the CosObjectIdentity::IdentifiableObject Interface.

Local-only

True

Intended usage

This Interface is inherited as a base implementation for all of the local only objects implemented by the LifeCycle Service.

Ancestor interfaces

CosObjectIdentity::IdentifiableObject
IManagedLocal::ILocalOnly

IDL syntax

```
interface ILocalOnly :
  IManagedLocal::ILocalOnly,
  CosObjectIdentity::IdentifiableObject {
};
#pragma meta ILocalOnly localonly, abstract
```

Supported operations

ILocalOnly::is_identical
ILocalOnly::constant_random_id

ILocalOnly::is_identical

Determines whether this object is identical to the object passed in the other_object parameter.

Original interface

ILifeCycleLocalObjectImpl::ILocalOnly

IDL syntax

```
is_identical (IdentifiableObject other_object);
```

ILocalOnly::constant_random_id

Returns the ObjectIdentifier associated with this object.

Original interface

ILifeCycleLocalObjectImpl::ILocalOnly

OrderedLocation Interface

Provides a local only implementation for the ExtendedLifeCycle::OrderedLocation Interface.

This interface introduces new operations which create and initialize local only instances of OrderedLocation objects. These instances are non-remotable and non-managed.

Local-only

True

Ancestor interfaces

OrderedLocation Interface

Supported operations

OrderedLocation::_create(OrderedScopes)
OrderedLocation::_create(OrderedScopeStrings)
OrderedLocation::_create(SequenceOfLocations)

OrderedLocation::_create(OrderedScopes)

Creates a local only OrderedLocation object.

Original interface

ILifeCycleLocalObjectImpl::OrderedLocation

IDL syntax

```
_create(in IExtendedLifeCycle::OrderedScopes scope)
   raises(IExtendedLifeCycle::InvalidScope);
```

Parameters

scopes
IExtendedLifeCycle::OrderedScopes

Return value

ILifeCycleLocalObjectImpl::OrderedLocation

Exceptions

CORBA standard exception and the following user exceptions:
IExtendedLifeCycle::InvalidScope

The Scope parameter did not obey the rules for valid scopes. See “Defining scope of location” in the *Component Broker Advanced Programming Guide* for more information.

Remarks

This static class method creates a local only SingleLocation object for each scope specified in the given scope sequence. It then creates a local only OrderedLocation object and initializes it with the newly created SingleLocation objects.

See “Defining scope of location” in the *Component Broker Advanced Programming Guide* for more information on scope structures.

The OrderedLocation returned by this call can be used to create a factory finder which will operate with the given scopes; first attempting to find a factory with the first scope, followed by the second, etc. Note that an equivalent factory finder can be created by invoking ILifeCycleLocalObjectImpl::FactoryFinder::_create(OrderedScopes);

Example

```
// C ++ example
// Note: for code clarity, some standard client setup, exception handling,
//  error checking and cleanup have been omitted from this example.

 // create and initialize a orderedscope sequence
```

```
   IExtendedLifeCycle::OrderedScopes_var    scopes;
   scopes = new IExtendedLifeCycle::OrderedScopes(2);
   scopes->length(2);

   // first scope is for server1
   (*scopes)[0].cell = CORBA::string_dup("*LOCAL");
   (*scopes)[0].workgroup = CORBA::string_dup("*LOCAL");
   (*scopes)[0].host = CORBA::string_dup("*LOCAL");
   // specify server1 name
   (*scopes)[0].server = CORBA::string_dup("server1");
   (*scopes)[0].container = CORBA::string_dup("*ANY");
   (*scopes)[0].home = CORBA::string_dup("*ANY");

   // second scope is for server2
   (*scopes)[1].cell = CORBA::string_dup("*LOCAL");
   (*scopes)[1].workgroup = CORBA::string_dup("*LOCAL");
   (*scopes)[1].host = CORBA::string_dup("*LOCAL");
   // specify server2 name
   (*scopes)[1].server = CORBA::string_dup("server2");
   (*scopes)[1].container = CORBA::string_dup("*ANY");
   (*scopes)[1].home = CORBA::string_dup("*ANY");

// create a local-only ordered location specifying the scope.

   ILifeCycleLocalObjectImpl::OrderedLocation_var ordered_location;
   ordered_location =
      ILifeCycleLocalObjectImpl::OrderedLocation::_create(scopes);
```

OrderedLocation::_create(OrderedScopeStrings)

Creates a local only OrderedLocation object.

Original interface

ILifeCycleLocalObjectImpl::OrderedLocation

IDL syntax

```
_create (in IExtendedLifeCycle::OrderedScopeStrings scopes)
   raises (IExtendedLifeCycle::InvalidScope,
           IExtendedLifeCycle::UnrecognizedScopeElement,
           IExtendedLifeCycle::IllegalStringSyntax,
           IExtendedLifeCycle::UnMatchedQuote);
```

Parameters

scopes

IExtendedLifeCycle::OrderedScopeStrings

Return value

ILifeCycleLocalObjectImpl::OrderedLocation

Exceptions

CORBA standard exceptions and the following user exceptions:

IExtendedLifeCycle::InvalidScope
The Scope parameter did not obey the rules for valid scopes. See “Defining scope of location” in the *Component Broker Advanced Programming Guide* for more information.

IExtendedLifeCycle::UnrecognizedScopeElement
The ScopeString parameter contains an element other than cell, workgroup, host, server, container, or home. See “Defining scope of location” in the *Component Broker Advanced Programming Guide* for more information.

IExtendedLifeCycle::IllegalStringSyntax
Indicates that the ScopeString has a syntax error.

IExtendedLifeCycle::UnmatchedQuote
Indicates that there is an unmatched quote in the ScopeString.

Remarks

This static class method creates a local only SingleLocation object for each scope string specified in the given scope string sequence. It then creates a local only OrderedLocation object and initializes it with the newly created SingleLocation objects.

See “Defining scope of location” in the *Component Broker Advanced Programming Guide* for more information.

The OrderedLocation returned by this call can be used to create a factory finder which will operate with the given scope strings; first attempting to find a factory with the first scope, followed by the second, etc. Note that an equivalent factory finder can be created by invoking
ILifeCycleLocalObjectImpl::FactoryFinder::_create(OrderedScopeStrings);

Example

```
// C ++ example
// Note: for code clarity, some exception handling, error checking and
//  cleanup have been omitted from this example.

#include <ILifeCycleLocalObjectImpl.hh>
#include <io.h>
#include <stdio.h>
#include <fstream.h>
#include <fcntl.h>
#include <sys.stat.h>
#include <orb.h>
```

```
#include <assert.h>
#include <stdlib.h>

int main(int argc, char * argv[])
{

   CosLifeCycle::Factory_var    retfac;

// create and initialize a orderedscopestring sequence

   IExtendedLifeCycle::OrderedScopeStrings_var    scopestrings;
   scopestrings = new IExtendedLifeCycle::OrderedScopeStrings(2);
   scopestrings->length(2);

   // first scope is for server1
   (*scopesstrings)[0] = CORBA::string_dup("server1.server");
   // second scope is for server2
   (*scopesstrings)[1] = CORBA::string_dup("server2.server");

// create a local-only ordered location specifying the scope.

   ILifeCycleLocalObjectImpl::OrderedLocation_var ordered_location;
   ordered_location = ILifeCycleLocalObjectImpl::OrderedLocation
      ::_create(scopestrings);
   if (!ordered_location)
   {
      cout << "Failed to create ordered location \n";
      exit (-1);
   }

   cout << "ordered location created\n";
   cout.flush();

// narrow the single_location to a location.
   IExtendedLifeCycle::Location_var location;
   location = IExtendedLifeCycle::Location::_narrow(ordered_location);

// create a local-only factory finder specifying the location.
// Note that the same results could have been acheived by doing
// ILifeCycleLocalObjectImpl::FactoryFinder::_create(scopestrings);

   ILifeCycleLocalObjectImpl::FactoryFinder_var factory_finder;
   factory_finder =
      ILifeCycleLocalObjectImpl::FactoryFinder::_create(location);
   if (!factory_finder)
   {
      cout << "Failed to create factory finder \n";
      exit (-1);
   }

   cout << "Factory Finder created\n";
   cout.flush();


// find 'MyTest' factory
```

```
    cout << "Trying  find_factories_from_string on interface MyTest \n";
    retfac = factory_finder->find_factory_from_string("MyTest.object
       interface");
    cout << "Got factory! " << endl;

    cout << "GoodBye! "<< endl;

}
```

OrderedLocation::_create(SequenceOfLocations)

Creates a local only OrderedLocation object.

Original interface

ILifeCycleLocalObjectImpl::OrderedLocation

IDL syntax

```
_create (in IExtendedLifeCycle::SequenceOfLocations locations)
  raises(IExtendedLifeCycle::InvalidScope);
```

Parameters

locations
IExtendedLifeCycle::SequenceOfLocations

Return value

ILifeCycleLocalObjectImpl::OrderedLocation

Exceptions

CORBA standard exceptions and the following user exceptions:

InvalidScope
The Scope parameter did not obey the rules for valid scopes. See “Defining scope of location” in the *Component Broker Advanced Programming Guide* for more information.

Remarks

This static class method creates a local only OrderedLocation object and initializes it using the given locations sequence. The locations in the sequence may consist of SingleLocations, other OrderedLocations, or user supplied implementations of the IExtendedLifeCycle::Location interface, in any combination.

This method may be useful for creating new ordered locations by using existing location objects in different orders. The OrderedLocation returned by this call can be used to create a factory finder which will operate with the given location sequence; first

attempting to find a factory with the first location, followed by the second, etc. Note that an equivalent factory finder can be created by invoking ILifeCycleLocalObjectImpl::FactoryFinder::_create(SequenceOfLocations);

Example

```
// C ++ example
// Note: for code clarity, some exception handling, error checking and
//  cleanup have been omitted from this example.

#include <ILifeCycleLocalObjectImpl.hh>
#include <io.h>
#include <stdio.h>
#include <fstream.h>
#include <fcntl.h>
#include <sys/stat.h>
#include <orb.h>
#include <assert.h>
#include <stdlib.h>

int main(int argc, char * argv[])
{

   CosLifeCycle::Factory_var    retfac;

// create a server1 local-only singlelocation.
   ILifeCycleLocalObjectImpl::SingleLocation_var single_location1;
   single_location1 = ILifeCycleLocalObjectImpl::SingleLocation
      ::_create("server1.server");

// narrow the single_location to a location.
   IExtendedLifeCycle::Location_var location1;
   location1 = IExtendedLifeCycle::Location::_narrow(single_location1);

// Use the supplied default "host-scope-widened" location as the second
// location in the sequence.
   CORBA::Object_var temp_var = root_context->resolve_with_string(
                  "host/resources/locations/host-scope-widened");

   IExtendedLifeCycle::Location_var location2;
   location2 = IExtendedLifeCycle::Location::_narrow(temp_var);

// put the locations in a sequence such that the search order will be
// server1 followed by host-scope-widened

   IExtendedLifeCycle::SequenceOfLocations    *locations;
   locations = new IExtendedLifeCycle::SequenceOfLocations(2);
   locations->length(2);
   (*locations)[0] = location1;
   (*locations)[1] = location2;

// create a local-only ordered location specifying the locations.

   ILifeCycleLocalObjectImpl::OrderedLocation_var ordered_location;
   ordered_location =
```

```
      ILifeCycleLocalObjectImpl::OrderedLocation::_create(locations);
   if (!ordered_location)
   {
      cout << "Failed to create ordered location \n";
      exit (-1);
   }

   cout << "ordered location created\n";
   cout.flush();

// narrow the single_location to a location.
   IExtendedLifeCycle::Location_var location;
   location = IExtendedLifeCycle::Location::_narrow(ordered_location);

// create a local-only factory finder specifying the location.
// Note that the same results could have been acheived by doing
// ILifeCycleLocalObjectImpl::FactoryFinder::_create(locations);

   ILifeCycleLocalObjectImpl::FactoryFinder_var factory_finder;
   factory_finder =
      ILifeCycleLocalObjectImpl::FactoryFinder::_create(location);
   if (!factory_finder)
   {
      cout << "Failed to create factory finder \n";
      exit (-1);
   }

   cout << "Factory Finder created\n";
   cout.flush();


// find 'MyTest' factory
   cout << "Trying  find_factories_from_string on interface MyTest \n";
   retfac = factory_finder->find_factory_from_string("MyTest.object
      interface");
   cout << "Got factory! " << endl;

   cout << "GoodBye! "<< endl;

}
```

ScopeManipulator Interface

Provides a local only implementation for the IExtendedLifeCycle::ScopeManipulator interface.

Local-only

True

Ancestor interfaces

IExtendedLifeCycle::ScopeManipulator
ILifeCycleLocalOnlyImpl::ILocalOnly

IDL syntax

```
interface ScopeManipulator :
    IExtendedLifeCycle::ScopeManipulator,
    ILocalOnly {
  // The following _create methods are provided as static
  // class methods which new up the object and invoke the
  // corresponding initialization method.
  //  _create();
    void initialize_with_defaults(
    )raises(
      AlreadyInitialized
    );
    #pragma meta initialize_with_defaults init

    #pragma meta initialize_with_ssm init
};
#pragma meta ScopeManipulator localonly
```

Supported operations

ScopeManipulator::_create

ScopeManipulator::_create

Creates a local only ScopeManipulator object.

Original interface

ILifeCycleLocalObjectImpl::ScopeManipulator

C++ syntax

```
_create();
```

Return value

ILifeCycleLocalObjectImpl::ScopeManipulator

Remarks

This static class method creates a local only ScopeManipulator object that is then initialized with the default StandardSyntax Model.

The default StandardSyntax Model is implemented by the NamingStringSyntax module. The resulting ScopeManipulator can be used to convert a Scope to a ScopeString or visa versa. See “Defining scope of location” in the *Component Broker Advanced Programming Guide* for more information.

Example

```
// C ++ example
// Note: for code clarity, some exception handling, error checking and
//  cleanup have been ommitted from this example.

// create a local only ScopeManipulator

   ILifeCycleLocalObjectImpl::ScopeManipulator_var      scopeman;
   scopeman = ILifeCycleLocalObjectImpl::ScopeManipulator::_create();

// create and initialize a scope structure to specify MyServer server

   IExtendedLifeCycle::Scope    scope;
   scope.cell = CORBA::string_dup("*LOCAL");
   scope.workgroup = CORBA::string_dup("*LOCAL");
   scope.host = CORBA::string_dup("*LOCAL");
   scope.server = CORBA::string_dup("MyServer");   // specify my server name
   scope.container = CORBA::string_dup("*ANY");
   scope.home = CORBA::string_dup("*ANY");

// use the scope manipulator to convert the scope to a string and print it

   cout << " The scope is " << scopeman->scope_to_string(scope) << endl;
```

SingleLocation Interface

Provides a local only implementation for the IExtendedLifeCycle::SingleLocation Interface.

This interface introduces new operations which create and initialize local only instances of SingleLocation objects. These instances are non-remotable and non-managed.

Local-only

True

Ancestor interfaces

SingleLocation Interface

Supported methods

SingleLocation::_create
SingleLocation::_create(Scope)
SingleLocation::_create(ScopeString)

SingleLocation::_create

Creates a local only SingleLocation object.

Original interface

ILifeCycleLocalObjectImpl::SingleLocation

C++ syntax

```
_create();
```

Return value

ILifeCycleLocalObjectImpl::SingleLocation

Remarks

This is a static class method that creates a local only SingleLocation object that is then initialized with the default scope. The default scope is defined as follows:

Cell = *LOCAL
Workgroup = *LOCAL
Host = *LOCAL
Server = *ANY
Container = *ANY
Home = *ANY

See “Defining scope of location” in the *Component Broker Advanced Programming Guide* for more information.

The SingleLocation returned by this call can be used to create a factory finder with the default scope. Note that is can also be achieved by invoking ILifeCycleLocalObjectImpl::FactoryFinder::_create();

SingleLocation::_create(Scope)

Creates a local only SingleLocation object.

Original interface

ILifeCycleLocalObjectImpl::SingleLocation

C++ syntax

```
_create(in IExtendedLifeCycle::Scope scope)
   raises(ExtendedLifeCycle::InvalidScope);
```

Parameters

scope IExtendedLifeCycle::Scope

Return value

ILifeCycleLocalObjectImpl::SingleLocation

Exceptions

CORBA standard exception and the following user exceptions:
IExtendedLifeCycle::InvalidScope

The Scope parameter did not obey the rules for valid scopes. See “Defining scope of location” in the *Component Broker Advanced Programming Guide* for more information.

Remarks

This static class method creates a local only SingleLocation object that is then initialized with the given scope.

See “Defining scope of location” in the *Component Broker Advanced Programming Guide* for more information.

The SingleLocation returned by this call can be used to create a factory finder which will operate with the given scope. Note that is can also be achieved by invoking ILifeCycleLocalObjectImpl::FactoryFinder::_create(scope);

Example

```
// C ++ example
// Note: for code clarity, some standard client setup, exception handling,
//  error checking and cleanup have been omitted from this example.

// create and initialize a scope structure to specify MyServer server

   IExtendedLifeCycle::Scope    scope;
   scope.cell = CORBA::string_dup("*LOCAL");
   scope.workgroup = CORBA::string_dup("*LOCAL");
   scope.host = CORBA::string_dup("*LOCAL");
   scope.server = CORBA::string_dup("MyServer");   // specify my server name
   scope.container = CORBA::string_dup("*ANY");
   scope.home = CORBA::string_dup("*ANY");

// create a local-only single location specifying the scope.

   ILifeCycleLocalObjectImpl::SingleLocation_var single_location;
   single_location = ILifeCycleLocalObjectImpl::SingleLocation::_create(scope);
```

SingleLocation::_create(ScopeString)

Creates a local only SingleLocation object.

Original interface

ILifeCycleLocalObjectImpl::SingleLocation

C++ syntax

```
_create (in IExtendedLifeCycle::ScopeString scope)
   raises (IExtendedLifeCycle::InvalidScope,
           IExtendedLifeCycle::UnrecognizedScopeElement,
           IExtendedLifeCycle::IllegalStringSyntax,
           IExtendedLifeCycle::UnMatchedQuote);
```

Parameters

scope IExtendedLifeCycle::ScopeString

Return value

ILifeCycleLocalObjectImpl::SingleLocation

Exceptions

CORBA standard exceptions and the following user exceptions:

IExtendedLifeCycle::InvalidScope - The Scope parameter did not obey the rules for valid scopes. See “Defining scope of location” in the *Component Broker Advanced Programming Guide* for more information.

IExtendedLifeCycle::UnrecognizedScopeElement - The ScopeString parameter contains an element other than cell, workgroup, host, server, container, or home. See “Defining scope of location” in the *Component Broker Advanced Programming Guide* for more information.

IExtendedLifeCycle::IllegalStringSyntax - indicates that the ScopeString has a syntax error.

IExtendedLifeCycle::UnmatchedQuote - indicates that there is an unmatched quote in the ScopeString.

Remarks

This is a static class method that creates a local only SingleLocation object that is then initialized with the given scope string.

See “Defining scope of location” in the *Component Broker Advanced Programming Guide* for more information.

The SingleLocation returned by this call can be used to create a factory finder which will operate with the given scope. Note that is can also be achieved by invoking ILifeCycleLocalObjectImpl::FactoryFinder::_create(ScopeString);

Example

```
// C ++ example
// Note: For code clarity, some exception handling, error checking and
// cleanup have been omitted from this example.

// create a local-only single location specifying a scope string

   ILifeCycleLocalObjectImpl::SingleLocation_var single_location;
   single_location =
      ILifeCycleLocalObjectImpl::SingleLocation::_create("MyServer.server");
   cout << "single location created\n";
```

Chapter 32. ILifeCycleManagedClient in the LifeCycle Service

The other modules in the LifeCycle Service are:
- CosLifeCycle
- IExtendedLifeCycle
- ILifeCycleLocalObjectImpl

ILifeCycleManagedClient Module

This module introduces the inherited interfaces defined by the programming model that are needed to make objects manageable (this is equivalent to the full interface definition for business logic in the programming model, e.g. Policy). Clients should use this module when dealing with the managed object version of Factory Finders or Single Locations.

File name

ILifeCycleManagedClient.idl

Interfaces

ILifeCycleManagedClient::FactoryFinder Interface
ILifeCycleManagedClient::OrderedLocation Interface
ILifeCycleManagedClient::OrderedLocationHome Interface
ILifeCycleManagedClient::SingleLocation Interface
ILifeCycleManagedClient::FactoryFinderHome Interface
ILifeCycleManagedClient::SingleLocationHome Interface

FactoryFinder Interface

Defines the interfaces needed by clients when using a managed object implementation of the IExtendedLifeCycle::FactoryFinder interface.

Local-only

False

Ancestor interfaces

IExtendedLifeCycle::FactoryFinder

IDL syntax

```
interface FactoryFinder :
   IExtendedLifeCycle::FactoryFinder,
   IManagedClient::IManageable {
};
```

FactoryFinderHome Interface

Defines the interfaces needed by clients when using a managed object implementation of the IExtendedLifeCycle::FactoryFinderHome interface.

Local-only

False

Ancestor interfaces

IExtendedLifeCycle::FactoryFinderHome

IDL syntax

```
interface FactoryFinderHome :
   IExtendedLifeCycle::FactoryFinderHome {
};
```

OrderedLocation Interface

Defines the interfaces needed by clients when using a managed object implementation of the IExtendedLifeCycle::OrderedLocation interface.

Local-only

False

Ancestor interfaces

IExtendedLifeCycle::OrderedLocation

IDL syntax

```
interface OrderedLocation :
   IExtendedLifeCycle::OrderedLocation,
   IManagedClient::IManageable {
};
```

OrderedLocationHome Interface

Defines the interfaces needed by clients when using a managed object implementation of the IExtendedLifeCycle::OrderedLocationHome interface.

Local-only

False

Ancestor interfaces

IExtendedLifeCycle::OrderedLocationHome

IDL syntax

```
interface OrderedLocationHome :
   IExtendedLifeCycle::OrderedLocationHome {
};
```

Example

See example in IExtendedLifeCycle::“OrderedLocationHome Interface” on page 587.

SingleLocation Interface

Defines the interfaces needed by clients when using a managed object implementation of the IExtendedLifeCycle::SingleLocation interface.

Local-only

False

Ancestor interfaces

IExtendedLifeCycle::SingleLocation

IDL syntax

```
interface SingleLocation :
   IExtendedLifeCycle::SingleLocation,
   IManagedClient::IManageable {
};
```

SingleLocationHome Interface

Defines the interfaces needed by clients when using a managed object implementation of the IExtendedLifeCycle::SingleLocationHome interface.

Local-only

False

Ancestor interfaces

IExtendedLifeCycle::SingleLocationHome

IDL syntax

```
interface SingleLocationHome :
   IExtendedLifeCycle::SingleLocationHome {
};
```

Example

See example in IExtendedLifeCycle::FactoryFinderHome.

Chapter 33. IManagedAdvancedClient in the Managed Object Framework

The other modules in the Managed Object Framework are:

- CBSeriesGlobal
- IBOIMExtLocal
- IBOIMExtLocalToServer
- IBOIMInstanceManagerFriendQOS
- IBOIMManagedObjectFriendQOS
- IBOIMServiceFriendQOS
- ICollectionsBase
- IManagedAdvancedServer
- IManagedClient
- IManagedCollections
- IManagedLocal
- IManagedServer
- IRDBIMExtLocalToServer

IManagedAdvancedClient Module

This module contains the interfaces for the advanced concepts in the Managed Object Framework in conjunction with collections. It extends the basic interface to included views, support for query, and iteration. The interfaces described here are not supported by all application adaptors.

File name

IManagedAdvancedClient.idl

Interfaces

IManagedAdvancedClient::IIterableHome Interface
IManagedAdvancedClient::IQueryableIterableHome Interface
IManagedAdvancedClient::IView Interface

IIterableHome Interface

This interface is used to provide Homes that are iterable. Many times an application needs access to multiple objects that are contained within a single home. This interface provides a mechanism to access all objects contained by a home.

File name

IManagedAdvancedClient.idl

Intended usage

Derived from:
- IMIterable Interface
- IHome Interface

IQueryableIterableHome Interface

This interface is used to provide Homes that are queryable and iterable. Using the IIterableHome interface is sufficient for some applications, especially for those cases where the same operation needs to be performed for all objects in the Home. However, if the goal is to select a subset of all business objects in the Home then using IQueryableIterableHome interface is a better approach. This interface provides this mechanism of selecting the subset of objects to work on using Query and then iterating over that subset to perform the desired operations. It uses the properties of the types that it derives from to provide this support. No operations specific to the interface are defined.

File name

IManagedAdvancedClient.idl

Intended usage

Derived from:
- IIterableHome Interface
- IView Interface

IView Interface

This interface is used to provide a View over a collection. A view provides a simpler alternative for generating a collection of objects on a home. The result is not directly iterable. If iteration is required then an iterator can be created over the view after it has been created. A string that forms a query is required to define an instance of a view.

File name

IManagedAdvancedClient.idl

Intended usage

Derived from:
- IMIterable Interface
- IMQueryable Interface
- IManageable Interface

Supported operations

IView::createView

IView::createView

This operation creates an instance of a view over a collection.

Original interface

IManagedAdvancedClient::IView

IDL syntax

```
IView createView(in string queryPredicate);
```

Parameters

queryPredicate
This is a string that forms a query predicate. This value is used to create an instance of a View.

Return value

IView Returns an instance of a View.

Chapter 34. IManagedAdvancedServer in the Managed Object Framework

The other modules in the Managed Object Framework are:

- CBSeriesGlobal
- IBOIMExtLocal
- IBOIMExtLocalToServer
- IBOIMInstanceManagerFriendQOS
- IBOIMManagedObjectFriendQOS
- IBOIMServiceFriendQOS
- ICollectionsBase
- IManagedAdvancedClient
- IManagedClient
- IManagedCollections
- IManagedLocal
- IManagedServer
- IRDBIMExtLocalToServer

IManagedAdvancedServer Module

This module provides advanced server side interfaces provided by the Managed Object Framework. It provides interfaces for business object developers that want to create specialized homes. Specialized homes are special managed objects that provide specific behavior related to business objects. See the specialized home section in the *Component Broker Programming Guide* for more information on specialized homes.

File name

IManagedAdvancedServer.idl

Interfaces

IManagedAdvancedServer::ISpecializedHome Interface
IManagedAdvancedServer::ISpecializedHomeDataObject Interface
IManagedAdvancedServer::ISpecializedHomeManagedObject Interface
IManagedAdvancedServer::ISpecializedIterableHome Interface
IManagedAdvancedServer::ISpecializedQueryableIterableHome Interface
IManagedAdvancedServer::ISpecializedQueryableIterableHomeManagedObject Interface
IManagedAdvancedServer::IUUIDCopyHelperBase Interface
IManagedAdvancedServer::IUUIDPrimaryKey Interface

ISpecializedHome Interface

This interface is used by business object developers when creating specialized homes. It is the business object interface that business object developers would inherit from for implementing basic specialized homes. It provides no specific operations, it does provide the necessary support within the framework that is required for homes. This enables business application developers to transparently add logic to their homes that is specific to their business objects without having to develop all of the support required of a home in the Managed Object Framework.

File name

IManagedAdvancedServer.idl

ISpecializedHomeDataObject Interface

This interface is used by business object developers when creating specialized homes. It is the data object that is used within the implementation of a specialized home. This interface should not inherited from with the intent of providing a specific data object for a specialized home. User defined data objects for specialized homes is not support by the Managed Object Framework. The framework provides this data object as a mechanism for accessing data that was assigned to a home either in the object builder interface or by editing of the DDL.

File name

IManagedAdvancedServer.idl

Supported operations

"ISpecializedHomeDataObject ::getConfigInfo"

ISpecializedHomeDataObject ::getConfigInfo

This operation is the mechanism for accessing data that was assigned to a home either in the object builder interface or by editing of the DDL. This data is read-only and cannot be changed.

IDL syntax

```
char* getConfigInfo();
```

Example

```
// Go to my DO to get the userData attribute value out of the home
string_var phUserData = myDO->getConfigInfo();
if (strcmp(phUserData,"KeyAlgorithm1") == 0 )
 {
```

```
// use the first key algorithm
policyNo = getUniqueKey();
 }
  else
 {
// use the second key algorithm
policyNo = 100 + getUniqueKey();
 }
```

ISpecializedHomeManagedObject Interface

This interface is used by business object developers when creating specialized homes. It is the managed object interface that business object developers would inherit from for implementing basic specialized homes. It provides no specific operations, it does provide the necessary support within the framework that is required for homes. This is used in conjunction with ISpecializedHome.

File name

IManagedAdvancedServer.idl

ISpecializedIterableHome Interface

This interface is used by business object developers when creating specialized homes. It is the business object interface that business object developers would inherit from for implementing specialized homes that support iteration. It provides no specific operations, it does provide the necessary support within the framework that is required for homes. This enables business application developers to transparently add logic to their homes that is specific to their business objects without having to develop all of the support required of a home in the Managed Object Framework.

File name

IManagedAdvancedServer.idl

ISpecializedIterableHomeManagedObject Interface

This interface is used by business object developers when creating specialized homes. It is the managed object interface that business object developers would inherit from for implementing specialized homes that support iteration. It provides no specific operations, it does provide the necessary support within the framework that is required for homes. This is used in conjunction with ISpecializedIterableHome.

File name

IManagedAdvancedServer.idl

ISpecializedQueryableIterableHome Interface

This interface is used by business object developers when creating specialized homes. It is the business object interface that business object developers would inherit from for implementing specialized homes that support query and iteration. It provides no specific operations, it does provide the necessary support within the framework that is required for homes. This enables business application developers to transparently add logic to their homes that is specific to their business objects without having to develop all of the support required of a home in the Managed Object Framework.

File name

IManagedAdvancedServer.idl

ISpecializedQueryableIterableHomeManagedObject Interface

This interface is used by business object developers when creating specialized homes. It is the managed object interface that business object developers would inherit from for implementing specialized homes that support query and iteration. It provides no specific operations, it does provide the neccesary support within the framework that is required for homes. This is used in conjunction with ISpecializedQueryableIterableHome.

File name

IManagedAdvancedServer.idl

IUUIDCopyHelperBase Interface

This interface is intended to be a base class for the copy helpers of those managed objects that are based on the UUID (are transient).

Supported on AIX and Windows NT only

File name

IManagedAdvancedServer.idl

Intended usage

This interface is intended to be a base class for the copy helpers of those managed objects that are based on the UUID (are transient). Initializing the UUID portion of this interface is expected to occur during the construction of the object.

IDL syntax

```
interface IUUIDCopyHelperBase : IManagedLocal::INonManageable
{
   ByteString getUuid();
};
```

Supported operations

IUUIDCopyHelperBase::IUUIDCopyHelperBase::getUuid

IUUIDCopyHelperBase::getUuid

This operation returns the value of the unique identifier of the primary key.

Return value

The return value of this operation is a byte string that can be used as the key attribute in the data object. Use of this value with the fromString call on a primary key is not valid.

IUUIDPrimaryKey Interface

This interface is a default for primary key of managed objects that are transient.

Supported on AIX and Windows NT only

File name

IManagedAdvancedServer.idl

Intended usage

A new initializer is provided for the primary key. The primary key must be initialized by either the fromString method (supporting the inherited interface) or with the initializer method ″generateUuid″. Also a way to get at the one attribute that is the primary key is provided.

To guarantee uniqueness across Component Broker platforms, the use of this interface should be within a ″create″ method on a specialized home.

IDL syntax

```
interface IUUIDPrimaryKey : IManagedLocal::IPrimaryKey
{
   void generateUuid()
      raises(IManagedAdvancedServer::IUuidSetFailed);
   ByteString getUuid();
};
```

Supported operations

IUUIDPrimaryKey::IUUIDPrimaryKey::generateUuid
IUUIDPrimaryKey::IUUIDPrimaryKey::getUuid

IUUIDPrimaryKey::generateUuid

This operation is an initializer of the primary key. If a new primary key is needed to create an object then the ″generateUuid″ method is used to initialize the primary key with a new unique value.

Exceptions

IManagedAdvancedServer::IUuidSetFailed

This exception is raised if a unique identifier cannot be generated. The cause for this type of failure is either the primary key has already been initialized or the Component Broker runtime has had some problem. The event log will include any further information if the problem is in the Component Broker runtime.

IUUIDPrimaryKey::getUuid

This operation returns the value of the unique identifier of the primary key.

Return value

The return value of this operation is a byte string that can be used as the key attribute in the data object. Use of this value with the fromString call on a primary key is not valid.

Chapter 35. IManagedClient in the Managed Object Framework

The other modules in the Managed Object Framework are:

- CBSeriesGlobal
- IBOIMExtLocal
- IBOIMExtLocalToServer
- IBOIMInstanceManagerFriendQOS
- IBOIMManagedObjectFriendQOS
- IBOIMServiceFriendQOS
- ICollectionsBase
- IManagedAdvancedClient
- IManagedAdvancedServer
- IManagedCollections
- IManagedLocal
- IManagedServer
- IRDBIMExtLocalToServer

IManagedClient Module

This module contains the interfaces for managed objects. This includes the interfaces for the managed objects themselves as well as the interfaces for homes. Homes are containers for managed objects. Each managed object instance has a Primary Key, and is created in and found in exactly one home.

File name

IManagedLocal.idl

System Environment

typedef sequence<octet> ByteString

This typedef defines a basic type used throughout the Managed Object Framework. It is an sequence of octets (bytes) that is used for the various string operations. It is also used as the exchange mechanism for externalizing and internalizing the data that defines objects with the Managed Object Framework.

Interfaces

IManagedClient::IHome Interface
IManagedClient::IManageable Interface

Exceptions

IDuplicateKey
IInvalidCopy
IInvalidKey
INoObjectKey
INotUseable

IHome Interface

This interface provides the base support for Homes. A Home is the basic interface for creating and locating managed objects. Each managed object instance is in exactly one Home. At a minimum, each managed object has a unique, Primary Key that defines its identity within the Home.

Server Programmers may provide specialized Home types that add business logic, additional keys, query, and so forth. Some of these functions are also provided by the Managed Object Framework.

File name

IManagedClient.idl

Intended usage

Derived from:
- CosLifeCycle::GenericFactory
- IManageable

Supported operations

- IHome::create_object
- IHome::createFromCopyString
- IHome::createFromPrimaryKeyString
- IHome::findByPrimaryKeyString
- IHome::getManagedObjectClass
- IHome::getPrimaryKeyClass

IHome::create_object

Creates the object, using the key and criteria.

Not supported by OS/390 Component Broker

Original interface

IManagedClient::IHome

IDL syntax

```
Object create_object (in Key k,
                      in Criteria the_criteria)
  raises (NoFactory,
          InvalidCriteria,
          CannotMeetCriteria);
```

Parameters

k Name which identifies the desired object to be created.

the_criteria
Sequence of Name Value Pairs which are passed through to the factory.

Return value

Object The object that is created.

Exceptions

NoFactory
The factory cannot create the object specified by the key.

InvalidCriteria
The target factory does not understand the criteria.

CannotMeetCriteria
The target factory can not satisfy the criteria.

CORBA::NO_IMPLEMENT (OS/390)
This operation is not supported on OS/390.

IHome::createFromCopyString

This operation is used to create a new object based on the stringified form of a Copy Helper instance that is specific to its subtype. The instance of the Copy Helper must have all the required attributes set before this operation is called. The required attributes include those that make up the Primary Key of the object to be created. The correct Home is required for use of this operation. This is found using a factory finder. The input required for the factory finder is the name of the interface of the type that is being created.

Original interface

IManagedClient::IHome

IDL syntax

```
IManageable createFromCopyString(in ByteString objString)
  raises(IInvalidKey, IDuplicateKey, IInvalidCopy);
```

Parameters

objString
The stringified form of the Copy Helper to be used to find the object. This string is derived from the toString() operation on the Copy Helper instance.

Return value

IManageable
An instance of the object that was created. This value will only be returned if the object was successfully created. Due to the CORBA programming model, the returned value must first be narrowed to the specific managed object type before it can be used for business object interface specific methods.

Exceptions

IDuplicateKey
This exception is thrown if an object already exists in the Home that is used for this operation. Two objects cannot be created with the same key.

IInvalidCopy
This exception is thrown if the input ByteString could not be used to create an instance of a Copy Helper. Either the ByteString has been derived from the wrong type or it has been corrupted.

IInvalidKey
This exception is thrown if the input ByteString could not be used to create an instance of a Primary Key. Either the ByteString has been derived from the wrong type or it has been corrupted.

IHome::createFromPrimaryKeyString

This operation is used to create a new object based on the stringified form of a Primary Key that is specific to its subtype. The instance of the Primary Key must have all the required attributes set before this operation is called. The correct Home is required for use of this operation. This is found using a factory finder. The input required for the factory finder is the name of the interface of the type that is being created.

Original interface

IManagedClient::IHome

IDL syntax

```
IManageable createFromPrimaryKeyString(in ByteString key)
  raises(IInvalidKey, IDuplicateKey);
```

Parameters

key The stringified form of the Primary Key to be used to find the object. This string is derived from the toString() operation on the Primary Key instance.

Return value

IManageable
An instance of the object that was created. This value will only be returned if the object was successfully created. Due to the CORBA programming model,

the returned value must first be narrowed to the specific managed object type before it can be used for business object interface specific methods.

Exceptions

IDuplicateKey
This exception is thrown if an object already exists in the Home that is used for this operation. Two objects cannot be created with the same key.

IInvalidKey
This exception is thrown if the input ByteString could not be used to create an instance of a Primary Key. Either the ByteString has been derived from the wrong type or it has been corrupted.

IHome::findByPrimaryKeyString

This operation is used to find an object in a Home using a stringified form of the Primary Key that is used for that type. The object may or may not exist in the instance of the Home. If the object is successfully found then it is returned, otherwise an exception will be thrown. Note that it should not really be considered an error if the object cannot be found, it may merely mean that the object has not previously been created.

Original interface

IManagedClient::IHome

IDL syntax

```
IManageable findByPrimaryKeyString(in ByteString key)
  raises(IInvalidKey, INoObjectWKey);
```

Parameters

key The stringified form of the Primary Key to be used to find the object. This is usually generated by doing a toString() on the Primary Key instance.

Return value

IManageable
An instance of the object that was found. This value will only be returned if the object was successfully found. Because the CORBA programming model the returned value must first be narrowed to the specific managed object type before it can be used to run methods specified on the business object interface.

Exceptions

IInvalidKey
This exception is thrown if the input ByteString could not be used to create an instance of a Primary Key. Either the ByteString has been derived from the wrong type or it has been corrupted.

INoObjectWKey
This exception is thrown if no object could be found in the Home that matches *key*.

IHome::getManagedObjectClass

This operation returns the name of the type for the managed objects contained in the Home. What gets returned is the name of the client interface. For example “Claim”. This is the same as the interface name which would be used to do a find_factory_from_string() operation.

Original interface

IManagedClient::IHome

IDL syntax

```
string getManagedObjectClass();
```

Return value

string The name distinguishing the kind of the managed objects contained in the Home.

IHome::getPrimaryKeyClass

This operation returns the name of the type for the Primary Key objects associated with the managed objects that the home instance manufactures. What gets returned is the name of the client interface for the key class. For example “ClaimKey”, the ClaimKey could be used to do a ClaimKey::_create() of an empty instance of the kind of key that this home accepts on createFromPrimaryKeyString and findByPrimaryKeyString methods.

Original interface

IManagedClient::IHome

IDL syntax

```
string getPrimaryKeyClass();
```

Return value

string The name of the type for the Primary Key objects associated with the managed objects contained in the Home.

IManageable Interface

This interface is the abstract base class for all objects that are managed. Managed objects are maintained by Component Broker servers. All objects that are managed objects must support the IManageable interface. Copy Helpers, however, are not managed objects and are to be subtyped from INonManageable. Managed objects have the following attributes:

- They are streamable, meaning that the contents of IManageables can be placed into a CosStream::StreamIO.
- They have a lifecycle, this means that they can created, used, and destroyed. This does not imply anything about persistence or transient life. It merely supports the semantics of life.
- They have identity, this means that one particular object can be uniquely identified. This goes beyond the values that are used to define those objects. In theory this implies that even though two different objects may have the same values for their attributes they are not truly identical. For example, if these objects are mapped to a database, if two different objects are created from the same row in that table they will be identical since the values of the attributes of an object are used to resolve identity.

File name

IManagedClient.idl

Intended usage

Derived from:
CosLifeCycle::LifeCycleObject
CosObjectIdentity::IdentifiableObject
CosStream::Streamable

Supported operations

IManageable::copy
IManageable::getHandleString
IManageable::getHome
IManageable::getManagedObjectName
IManageable::getPrimaryKeyString
IManageable::move
IManageable::remove

Attributes

```
readonly attribute string managedObjectName
```

IManageable::copy

This operation is defined in the LifeCycle Service by the LifeCycleObject interface. Its intent is to make a copy of an object, using the FactoryFinder specified by the there parameter and a set of criteria that is understood by the factory.

This operation is not allowed for the Managed Object Framework. If it is called a CORBA::NO_IMPLEMENT exception is thrown.

Original interface

IManagedClient::IManageable

IDL syntax

```
CosLifeCycle::LifeCycleObject_ptr copy(
      CosLifeCycle::FactoryFinder_ptr there,
      const CosLifeCycle::Criteria & the_criteria)
   raises(NoFactory, NotCopyable, InvalidCriteria, CannotMeetCriteria);
```

Parameters

there The FactoryFinder that will make a copy of this object.

the_criteria
The criteria that the FactoryFinder will use to make a copy of the object.

Return value

CosLifeCycle::LifeCycleObject_ptr
A pointer to the copy of the object.

Exceptions

CORBA::NO_IMPLEMENT

IManageable::getHandleString

This operation returns a ByteString that is the representation of the Handle that points to this object. A default implementation of this operation is provided by the Managed Object Framework. Overriding this operation in a subtype of a business object should be done based on the same criteria that are used to determine if an override is needed in the base class. For example, this operation only needs to be overridden if this object is going to be the target of a relationship with another object and you want that relationship to be stored using a handle other than the stringified object reference.

Original interface

IManagedClient::IManageable

IDL syntax

```
ByteString getHandleString();
```

Return value

ByteString
A ByteString that is the representation of the handle that points to this object.

IManageable::getHome

This operation returns the instance of the Home that the managed object is contained within. This value can then be used to perform other operations on the Home. This operation may return NULL if the instance is a copy and not really a managed object in a Home. Use this method if you have a reference to an object and want to create another object in the same home. This avoids the need to do a factory-finder or more complex home finding operation.

The Managed Object Framework provides a sufficient default implementation for this method. Business object developers are not required to override this operation.

Note: On CB/390, system home collections are not themselves members of a parent home collection. Execution of the getHome() method on a CB/390 native home collection will throw the NO_IMPLEMENT exception.

Original interface

IManagedClient::IManageable

IDL syntax

```
IHome getHome();
```

Return value

IHome An instance of a Home that contains this managed object.

IManageable::getManagedObjectName

This operation is the getter of the attribute managedObjectName. The attribute is provided so the name of the object can be retrieved. The default implementation is NULL. When implemented by the business object provider, this is meant to return a path which can be resolved to using the naming service. It could be set in initForCreation() by specific methods added to managed objects or by returning a value directly in this operation. When implementing specialized homes this operation must be overridden if the default handle pattern of HomeName and Key is used. This is required for functions such as Collections to function properly.

Original interface

IManagedClient::IManageable

IDL syntax

```
string getManagedObjectName();
```

Return value

string The string which is the name of the managed object. The default value is NULL.

Example

```
PersonCLOHomeBO_I.cpp
char* PersonCLOHomeBO_Impl::managedObjectName()
{
  //Version identifier DCE:DD7B1F5D-CFD0-11d1-870D-400011532978:1
  //Insert Method modifications here
  return(CORBA::string_dup(
     "PersonCLO.object interface/PersonCLOMOFactory.object home"));
  //End Method modifications here
}
```

IManageable::getPrimaryKeyString

This operation returns a ByteString that contains the PrimaryKey subclass for the business object type being implemented. This could be used to extract the identity of a business object for the purpose of using it at a later time to locate the same business object using a findByPrimaryKeyString() method on a Home.

The most common way of implementing this operation is to create an instance of the PrimaryKey subtype, set the values into this object, and return the ByteString value.

Original interface

IManagedClient::IManageable

IDL syntax

```
ByteString getPrimaryKeyString();
```

Return value

ByteString
The ByteString of the Primary Key subtype for this managed object. This value can then be used to create a Primary Key subtype.

IManageable::move

This operation is defined in the LifeCycle Service by the LifeCycleObject interface. It's intent is to move an object, using the FactoryFinder specified by the there parameter and a set of criteria that is understood by the factory.

Original interface

IManagedClient::IManageable

IDL syntax

```
void move(CosLifeCycle::FactoryFinder_ptr there,
          const CosLifeCycle::Criteria & the_criteria)
   raises(NoFactory, NotMovable, InvalidCriteria, CannotMeetCriteria);
```

Parameters

there The FactoryFinder that will make a copy of this object.

the_criteria
The criteria that the FactoryFinder will use to make a copy of the object.

Return value

None.

Exceptions

If it is called, a CORBA::NO_IMPLEMENT will be thrown.

IManageable::remove

This operation is defined in the LifeCycle Service by the LifeCycleObject interface. It removes the object.

Original interface

IManagedClient::IManageable

IDL syntax

```
remove()
   raises(NotRemovable);
```

Parameters

None.

Return value

None.

Exceptions

See LifeCycleObject::remove for a complete description of the exceptions.

Chapter 36. IManagedCollections in the Managed Object Framework

The other modules in the Managed Object Framework are:

- CBSeriesGlobal
- IBOIMExtLocal
- IBOIMExtLocalToServer
- IBOIMInstanceManagerFriendQOS
- IBOIMManagedObjectFriendQOS
- IBOIMServiceFriendQOS
- ICollectionsBase
- IManagedAdvancedClient
- IManagedAdvancedServer
- IManagedClient
- IManagedLocal
- IManagedServer
- IRDBIMExtLocalToServer

IManagedCollections Module

Defines concrete collection interfaces composed from the more basic interfaces defined in ICollectionsBase.

A variety of interfaces for use with factory finders are provided for specifying the type of reference collections that you want to use. For example, calling find_factory_from_string passing the argument IManagedCollections::IReferenceCollection.object interface/PersistentReferenceCollectionFactory.object home creates a DB2 backed reference collection. Using the generic string IManagedCollections::IReferenceCollection.object interface returns whichever collection is configured as the default.

Transient Collections
IManagedCollections::IReferenceCollection.object interface/ TransientReferenceCollectionFactory.object home

Transient Collections Keyed
IManagedCollections::IKeyedReferenceCollection.object interface/ TransientKeyedReferenceCollectionFactory.object home

Persistent Collections
IManagedCollections::IReferenceCollection.object interface/ PersistentReferenceCollectionFactory.object home

Persistent Collections Keyed
IManagedCollections::IKeyedReferenceCollection.object interface/ PersistentKeyedReferenceCollectionFactory.object home

File name

IManagedCollections.idl

Interfaces

IManagedCollections::ICollectionHome Interface
IManagedCollections::ICommonCollection Interface
IManagedCollections::IKeyedReferenceCollection Interface
IManagedCollections::IReferenceCollection Interface

ICollectionHome Interface

Defines the operations that are common to all managed collection homes.

File name

IManagedCollections.idl

Supported operations

ICollectionHome::createCollection
ICollectionHome::createCollectionFor

ICollectionHome::createCollection

Manufactures a new collection for the collection home (this is the same as passing IManagedClient::IManageable::IManageable_RID to createCollectionFor()).

Original interface

IManagedCollection::ICollectionHome

IDL syntax

```
ICommonCollection createCollection();
```

Return value

The new collection.

ICollectionHome::createCollectionFor

Manufactures a new collection for the collection home that only holds elements of the specified type.

Original interface

IManagedCollection::ICollectionHome

IDL syntax

```
ICommonCollection createCollectionFor(in string elementTypeID);
```

Parameters

elementTypeID
Type of element that the new collection will be constrained to hold.

Return value

The new collection.

ICommonCollection Interface

Defines the operations that are common to all the collections defined in this module.

File name

IManagedCollections.idl

Intended usage

Derived from:
IMIterable Interface
IMTyped Interface
IManageable Interface

Supported operations

ICommonCollection::containsElement
ICommonCollection::getCardinality
ICommonCollection::isEmpty
ICommonCollection::removeAllElements

ICommonCollection::containsElement

Determines if a specified object is contained in the collection.

Original interface

IManagedCollections::ICommonCollection

IDL syntax

```
boolean containsElement(in IManaged Client::IManageable element);
```

Parameters

element
Manageable object to be searched for in the collection.

Return value

Returns true (1) if the specified element is contained in the collection, or false (0) if the specified element is not contained in the collection.

ICommonCollection::getCardinality

Determines the number of elements in the collection.

Original interface

IManagedCollections::ICommonCollection

IDL syntax

```
unsigned long getCardinality();
```

Return value

An unsigned long value representing the number of elements in the collection.

ICommonCollection::isEmpty

Determines if the collection is empty (contains no elements).

Original interface

IManagedCollections::ICommonCollection

IDL syntax

```
boolean isEmpty();
```

Return value

Returns true (1) if the collection is empty, or false (0) if the collection is not empty (contains elements).

ICommonCollection::removeAllElements

Removes all the elements from the collection.

Original interface

IManagedCollections::ICommonCollection

IDL syntax

```
void removeAllElements();
```

IKeyedReferenceCollection Interface

Defines the operations that are common to all keyed reference collections.

File name

IManagedCollections.idl

Intended usage

Derived from:
- IMIterableUpdate Interface
- IMKeyable Interface
- IMKeyedUpdate Interface
- ICommonCollection Interface

IReferenceCollection Interface

Defines the operations that are common to all reference collections.

File name

IManagedCollections.idl

Intended usage

Derived from:
- IMIterableUpdate Interface
- IMNonKeyedUpdate Interface
- ICommonCollection Interface

Chapter 37. IManagedLocal in the Managed Object Framework

The other modules in the Managed Object Framework are:

- CBSeriesGlobal
- IBOIMExtLocal
- IBOIMExtLocalToServer
- IBOIMInstanceManagerFriendQOS
- IBOIMManagedObjectFriendQOS
- IBOIMServiceFriendQOS
- ICollectionsBase
- IManagedAdvancedClient
- IManagedAdvancedServer
- IManagedClient
- IManagedCollections
- IManagedServer
- IRDBIMExtLocalToServer

IManagedLocal Module

This is the module that provides the mechanisms for those objects that are not managed by a Component Broker server. This module provides the abstract base interfaces for these objects. It also provides support for Keys and Handles. Keys are used to identify particular managed objects. Each managed object will have unique values for attributes that make up its primary key. Handles provide a mechanism for referencing objects.

File name

IManagedLocal.idl

Interfaces

IManagedLocal::IHandle Interface
IManagedLocal::IKey Interface
IManagedLocal::ILocalOnly Interface
IManagedLocal::INonManageable Interface
IManagedLocal::IPrimaryKey Interface
IManagedLocal::IUniqueKey Interface

Exceptions

IGetHandleFailed
IInvalidKey
IInvalidString

IHandle Interface

Handles are used to reference objects. This is an abstraction of the OMG Stringified Object References. The default implementation of IHandle is simply to wrapper an OMG Reference. A Server Application Programmer may provide alternative implementations that provide more robust referencing capability, for example the name of a collection and the Primary Key value into a collection. This would enable a server to be reconfigured and Homes to be moved, while ensuring the integrity of the handle.

File name

IManagedLocal.idl

Intended usage

Derived from:
INonManageable

Supported operations

IHandle::getObject
IHandle::isHandleFor

IHandle::getObject

This operation returns the managed object that the instance of the Handle has been created with. The default implementation of the framework returns NULL. The business object provider must override this operation with an implementation that is specific to their subclass. The business object provider decides which handle pattern a particular business object will support when creating the business object and is implicitly (by virtue of Object Builder generation) responsible for that implementation.

Original interface

IManagedLocal::IHandle

IDL syntax

```
IManagedClient::IManageable getObject()
  raises(IGetHandleFailed);
```

Exceptions

IGetHandleFailed
This exception is thrown if a failure occurred while retrieving the object for this handle.

IHandle::isHandleFor

This operation returns an indicator if the instance of the Handle is the handle for the input object. The default implementation of this support uses a random number to associate an object with a handle when the handle is created. This number is checked in the implementation of this operation to check for equality. Since this implementation cannot be guaranteed to be an exact match a value of `MAYBE` will be returned if the numbers are equal. To ensure upward compatibility the value of `NO` should be used in all conditional checking.

Original interface

IManagedLocal::IHandle

IDL syntax

```
CompareResult isHandleFor(in IManagedClient::IManageable theObject);
```

Parameters

theObject
The Handle is checked against this input object to see if it is being pointed to any this Handle.

Return value

CompareResult
The object with which this Handle instance is created.

Example

```
if (myHandle.isHandleFor(someIManageableObject) ==
  IManagedLocal::INonManageable::IHandle::NO)
  // The Handle is not for this object

else
  // The Handle is for this object
```

IKey Interface

Keys are used to identify particular managed objects. The key interfaces are abstract. The implementor of a managed object must also implement a subclass of the appropriate abstract key interfaces. These Key classes are used with operations on Homes to find and create these managed objects. Every managed object class has a Primary Key class that is used to find (and create) objects of that type. An instance of a Key is always local to the client process and language.

File name

IManagedLocal.idl

Intended usage

Derived from:
INonManageable

Supported operations

IKey::getHashForMO
IKey::getName
IKey::isEqualToKey
IKey::isEqualToKeyString

IKey::getHashForMO

This operation will return a hash value for the managed object identified by the key. This hash value can then be used to narrow the number of managed objects that need to be compared via the isEqualToKey or isEqualToKeyString methods (which can be expensive).

Note: This method is only currently supported and utilized on OS/390 Component Broker. The default implementation on Component Broker for workstations is to throw the CORBA::NO_IMPLEMENT exception.

Original interface

IManagedLocal::IKey

IDL syntax

```
unsigned long getHashForMO( in unsigned long hashSize );
```

Parameters

hashSize
May be used as a guideline for the primary key implementor in designing the hash algorithm (i.e. do something different for the hash generation for a table of size 12 versus size 12001 or whatever is the prime is that is close to this). The implementor is not required to return a value within the range of 0 to hashSize.

Return value

Returns the generated hash value for the managed object identified by this key.

IKey::getName

This operation returns a string which is the name of the class. All concrete classes that are derived from IKey should provide an implementation for this operation.

Original interface

IManagedLocal::IKey

IDL syntax

```
string getName();
```

Return value

string The name of the interface of the IKey subclass that is implementing this operation. To clarify, This means that in the above example, ClaimKey is the IDL interface. The implementation of that interface might be in C++ or Java, but the interface returned by this method is independent of the particular implementation of the key.

Example

```
char * ClaimKey_Impl::getName()
{
  return CORBA::string_dup("ClaimKey");
}
```

IKey::isEqualToKey

This operation provides a way of checking if two instances of a Key are equal. Two instances of a Key should be considered equal if the attributes that make up the keys contain the same values. The Managed Object Framework provides a default implementation of this operation that compares the stringified values of the keys. This operation can be optionally implemented by the Object provider. The implementation of this method should use the get methods on the keys and compare the values to see if the key is for the same object.

Original interface

IManagedLocal::IKey

IDL syntax

```
boolean isEqualToKey(in IKey aKey)
  raises (IInvalidKey);
```

Parameters

aKey An instance of a Key to compare.

Return value

Returns true (1) if the types and values of the input Key are the same; otherwise returns false(0).

Exceptions

IInvalidKey
This exception is thrown if the input parameters is not a subclass of IKey.

IKey::isEqualToKeyString

This operation provides a way of checking if two instances of a Key are equal. Two instances of a Key should be considered equal if the attributes that make up the keys contain the same values. The Managed Object Framework provides a default implementation of this operation that compares the stringified values of the keys. This operation is similar to isEqualToKey() except it uses the stringified versions of the keys. This operation can be optionally implemented by the Object provider. The implementation of this method should use the get methods on the keys and compare the values to see if the key is for the same object.

Original interface

IManagedLocal::IKey

IDL syntax

```
boolean isEqualToKeyString(in ByteString aKeyString)
  raises (IInvalidKey);
```

Parameters

aKeyString
A ByteString that was generated using the toString() operation of a subclass of an IKey.

Return value

Returns true (1) if the ByteString of the input Key is the same; otherwise returns false(0).

Exceptions

IInvalidKey
This exception is thrown if the input parameters is not a subclass of IKey.

ILocalOnly Interface

This interface is one of the abstract building blocks of the Managed Object Framework. It provides a consistent programming model between local and remotable objects in the Component Broker environment. It inherits directly from CORBA::Object and is used as the basis for objects that are not managed. Interfaces that inherit from it are not manageable and cannot be accessed remotely. It is not used directly by Application developers.

File name

IManagedLocal.idl

Intended usage

Derived from:
CORBA::Object

INonManageable Interface

This is the base interface for objects that are not managed on a Component Broker server. These include Keys as well as transient, local objects that may be used to create managed objects. Copy Helper objects inherit directly from this interface. By virtue of inheriting from CosStream::Streamable, these objects are all capable of being stored as a ByteString and sent to a Component Broker server. For example, a Primary Key is created in a client program, converted to a ByteString and sent to a Component Broker server to create a new instance of a Primary Key using the same values as the original Primary Key.

If arbitrary local-only objects are required as part of the programming model either on the client or on the server, this interface should be used instead of the ILocalOnly interface since the object is remoteable, that is it could be shipped from process to process.

File name

IManagedLocal.idl

Intended usage

Derived from:
ILocalOnly,
CosStream::Streamable

Supported operations

INonManageable::toString
INonManageable::fromString

INonManageable::toString

This operation stores the data from an object into a ByteString and returns that ByteString. This returned data then can be used with some standard operations on a Home to create Managed objects on a Component Broker server. It could also be stored somewhere (e.g. a file) to be used in conjunction with the fromString() operation to later re-create the object.

Original interface

IManagedLocal::INonManageable

IDL syntax

```
ByteString toString();
```

Return value

ByteString
The data of the object in ByteString format. This data can be used in conjunction with the fromString operation to re-create the object.

INonManageable::fromString

This operation creates a new object from the ByteString that was passed as a parameter. The ByteString must have been returned from a call to toString() of the same interface type. The toString() and fromString() operations are tightly coupled, that is each operation assumes the same data members will be stored into and retrieved from the string and in the same order. Be careful of migration issues if attributes are added or removed from an interface. In this case if a ByteString returned by a toString() operation is saved from the original implementation and then the internal representation of the interface is changed (e.g. data members are added or removed) then the toString() and fromString() operations will need to be updated. The ByteString that was saved will not work with this newer version of the interface because the structure of what was stored differs from what is expected in the new version of fromString().

Original interface

IManagedLocal::INonManageable

IDL syntax

```
void fromString(in ByteString theString)
  raises(IInvalidString);
```

Parameters

theString
The ByteString that was returned from the toString() operation of this same interface.

Exceptions

IInvalidString
The ByteString that was passed as theString was not a valid ByteString for the class. Either the string has been corrupted, was created from a different interface or was created from a different version of this same interface type. Recovery is to pass in a ByteString that is a valid representation of an instance of this interface.

IPrimaryKey Interface

The current implementation of the Managed Object Framework requires a Primary key be defined for each managed object. A primary key can be defined by one or more attributes of its associated managed object. Each instance of a managed object must have unique values for attributes that make up its primary key. When you create an instance of a Primary Key, the key must be set by one or more attributes on the Primary Key object. When all the key attributes have been set, the Primary Key object is now usable.

File name

IManagedLocal.idl

Intended usage

Derived from:
IUniqueKey Interface

IUniqueKey Interface

This interface provides another abstract interface for support Key. It is typically not currently used by Application developers but is intended for future expansion beyond Primary Key support. It provides a base for keys that are unique but are not required to be Primary keys. A combination of other attributes other than those that are used to identify the Primary key may be used to create a Unique Key. For example, a Social Security number may be used for the Primary key while an employee number may be used for a Unique Key. No additional operations are defined for this interface.

File name

IManagedLocal.idl

Intended usage

Derived from:
IKey Interface

Chapter 38. IManagedServer in the Managed Object Framework

The other modules in the Managed Object Framework are:

- CBSeriesGlobal
- IBOIMExtLocal
- IBOIMExtLocalToServer
- IBOIMInstanceManagerFriendQOS
- IBOIMManagedObjectFriendQOS
- IBOIMServiceFriendQOS
- ICollectionsBase
- IManagedAdvancedClient
- IManagedAdvancedServer
- IManagedClient
- IManagedCollections
- IManagedLocal
- IRDBIMExtLocalToServer

IManagedServer Module

This module defines the base interfaces for the server (object provider) side of the Managed Object Framework. The most important concept in this module is the business object/data object model. This combination of business object with its associated data object is referred to as a managed object assembly. The various combinations of managed object assemblies are further described in the following sections. The interfaces in this module are intended to be implemented by the object provider, and used (only) by an instance manager.

File name

IManagedServer.idl

Interfaces

IManagedServer::IDataObject Interface
IManagedServer::IManagedObject Interface
IManagedServer::IManagedObjectWithoutDataObject Interface
IManagedServer::IManagedObjectWithCachedDataObject Interface
IManagedServer::IManagedObjectWithDataObject Interface
IManagedServer::IWrappable Interface

Exceptions

ICreationFailed
IDestructionFailed
IPassiveFailed
IReactivationFailed
ISyncronizationFailed

IDataObject Interface

This is the base class for all data objects for managed objects. The data object manages the essential state of the business object. The interface for the data object contains one attribute for each piece of the essential state of the business object. This does not imply that there is a one-to-one mapping of all the attributes of the business object.

There are several different data object patterns that can be used to store the data. These range from transient to various forms of persistence and described more fully in the following sections of IManagedObjectWith... interfaces.

This interface is enough to allow development of the business object logic, without having to actually implement the underlying data access methods. However, before the business object can be tested, the data object interface must be implemented.

There are several considerations when defining and naming the data object and its attributes. First, be careful when naming the attributes for the data object. The Query Service operates more efficiently if the attribute names on the business object are the same as the attribute names on the data object. Second, while attributes on the business object interface may not be a good idea, depending on the amount of encapsulation desired, they are a convenient way of declaring get and set methods on the data object. Because data objects are only used by business objects, and by the application adapter, encapsulation is not usually a concern.

File name

IManagedServer.idl

Intended usage

Derived from:
    ILocalOnly Interface

IManagedObject Interface

IManagedObject is the base class for the managed objects of the Managed Object Framework. It defines no new operations or attributes, it is merely an abstract building block for the other managed object assembly types.

File name

IManagedServer.idl

Intended usage

Derived from:
IManageable Interface

IManagedObjectWithCachedDataObject Interface

This interface is provided for support of managed object assemblies that have state data and the data objects are configured to use the Caching pattern. These objects may or may not have persistent state They may have the advantage of being backed by a datastore and can be recreated at a later time with the stored values.

The Object Provider is expected to implement the initForCreation() and initForReactivation() operations, if only to save a reference to the data object which the Instance Manager will provide to it. The Object Provider might also want to do its own management of resources, based on the life-cycle of the managed object. The uninitForDestruction() and uninitForPassivation() operations only need to be implemented by the Object Provider if they are doing their own resource management.

File name

IManagedServer.idl

Intended usage

Derived from:
IManagedObject

Supported operations

IManagedObjectWithCachedDataObject::initForCreation
IManagedObjectWithCachedDataObject::initForReactivation
IManagedObjectWithCachedDataObject::syncFromDataObject
IManagedObjectWithCachedDataObject::syncToDataObject
IManagedObjectWithCachedDataObject::uninitForDestruction
IManagedObjectWithCachedDataObject::uninitForPassivation

IManagedObjectWithCachedDataObject::initForCreation

This operation is invoked on every managed object when the object is initially created. The operation is equivalent to a C++ constructor. Unlike a C++ constructor, by the time this operation is called, the object essential data has already been initialized. In Component Broker, an object state is initialized either by the createFromPrimaryKeyString() or the createFromCopyString() operations.

The data object can be used for any initialization tasks that need to be done. There is no guarantee as to the number of attributes in the data object that are validly set. That would be based on whether a Copy Helper or Primary Key was used to perform the

create. Caching business objects should take the opportunity to load the cache for the business object with any relevant values from the data object. At this point, the data object has values for any Key or Copy Helper data that was passed along during the create. In fact, the Primary Key information must be extracted from the data object and placed into the cache of the business object.

Original interface

IManagedServer::IManagedObjectWithCachedDataObject

IDL syntax

```
void initForCreation(in IDataObject theDO)
  raises (ICreationFailed);
```

Parameters

theDO This parameter is the data object that is associated with this managed object. It can be used during object creation to initialize attributes in the managed object.

Exceptions

ICreationFailed
This exception is thrown if an internal failure occurred during object creation.If this occurs then there are other messages in the Component Broker activity log that can be used for further information about the error.

IManagedObjectWithCachedDataObject::initForReactivation

This operation is invoked on every managed object when the object is being recreated in storage. This is the Managed Object Framework equivalent of an operating system paging storage in from disk. Because the managed object is being put into storage, it needs to make sure that any resources that it was managing are ready to be used. For example, if the managed object was managing its own network connection, the implementation of this operation would need to re-establish that connection.

In addition the data object input parameter on this operation needs to be saved for later usage. While it is necessary to get the pointer to the data object in this operation, the data object should not be used in any way during this operation. This operation has no access to the essential state stored in the data object.

Original interface

IManagedServer::IManagedObjectWithCachedDataObject

IDL syntax

```
void initForReactivation(in IDataObject theDO)
  raises (IReactivationFailed);
```

Parameters

theDO This parameter is the data object that is associated with this managed object. It can should be stored for later usage but not used during this operation.

Exceptions

IReactivationFailed
This exception is thrown if an internal failure occurred during object reactivation.

IManagedObjectWithCachedDataObject::syncFromDataObject

This operation is used to load the business object with the data that is contained in the data object. The initForCreation() operation stores a pointer to the data object that is used in this operation. In this operation the data object must be accessed to initialize the attributes of the business object.

To avoid duplicating this code, another operation can be used, for example initializeState(),that is called from both the initForCreation() and syncFromDataObject() operations.

This operation is called by the Managed Object Framework at the appropriate time. It should not be called by the business object in any of its operations.

Original interface

IManagedServer::IManagedObjectWithCachedDataObject

IDL syntax

```
void syncFromDataObject()
  raises (ISyncronizationFailed);
```

Exceptions

ISynchronizationFailed
This exception is thrown if an internal failure occurred during object synchronization. If this occurs then there are other messages in the Component Broker activity log that can be used for further information about the error.

IManagedObjectWithCachedDataObject::syncToDataObject

This operation is used to store the business object to the data that is contained in the data object. This puts the data object in a state in which it can deal with the underlying resource manager, and ensure that this data is properly stored persistently, using the correct transactions interfaces to the underlying resource manager.

This operation is called by the Managed Object Framework at the appropriate time. It should not be called by the business object in any of its operations.

Original interface

IManagedServer::IManagedObjectWithCachedDataObject

IDL syntax

```
void syncToDataObject()
  raises (ISyncronizationFailed);
```

Exceptions

ISynchronizationFailed

This exception is thrown if an internal failure occurred during object synchronization.If this occurs then there are other messages in the Component Broker activity log that can be used for further information about the error.

IManagedObjectWithCachedDataObject::uninitForDestruction

This operation is invoked on every managed object when the object is being destroyed. This method is the Managed Object Framework equivalent to a C++ destructor. If the managed object was managing its own resources then this operation is where those resources would be released.

The data object can be used for anything that is needed. Information can be gotten out of the data object at this point. This is also an opportunity to enforce any referencial integrity constraints. For example, this might mean removing an object that it was referencing.

Original interface

IManagedServer::IManagedObjectWithCachedDataObject

IDL syntax

```
void uninitForDestruction()
  raises (IDestructionFailed);
```

Exceptions

IDestructionFailed

This exception is thrown if an internal failure occurred during object destruction.This exception is thrown if an internal failure occurred during object destruction.If this occurs then there are other messages in the Component Broker activity log that can be used for further information about the error.

IManagedObjectWithCachedDataObject::uninitForPassivation

This operation is invoked on every managed object when the object is being temporarily removed from storage. This is the Managed Object Framework equivalent of an operating system paging storage out to disk. Because the managed object is removed from storage, it is no longer using any resources that it is referencing. The choice can be made of releasing any held resources at this time, or continuing to hold the resources. If the managed object is not managing resources, or if there is a need to hold onto the resources, then no implementation of this operation is required.

Original interface

IManagedServer::IManagedObjectWithCachedDataObject

IDL syntax

```
void uninitForPassivation()
  raises (IPassivationFailed);
```

Exceptions

IPassivationFailed

This exception is thrown if an internal failure occurred during object passivation.If this occurs then there are other messages in the Component Broker activity log that can be used for further information about the error.

IManagedObjectWithDataObject Interface

This interface is provided for support of managed object assemblies that have state data and the data objects are not configured to use the Caching pattern. These objects may or may not have persistent state They may have the advantage of being backed by a datastore and can be recreated at a later time with the stored values.

The Object Provider is expected to implement the initForCreation() and initForReactivation() operations, if only to save a reference to the data object which the Instance Manager will provide to it. The Object Provider might also want to do its own management of resources, based on the life-cycle of the managed object. The uninitForDestruction() and uninitForPassivation() methods only need to be implemented by the Object Provider if they are doing their own resource management.

File name

IManagedServer.idl

Intended usage

Derived from:
IManagedObject

Supported operations

IManagedObjectWithDataObject::initForCreation
IManagedObjectWithDataObject::initForReactivation
IManagedObjectWithDataObject::uninitForDestruction
IManagedObjectWithDataObject::uninitForPassivation

IManagedObjectWithDataObject::initForCreation

This operation is invoked on every managed object when the object is initially created. The operation is equivelant to a C++ constructor. Unlike a C++ constructor, by the time this operation is called, the object essential data has already been initialized. In Component Broker, an object state is initialized either by the createFromPrimaryKeyString() or the createFromCopyString() operations.

The data object can be used for any initialization tasks that need to be done. There is no guarantee as to the number of attributes in the data object that are validly set. That would be based on whether a Copy Helper or Primary Key was used to perform the create. Caching business objects should take the opportunity to load the cache for the business object with any relevant values from the data object. At this point, the data object has values for any Key or Copy Helper data that was passed along during the create. In fact, the Primary Key information must be extracted from the data object and placed into the cache of the business object. This is usually accomplished by invoking the private, language specific, non-remotable method called initializeState() from inside of initForCreation().

Original interface

IManagedServer::IManagedObjectWithDataObject

IDL syntax

```
void initForCreation(in IDataObject theDO)
  raises (ICreationFailed);
```

Parameters

theDO This parameter is the data object that is associated with this managed object. It can be used during object creation to initialize attributes in the managed object.

Exceptions

ICreationFailed
This exception is thrown if an internal failure occurred during object creation. If this occurs,then there are other messages in the Component Broker activity log that can be used for further information about the error.

IManagedObjectWithDataObject::initForReactivation

This operation is invoked on every managed object when the object is being recreated in storage. This is the Managed Object Framework equivalent of an operating system paging storage in from disk. Because the managed object is being put into storage, it needs to make sure that any resources that it was managing are ready to be used. For example, if the managed object was managing its own network connection, the implementation of this operation would need to re-establish that connection.

In addition the data object input parameter on this operation needs to be saved for later usage. While it is necessary to get and retain the pointer to the data object in this operation, the data object should not be used in any way during this operation. This operation has no access to the essential state stored in the data object. A retrieve on the dataobject has not yet been done when this method is called.

Original interface

IManagedServer::IManagedObjectWithDataObject

IDL syntax

```
void initForReactivation(in IDataObject theDO)
  raises (IReactivationFailed);
```

Parameters

theDO This parameter is the data object that is associated with this managed object. It can should be stored for later usage but not used during this operation.

Exceptions

IReactivationFailed
This exception is thrown if an internal failure occurred during object reactivation.If this occurs then there are other messages in the Component Broker activity log that can be used for further information about the error.

IManagedObjectWithDataObject::uninitForDestruction

This operation is invoked on every managed object when the object is being destroyed. This method is the Managed Object Framework equivalent to a C++ destructor. If the managed object was managing its own resources then this operation is where those resources would be released.

The data object can be used for anything that is needed. Information can be gotten out of the data object at this point. This is also an opportunity to enforce any referencial integrity constraints. For example, this might mean removing an object that it was referencing.

Original interface

IManagedServer::IManagedObjectWithDataObject

IDL syntax

```
void uninitForDestruction()
  raises (IDestructionFailed);
```

Exceptions

ICreationFailed

This exception is thrown if an internal failure occurred during object destruction.If this occurs then there are other messages in the Component Broker activity log that can be used for further information about the error.

IManagedObjectWithDataObject::uninitForPassivation

This operation is invoked on every managed object when the object is being temporarily removed from storage. This is the Managed Object Framework equivalent of an operating system paging storage out to disk. Because the managed object is removed from storage, it is no longer using any resources that it is referencing. The choice can be made of releasing any held resources at this time, or continuing to hold the resources. If the managed object is not managing resources, or if there is a need to hold onto the resources, then no implementation of this operation is required.

Original interface

IManagedServer::IManagedObjectWithDataObject

IDL syntax

```
void uninitForPassivation()
  raises (IPassivationFailed);
```

Exceptions

IPassivationFailed

This exception is thrown if an internal failure occurred during object passivation.If this occurs then there are other messages in the Component Broker activity log that can be used for further information about the error.

IManagedObjectWithoutDataObject Interface

This interface is provided for support of managed object assemblies that are transient and have no state data to store, even temporarily. These objects have no persistent state and their lifetime is constrained to no more than the lifetime of the Component Broker server that they have been created within.

Note: This interface is not currently supported by any Component Broker application adaptor on any platform.

File name

IManagedServer.idl

Intended usage

Derived from:
IManagedObject

Supported operations

IManagedObjectWithoutDataObject::initForCreation
IManagedObjectWithoutDataObject::uninitForDestruction

IManagedObjectWithoutDataObject::initForCreation

This operation is invoked on every managed object when the object is initially created. The operation is equivalent to a C++ constructor. Unlike a C++ constructor, by the time this operation is called, the object essential data has already been initialized. In Component Broker, an object state is initialized either by the createFromPrimaryKeyString() or the createFromCopyString() operations.

Original interface

IManagedServer::IManagedObjectWithoutDataObject

IDL syntax

```
void initForCreation()
  raises (ICreationFailed);
```

Exceptions

ICreationFailed
This exception is thrown if an internal failure occurred during object creation.If this occurs, then there are other messages in the Component Broker activity log that can be used for further information about the error.

IManagedObjectWithoutDataObject::uninitForDestruction

This operation is invoked on every managed object when the object is being destroyed. This method is the Managed Object Framework equivalent to a C++ destructor. If the managed object was managing its own resources then this operation is where those resources would be released.

Original interface

IManagedServer::IManagedObjectWithoutDataObject

IDL syntax

```
void uninitForDestruction()
  raises (IDestructionFailed);
```

Exceptions

ICreationFailed

This exception is thrown if an internal failure occurred during object creation. If this occurs, then there are other messages in the Component Broker activity log that can be used for further information about the error.

IWrappable Interface

This interface is not part of the programming model and should not be directly invoked or overridden.

Chapter 39. INotifyChannelAdminManagedClient in the Notification Service

The other modules in the Notification Service are:

- CosNotification
- CosNotifyChannelAdmin
- CosNotifyComm
- CosNotifyFilter
- INotifyFilterManagedClient

INotifyChannelAdminManagedClient Module

Allows applications to create or find an EventChannel object on the server.

File name

INotifyChannelAdminManagedClient.idl

Intended usage

The INotifyChannelAdminManagedClient module is an abstract class from which actual EventChannelFactory implementation inherits. The EventChannelFactory object is implemented by the Notification Service in the Component Broker. It is not intended to be implemented or modified by any customers.

Not supported by OS/390 Component Broker

Interfaces

INotifyChannelAdminManagedClient::EventChannelFactory Interface

EventChannelFactory Interface

Defines three operations: create EventChannel, find EventChannel, and create a visible EventChannel in the CDS.

File name

INotifyChannelAdminManagedClient.idl

Intended usage

An abstract base class from which actual create or find EventChannel implementation inherits.

IDL syntax

```
interface EventChannelFactory : CosNotifyChannelAdmin::EventChannelFactory,
                                IManagedClient::IHome
  {
      EventChannel createEventChannel(
         out ByteString key,
         in CosNotification::QoSProperties initial_qos,
         NULL,
         out CosNotifyChannelAdmin::ChannelID id);
      EventChannel findEventChannel(
         in ByteString key,
         out CosNotifyChannelAdmin::ChannelID id);
      EventChannel createVisibleEventChannel(
         out ByteString key,
         in string relativeName,
         in boolean visibleInCellNameTree,
         in boolean visibleInHostNameTree,
         in boolean visibleInWorkGroupNameTree,
         in CosNotification::QoSProperties initial_qos,
         NULL,
         out CosNotifyChannelAdmin::ChannelID id);
  };
```

Supported operations

EventChannelFactory::createEventChannel
EventChannelFactory::createVisibleEventChannel
EventChannelFactory::findEventChannel

EventChannelFactory::createEventChannel

Returns both an EventChannel object and a ByteString as key which can be used to find an EventChannel object.

Original interface

INotifyChannelAdminManagedClient::EventChannelFactory

IDL syntax

```
INotifyChannelAdminManagedClient::EventChannel
   createEventChannel(out ByteString key,
                      in CosNotification::QoSProperties initial_qos,
                      NULL,
                      out CosNotifyChannelAdmin::ChannelID id);
```

Parameters

key A ByteString used as a key for the created EventChannel object.

initial_qos
A CosNotification::QoSProperties used as the initial value.

id A CosNotifyChannelAdmin::ChannelID used as the returned channel id.

Return value

INotifyChannelAdminManagedClient::EventChannel
A new EventChannel object.

Example

```
CosNotifyChannelAdmin::ChannelID cid;
CosNotification::QoSProperties *qos;
ByteString_var key;
INotifyChannelAdminManagedClient::EventChannelFactory_var ef = NULL;
CosNotifyChannelAdmin::EventChannel_var ec = NULL;
...
ec = ef->createEventChannel(key, *qos, NULL, cid);
```

EventChannelFactory::createVisibleEventChannel

Returns both an EventChannel object a ByteString as key which can be used to find an EventChannel object, and a ChannelID. The user can decide whether or not the created EventChannel object should be visible in the CDS under the cell name tree, host name tree, or workgroup name tree.

Original interface

INotifyChannelAdminManagedClient::EventChannelFactory

IDL syntax

```
INotifyChannelAdminManagedClient::EventChannel createVisibleEventChannel(
   out ByteString key,
   in string relativeName,
   in boolean visibleInCellNameTree,
   in boolean visibleInHostNameTree,
   in boolean visibleInWorkGroupNameTree,
   in CosNotification::QoSProperties initial_qos,
   NULL,
   out CosNotifyChannelAdmin::ChannelID id);
```

Parameters

key Indicates a ByteString for the created EventChannel object.

relativeName
A string name to be used under the name tree.

visibleInCellNameTree
A boolean value indicates whether or not the relativeName should appear under the cell name tree.

visibleInHostNameTree
A boolean value indicates whether or not the relativeName should appear under the host name tree.

visibleInWorkGroupNameTree
A boolean value indicates whether or not the relativeName should appear under the workgroup name tree.

initial_qos
A CosNotification::QoSProperties to be used as the initial setting.

id A CosNotifyChannelAdmin::ChannelID which can be used to reference this EventChannel.

Return value

INotifyChannelAdminManagedClient::EventChannel
A new EventChannel object.

Remarks

This method is called by the clients and implemented by the Notification Server in the Component Broker. It is not intended to be implemented or modified by any customers.

Example

```
CosNotifyChannelAdmin::ChannelID cid;
CosNotification::QoSProperties *qos;
ByteString_var key;
INotifyChannelAdminManagedClient::EventChannelFactory_var ef = NULL;
CosNotifyChannelAdmin::EventChannel_var ec = NULL;
...
ec = ef->createVisibleEventChannel(key, "My_NC", 0, 1, 0, *qos, NULL, cid);
```

EventChannelFactory::findEventChannel

Finds an EventChannel object on the server from a ByteString key or ChannelID.

Original interface

INotifyChannelAdminManagedClient::EventChannelFactory

IDL syntax

```
INotifyChannelAdminManagedClient::EventChannel findEventChannel(
                                    in ByteString key,
                                    out CosNotifyChannelAdmin::ChannelID id);
```

Parameters

key A ByteString used as the key of the EventChannel object.

id A CosNotifyChannelAdmin::ChannelID used to locate the EventChannel.

Return value

INotifyChannelAdminManagedClient::EventChannel
A new EventChannel object.

Remarks

This operation is called by the clients and implemented by the Notification Server in the Component Broker. It is not intended to be implemented or modified by you, the developer.

Example

```
CosNotifyChannelAdmin::ChannelID cid;
ByteString_var key;
INotifyChannelAdminManagedClient::EventChannelFactory_var ef = NULL;
CosNotifyChannelAdmin::EventChannel_var ec = NULL;
...
ec = ef->findEventChannel(key, cid);
```

Chapter 40. INotifyFilterManagedClient in the Notification Service

The other modules in the Notification Service are:

- CosNotification
- CosNotifyChannelAdmin
- CosNotifyComm
- CosNotifyFilter
- INotifyChannelAdminManagedClient

INotifyFilterManagedClient Module

Allows applications to locate the FilterFactory object on the server.

Not supported by OS/390 Component Broker

File name

INotifyFilterManagedClient.idl

Intended usage

The INotifyFilterManagedClient module is an abstract class from which actual FilterFactory implementation inherits. The FilterFactory object is implemented by the Notification Service in the Component Broker. It is not intended to be implemented or modified by any customers.

Interfaces

INotifyFilterManagedClient::FilterFactory

Chapter 41. IRDBIMExtLocalToServer in the Managed Object Framework

The other modules in the Managed Object Framework are:

- CBSeriesGlobal
- IBOIMExtLocal
- IBOIMExtLocalToServer
- IBOIMInstanceManagerFriendQOS
- IBOIMManagedObjectFriendQOS
- IBOIMServiceFriendQOS
- ICollectionsBase
- IManagedAdvancedClient
- IManagedAdvancedServer
- IManagedClient
- IManagedCollections
- IManagedLocal
- IManagedServer

IRDBIMExtLocalToServer Module

This module extends the data object interface.

Not supported by OS/390 Component Broker

File name

IRDBIMExtLocalToServer.idl

Intended usage

This module contains interfaces which can be extended by the framework or customer code. The objects described in this module run local to the server process.

Interfaces

IRDBIMExtLocalToServer::ICachingServiceDataObject Interface
IRDBIMExtLocalToServer::IDataObject Interface

ICachingServiceDataObject Interface

This interface provides an interface that is used for one portion of the application adaptor to interact with another portion. The implementation of the methods are provided in the implementaion of the base class and should not be invoked or used in any way.

File name

IRDBIMExtLocalToServer.idl

Intended usage

The usage of these interfaces is intended to be restricted to developers of the Component Broker product.

IDataObject Interface

Need one or two sentences to describe the interface and quickly differentiate it from other interfaces.

Implementations of this interface are local-only. (Is this true?)

File name

IRDBIMExtLocalToServer.idl

Intended usage

Need a description of how this interface is intended to be used.

Ancestor interfaces

List, if applicable.

Supported operations

IRDBIMExtLocalToServer::IDataObject::setConnection

IDataObject::setConnection

This operation is used as a part of the initialization of the data object that needs to communicate with a database. The implementation of this method is provided as a part of the application supplied code. This implementation should keep track of the database name provided to be able to used it when it is needed for other methods to use.

Original interface

IRDBIMExtLocalToServer::IDataObject Interface

IDL syntax

```
void setConnection(in string dataBaseName)
  raises (IBOIMException::IDataObjectFailed);
```

Parameters

dataBaseName

This string contains the name of the data base when the essential state is located.

Exceptions

IDataObjectFailed

This exception is raised to indicate that an attempt was made to locate the database in the back end and the attempt was unsuccessful for any reason.

Chapter 42. ISessions in the Session Service

The ISessions Module is the only module in the Session Service.

Note: The Session Service is not supported by OS/390 Component Broker.

ISessions Module

This section summarizes the interfaces of the Session Service. These interfaces are heavily modeled after the CosTransactions service interfaces.

Not supported by OS/390 Component Broker

File stem

ISessions

Types

Status This enum is used to indicate the status of the current session. Status can have one of the following values:

StatusSessionActive
The session has begun but has not ended, nor has it been suspended.

StatusNoSession
There is no current session, the session has either been ended or is suspended.

StatusCheckpointInProgress
A checkpoint process is currently in progress.

StatusResetInProgress
A reset process is currently in progress.

StatusEndSessionCheckpointInProgress
An end-session with a checkpoint option process is currently in progress.

StatusEndSessionResetInProgress
An end-session with reset option process is currently in progress.

StatusEndSessionResetForceInProgress
An end-session with reset-force option process is currently in progress.

StatusVoteEndWithCheckpointPending
An endSession(EndModeCheckpoint) was requested, but not processed on the session.

StatusVoteEndWithResetPending
An endSession(EndModeReset) was requested, but not processed on the session.

EndMode
This enum is used with the endSession request to indicate the type of end-session processing that should be performed.

EndModeCheckPoint
Attempt checkpoint all sessionable-resources at the end of the session.

EndModeReset
Attempt to reset all sessionable-resources at the end of the session.

EndModeResetForce
Attempt to reset all sessionable-resources, roll-back all transactions, and force the session to end immediately.

FailedResource
This structure identifies a sessionable resource that could not be checkpointed or reset, the identity of the exception, and a minor code. If the resource threw the NotProcessed exception, then the minor code from that exception is used. If the resource threw a system exception then the minor from the system exception is used.

failingExceptionId
The identity of the exception that was thrown by the failing resource.

failingMinorCode
The minor code passed on the exception thrown by the failing resource.

failingResource
The sessionable resource that could not be checkpointed or reset.

FailedResourceList
This typedef lists one or more FailedResources that were encountered in a session checkpoint or reset request.

Interfaces

- ISessions::Control Interface
- ISessions::Coordinator Interface
- ISessions::Current Interface
- ISessions::Resource Interface
- ISessions::SessionableObject Interface

Control Interface

An ISessions::Control object represents a single session context.

File stem

ISessions

Intended usage

An ISessions::Control object represents a single session context. The Control object is created by a SessionFactory at the beginning of a session context to represent that context.

This interface allows a client to define, set, and get any number of session-specific properties in the current session context. These properties are global to the session, even as the session is propagated across the distributed system. However, they only apply as long as the session is active.

Exceptions

Unavailable
This exception is raised by methods on the ISessions::Control object when the requested Coordinator object is unavailable.

IDL syntax

```
interface Control
{
  Coordinator getSessionCoordinator()
    raises(Unavailable);
};
```

Supported operations

Control::getSessionCoordinator

Control::getSessionCoordinator

This interface returns an object that supports the ISessions::Coordinator interface for this session context.

Original interface

ISessions

IDL syntax

```
Coordinator getSessionCoordinator()
  raises(Unavailable);
```

Return value

Coordinator
The object that supports the ISessions::Coordinator interface for this session context.

Exceptions

Unavailable

Remarks

This interface returns an object that supports the ISessions::Coordinator interface for this session context. The Coordinator can be used to register resources with the session and perform other session management tasks.

Coordinator Interface

The Coordinator interface provides support for coordinating a session, including registering resources in the session.

File stem

ISessions

Intended usage

The Coordinator interface provides support for coordinating a session, including registering resources in the session. Other coordinators can be registered as a Resource of the Coordinator so that coordinating a session end can be delegated to sub-coordinators. Each instance of a Coordinator is implicitly associated with only one session context.

Exceptions

AlreadyRegistered
This exception is raised by the session coordinator when the same resource is registered more than once.

Inactive
This exception is raised by the session coordinator when an attempt to register or unregister a Resource is made when the session is not in a valid state.

NotRegistered
This exception is raised by a session coordinator if a resource being unregistered was not previously registered with the session.

IDL syntax

```
interface Coordinator
{
  Status getSessionStatus();
  boolean isSameSession(in Coordinator session_coordinator);
  unsigned long hashSession();
  void registerResource(in Resource resource,
                        in unsigned short priority)
    raises(Inactive, AlreadyRegistered);
  void unregisterResource(in Resource resource)
    raises(Inactive, NotRegistered);
  string getSessionName();
};
```

Supported operations

Coordinator::getSessionName
Coordinator::getSessionStatus
Coordinator::hashSession
Coordinator::isSameSession
Coordinator::registerResource
Coordinator::unregisterResource

Coordinator::getSessionName

This operation returns the name of the current session. Typically this is human-readable form of a unique identity value. The string-form of a UUID, that distinctly identifies this session.

Original interface

ISessions

IDL syntax

```
string getSessionName();
```

Coordinator::getSessionStatus

This operation returns to the caller the status of the session context.

Original interface

ISessions

IDL syntax

```
Status getSessionStatus();
```

Return value

Status The status of the current session. The potential values of *Status* are listed in Chapter 42. ISessions in the Session Service.

Coordinator::hashSession

This operation returns a hash value representing an approximate identity for the session associated with the target object.

Original interface

ISessions

IDL syntax

```
unsigned long hashSession();
```

Return value

unsigned long
The hash value for the Coordinator.

Remarks

This operation returns a hash value representing an approximate identity for the session associated with the target object. Each session is assigned a hash value that remains constant throughout the life of that session. While the hash value may not be unique across two sessions, it can be used as a first-order check for whether two sessions are the same. When comparing two hash-values, if they are different then the sessions from which they were obtained are necessarily different. If they are the same, then sessions may be different or the same, you can only be certain by invoking the isSameSession operation.

Coordinator::isSameSession

Returns a boolean indicating whether the session implied by the specified Coordinator is the same as the one associated with the Coordinator on which this method is invoked.

Original interface

ISessions

IDL syntax

```
boolean isSameSession(in Coordinator session_coordinator);
```

Parameters

session_coordinator
The session Coordinator with which the target object is to be compared.

Return value

TRUE If the session implied by the specified Coordinator is the same as the one associated with the Coordinator on which this method is invoked.

FALSE The session implied by the specified Coordinator is not the same as the one associated with the Coordinator on which this method is invoked.

Remarks

Returns a boolean indicating whether the session implied by the specified Coordinator is the same as the one associated with the Coordinator on which this method is invoked. This is a simple way that a resource can determine whether they are being invoked in a session they have already seen, and from that determine whether they need to register themselves as a resource with the Coordinator.

Coordinator::registerResource

Registers an ISessions::Resource object as a participant in the session. Having been registered in the session, the Resource object will be notified when the session checkpoints, resets or ends.

Original interface

ISessions

IDL syntax

```
void registerResource(in Resource resource,
                      in unsigned short priority)
  raises(Inactive, AlreadyRegistered);
```

Parameters

resource
The Resource object to be registered.

priority
The priority with which the Resource object is to be registered. The valid range for priorities are:

Range (in Hexadecimal)	Priority	Recommended Use
0x0000 - 0x0FFF	Top	Business object supplied resources
0x1000 - 0x3FFF	High	Managed objects

Range (in Hexadecimal)	Priority	Recommended Use
0x4000		
0x4001 - 0xBFFF	Medium	
0xC000		Connections
0xC001 - 0xCFFF		
0xD000		CICON cache
0xD001 - 0xFFF	Low	

Exceptions

AlreadyRegistered
Inactive

Remarks

Registers an ISessions::Resource object as a participant in the session. Having been registered in the session, the Resource object will be notified when the session checkpoints, resets or ends. When registering a resource, you must specify its relative priority, indicating the order in which that resource will be processed relative to other resources registered in the local process. Resources with a lower priority value are processed before resources with a higher priority value. Resources registered with the same priority are processed in the order in which they were registered.

This operation will raise the Inactive exception if the session is no longer active, that is, it was ended, perhaps by another process. This operation will raise the AlreadyRegistered exception if the Resource has already been registered in this session, a resource can only be registered once per session.

Coordinator::unregisterResource

Unregisters the specified Resource from the session.

Original interface

ISessions

IDL syntax

```
void unregisterResource(in Resource resource)
  raises(Inactive, NotRegistered);
```

Parameters

resource
The Resource object to be unregistered.

Exceptions

Inactive
NotRegistered

Remarks

Unregisters the specified Resource from the session. After the resource was unregistered it will no longer be notified of session events, that is, checkpoints, resets, or ends. This operation will raise the Inactive exception if the session is no longer active, that is, it was ended, perhaps by another process. This operation will raise the NotRegistered exception if the Resource is not currently registered with the session.

Current Interface

The Current interface defines methods that allow a client of the session service to explicitly manage the association between thread and sessions.

File stem

ISessions

Intended usage

The Current interface defines methods that allow a client of the session service to explicitly manage the association between thread and sessions. The Current interface provides indirect control of the session. It also defines methods that simplify the use of the session service. The methods can be used to begin and end sessions, and to obtain information about the current session.

AIX WIN On AIX and Windows NT, the Current interface is derived from a CORBA::Current and can be obtained by invoking CORBA::ORB::resolve_initial_references(″SessionCurrent″), and narrowing the resulting CORBA::Object to an ISessions::Current.

SOLARIS On Solaris, the Current interface is derived from a CORBA::Current and can be obtained by invoking CORBA::ORB::get_Current(″ISessions::Current″), and narrowing the resulting CORBA::Current to an ISessions::Current.

Exceptions

CheckpointPending
This exception is raised by the session current on a checkpointSession, resetSession, or endSession if a checkpoint request is already in progress.

DifferentEndModeForced
This exception is raised by the session current on an endSession request if the session has already been marked for reset.

EndSessionPending
This exception is raised by the session current on a checkpointSession, resetSession, or endSession if a end-session request is already in progress.

IncompleteProcess
This exception is raised by the session current on a checkpointSession or resetSession if any of the resources that were processed could not perform their checkpoint or reset operation and raised the NotProcessed exception.

InvalidControl
This exception is raised by the session current on a resumeSession when an invalid Control object is passed as a parameter.

NoSession
This exception is raised by methods that requires session context to operate when there is none in the request.

NotAtTopCoordinator
This exception is raised by session current on a checkpointSession, resetSession, or endSession if the requestor is not in the same process as the top-coordinator.

ResetPending
This exception is raised by the session current on a checkpointSession, resetSession, or endSession if a reset request is already in progress.

SessionAlreadyActive
This exception can be raised by the session current on a beginSession or resumeSession if a session is already active on the requestor's thread of execution.

SubThreadPending
This exception is raised by the session current on a endSession if an asynchronous thread in a sub-coordinator is still active, has not suspended the session on its thread or forced a reset. This exception can also be raised on a suspendSession operation if all other threads have ended their association with the session and at least one has requested that the session end. In this case the suspendSession operation is, in effect, an endSession operation and throws the same exceptions.

TimeoutTooLarge
Indicates that the timeout value indicated for the requested session is too large. The requested session is not begun when this exception is raised. The maximum timeout value accepted by the session service implementation is indicated in the exception.

TransactionPending
This exception is raised by the session current on a endSession if a transaction is still pending, has not been committed or rolled-back, in the top-level or any sub-level coordinators related to the session. This exception can also be raised on a beginSession or joinSession operation if a transaction is already associated with the current thread.

IDL syntax

```
interface Current
{
  boolean beginSession(in string profile_name)
    raises(TimeoutTooLarge, SessionAlreadyActive, TransactionPending);
  boolean joinSession(in string profile_name)
    raises(TimeoutTooLarge,
           TransactionPending);
  void checkpointSession()
    raises( NoSession,
      NotAtTopCoordinator,
      SubThreadPending,
      CheckpointPending,
      ResetPending,
      EndSessionPending,
      IncompleteProcess);
  void resetSession()
    raises(
      NoSession,
      NotAtTopCoordinator,
      SubThreadPending,
      CheckpointPending,
      ResetPending,
      EndSessionPending,
      IncompleteProcess);
  void endSession(in EndMode end_processing,
                  in boolean wait)
    raises(
      CheckpointPending,
      DifferentEndModeForced,
      EndSessionPending,
      IncompleteProcess,
      NoSession,
      NotAtTopCoordinator,
      ResetPending,
      SubThreadPending,
      TransactionPending);
  Status getSessionStatus();
  string getSessionName();
  void setSessionTimeout(in unsigned long seconds)
    raises(TimeoutTooLarge);
  Control getSessionControl()
    raises(NoSession);
  Control suspendSession()
    raises(
      DifferentEndModeForced,
      IncompleteProcess,
      NoSession,
      ResetPending,
      SubThreadPending,
      TransactionPending);
  void resumeSession(in Control which)
    raises(InvalidControl, SessionAlreadyActive, TransactionPending);
};
```

Supported operations

Current::beginSession
Current::checkpointSession
Current::endSession
Current::getSessionControl
Current::getSessionName
Current::getSessionStatus
Current::joinSession
Current::resetSession
Current::resumeSession
Current::setSessionTimeout
Current::suspendSession

Current::beginSession

This operation creates a new session.

Original interface

ISessions::Current Interface

IDL syntax

```
boolean beginSession(in string profile_name)
  raises(TimeoutTooLarge, SessionAlreadyActive, TransactionPending);
```

Parameters

profile_name
The name of the session profile under which the session should operate.

Return value

TRUE If the specified profile was found in the system management database.

FALSE If the specified profile was not found in the system management database.

Exceptions

SessionAlreadyActive
TimeoutTooLarge
TransactionPending

Remarks

This operation creates a new session. The session context of the client thread is modified so that the thread is associated with the new session. A session can not be started if a transaction is already active on the thread of execution. If one is detected, the TransactionPending exception is raised.

The session profile is currently unused and any string may be specified. As a result, this method will currently return FALSE. It is provided for future expansion of the sessions service and will allow an application to associate application-specific context with the thread. This context is implicitly propagated with requests to any objects that support the SessionableObject interface.

Current::checkpointSession

This operation checkpoints the current session.

Original interface

ISessions

IDL syntax

```
void checkpointSession()
  raises(
    NoSession,
    NotAtTopCoordinator,
    SubThreadPending,
    CheckpointPending,
    ResetPending,
    EndSessionPending,
    IncompleteProcess);
```

Exceptions

CheckpointPending
EndSessionPending
IncompleteProcess
NoSession
NotAtTopCoordinator
ResetPending
SubThreadPending

Current::endSession

Ends the current session.

Original interface

ISessions

IDL syntax

```
void endSession(in EndMode end_processing,
                in boolean wait)
  raises(
    NoSession,
```

```
    NotAtTopCoordinator,
    SubThreadPending,
    TransactionPending,
    CheckpointPending,
    ResetPending,
    EndSessionPending,
    DifferentEndModeForced,
    IncompleteProcess);
```

Parameters

end_processing
The type of end session operation to be performed. The potential *EndMode* values are listed in Chapter 42. ISessions in the Session Service.

wait If you specify TRUE for the wait flag, then your request will be blocked until the checkpoint or reset processing is complete. The resulting outcome may be raise a corresponding exception back to you. If you specify FALSE for the wait flag, then your request will return immediately, you will not be informed of the final outcome of the session.

Exceptions

CheckpointPending
DifferentEndModeForced
EndSessionPending
IncompleteProcess
NoSession
NotAtTopCoordinator
ResetPending
SubThreadPending
TransactionPending

Remarks

Ends the current session. You must specify the way in which the session is to be ended. If you specify EndModeCheckpoint, then the session will be checkpointed before ending. If you specify EndModeReset or EndModeResetForce, the session will be reset before ending.

You can only request this operation with the EndModeCheckpoint or EndModeReset options from a top-level session process.

If this request is issued from one thread, and other threads are active under the same session, and you specify EndModeCheckpoint or EndModeReset, then the session is merely marked for the end-mode specified. The actual checkpoint or reset processing is not performed until after the last active thread has either suspended or ended the session on their thread. If you specify TRUE for the wait flag, then your request will be blocked until the checkpoint or reset processing is complete. The resulting outcome may be raise a corresponding exception back to you. On the other hand, if you specify FALSE for the wait flag, then your request will return immediately, you will not be informed of the final outcome of the session.

If this request is issued on the only remaining thread active under the session, then the end-mode processing is initiated immediately and the request will block until that processing has completed, irrespective of how the wait flag is set.

If any thread in the top-level session process invoked an endSession(EndModeReset), or any thread in a subordinate process invoked an endSession(EndModeResetForce) in the course of this session, then the session will be ended with a reset, irrespective of what end-mode you specify on this request.

If you specify EndModeResetForce, then resources are reset, transactions are marked for roll-back, and the session is ended immediately, even if other threads are actively involved in the session.

The session context is removed from the thread of execution after all resources are invoked with end Resource. Resource objects may therefore use the Current interface to determine which session is being ended.

Current::getSessionControl

This operation returns the ISessions::Control object representing the session context currently associated with the thread of execution.

Original interface

ISessions

IDL syntax

```
Control getSessionControl()
  raises(NoSession);
```

Return value

Control

The Control object representing the session context currently associated with the thread of execution.

Exceptions

NoSession

Remarks

This operation returns the ISessions::Control object representing the session context currently associated with the thread of execution. This can be used as input to the resumeSession operation to propagate the session context to another thread of execution.

This operation does not return a session control clone but instead returns a reference to the base session control. That is, even if a transaction is outstanding on the thread of execution when this request is invoked, the transaction context is not associated with the session control returned by this operation. Consequently, if you use this control to resume the session context in another thread of execution, it will not include the transaction context active on the original thread of execution.

Current::getSessionName

This operation returns the name of the current session. If there is no session associated with the current thread, an empty string is returned.

Original interface

ISessions

IDL syntax

```
string getSessionName();
```

Return value

string String representation of the name of the current session.

Current::getSessionStatus

This operation returns the status of the current session. Status is determined by private collaboration between the Current and the session coordinator.

Original interface

ISessions

IDL syntax

```
Status getSessionStatus();
```

Return value

Status The status of the current session. The potential values of *Status* are listed in Chapter 42. ISessions in the Session Service.

Current::joinSession

This operation can be used to conditionally begin a new session.

Original interface

ISessions

IDL syntax

```
boolean joinSession(in string profile_name)
  raises(
    TimeoutTooLarge,
    TransactionPending);
```

Parameters

profile_name
The name of the session profile under which the session should operate.

Return value

TRUE If no session is already active on the thread of execution, then a new one is begun, and TRUE is returned.

FALSE If there is already a session active, no new session started, and FALSE is returned.

Exceptions

TimeoutTooLarge
TransactionPending

Remarks

The session profile is currently unused and any string may be specified. As a result, this method will currently return FALSE. It is provided for future expansion of the sessions service and will allow an application to associate application-specific context with the thread. This context is implicitly propagated with requests to any objects that support the SessionableObject interface.

Current::resetSession

This operation restores the current session back to its last saved state.

Original interface

ISessions

IDL syntax

```
void resetSession()
  raises(
    NoSession,
    NotAtTopCoordinator,
    SubThreadPending,
```

```
    CheckpointPending,
    ResetPending,
    EndSessionPending,
    IncompleteProcess);
```

Exceptions

CheckpointPending
EndSessionPending
IncompleteProcess
NoSession
NotAtTopCoordinator
ResetPending
SubThreadPending

Current::resumeSession

This operation attaches the session context, represented by the specified ISessions::Control object, to the current thread of execution and resumes your participation in the session.

Original interface

ISessions

IDL syntax

```
void resumeSession(in Control which)
  raises(
    InvalidControl,
    SessionAlreadyActive,
    TransactionPending);
```

Parameters

which The Control object representing the session context to be resumed.

Exceptions

InvalidControl
SessionAlreadyActive
TransactionPending

Remarks

This operation attaches the session context, represented by the specified ISessions::Control object, to the current thread of execution and resumes your participation in the session. If a session already exists on the current thread of execution, then the SessionAlreadyActive exception is raised.

This exception may indicate a programming error in your application. The normal response to this exception is to suspend the active session (as a separate

ISessions::Control object) and re-issue the resumeSession request with the session Control that you actually want to operate under.

If a transaction was active when the session was originally suspended, it will have been recorded in the ISessions::Control being resumed, and the original transaction will be resumed as well. If a transaction is already active when the resume is invoked, then a TransactionPending exception will be raised and your involvement in the session will not resume.

This exception may indicate a programming error in your application, however the normal response to this exception is to suspend the active transaction (as a separate CosTransactions::Control object) and re-issue the resumeSession request.

If a null Control object is specified, the client thread becomes associated with no session. If the Control passed in is invalid, an InvalidControl exception is raised. In particular, if the session was ended while your interest in the session was suspended, then the InvalidControl exception will be raised, and the Control no longer represents a valid session context.

Current::setSessionTimeout

This operation specifies the maximum duration of the session in seconds.

Original interface

ISessions

IDL syntax

```
void setSessionTimeout(in unsigned long seconds)
  raises(TimeoutTooLarge);
```

Parameters

seconds

Specifies the maximum duration of the session in seconds. The session timeout is set to zero by default.

Exceptions

TimeoutTooLarge

Remarks

This operation specifies the maximum duration of the session in seconds. If this timeout is exceeded, the session is automatically ended. This is useful for preventing the session from being leaked in the case that the client fails before ending the session. If the session timeout is set to zero (0), the session will not timeout.

The maximum session timeout value you can specify may be limited by the session service implementation, possibly based on a policy value set with the service. This is useful for preventing clients from accidentally or deliberately specifying absurdly large time-out values in a particular installation. The TimeoutTooLarge exception can be raised on the setSessionTimeout method if the time-out exceeds the maximum acceptable limit for the service implementation. However, in most implementations, this exception will not be raised on the setSessionTimeout method, but rather on the beginSession method.

The session timeout is set to zero by default.

Current::suspendSession

Returns an object that represent the session context currently associated with the client thread and suspends the session on that thread.

Original interface

ISessions

IDL syntax

```
Control suspendSession()
  raises(
    NoSession,
    SubThreadPending,
    TransactionPending,
    ResetPending,
    DifferentEndModeForced,
    IncompleteProcess);
```

Return value

Control

The Control object that represents the session context associated with the client thread.

Exceptions

DifferentEndModeForced
IncompleteProcess
NoSession
ResetPending
SubThreadPending
TransactionPending

Remarks

Returns an object that represent the session context currently associated with the client thread and suspends the session on that thread. The client thread is no longer associated with a session context after the operation is invoked. This can be used to

change to a different session, (beginSession or resumeSession can be invoked after this operation completes), or this can be used by an asynchronous thread to demark its involvement in the session. If an asynchronous thread is involved in a session, it must issue a suspendSession at the completion of its activity, otherwise the session will be forced to reset.

If a transaction context is active on the thread from which this request is issued, then the transaction context is suspended along with the session. The transaction context is then embedded in the returned Control object. Since the combination of session and transaction may be unique to the requesting thread, then the resulting Control returned from this request may be unique as well. Using this Control object in a resumeSession request in place of a Control object returned for another thread, within a different transaction context, or from the getSessionControl operation may yield different results.

In a top-level process, if another thread has already issued an endSession, and all other threads have completed their involvement in the session, then the end-mode processing will occur when you invoke this operation, you effectively represent the last controlling participant in the session. Any problems that are encountered during that end-mode processing will raise exceptions on this operation, including TransactionPending if an asynchronous thread in a sub-level process is still involved in a transaction, ResetPending if an endSession(EndModeResetForce) is issued by an asynchronous thread in a sub-level process at the same time, or DifferentEndModeForced if another thread had voted to reset the session.

The normal response to a TransactionPending exception is to wait for the pending transaction to complete, and then re-issue the suspendSession request. Depending on how your application is organized, you may have to coerce the transaction to end, either through normal collaboration or through an abnormal rollback request (assuming you can get access to the offending transaction context). You can force the transaction and the session to terminate abnormally (with a transaction rollback) by issuing an endSession(EndModeResetForce) in place of this suspendSession request. Otherwise, you may have to wait for the transaction to take its normal course, or timeout. Depending on how long the transaction takes to terminate, this exception may be raised repeatedly on subsequent requests until the transaction is cleared.

The normal response to a ResetPending exception is to wait for the pending reset to complete, and then re-issue the suspendSession request. This condition should only ever occur in the case that a sub-level process had issued an endSession(EndModeResetForce) request at the same time that you issued the suspendSession request. As a result, the reset will occur and the session will necessarily terminate as soon as you re-issue the suspendSession request. Depending on how long the reset takes to complete, this exception may be raised repeatedly on subsequent requests until the reset is complete.

The normal response to a DifferentEndModeForced exception depends on the relevance of this difference to your application. Generally, you do not have to do anything else for the session, your involvement will be terminated anyway. However, if the session was reset instead of being checkpointed as you expected, then you may have to notify your end user or take other action to compensate for the potential loss of data that could have resulted from the reset.

End-mode processing is only performed when another thread in the top-level process has issued the endSession request for the session. If no other thread has issued the endSession request, then the session is simply suspended without terminating it.

The session context is removed from the thread of execution after all resources are invoked with endResource. Resource objects may therefore use the Current interface to determine which session is being ended.

Resource Interface

The Resource interface defines a resource that needs to know about the completion of the sessions in which they are participating.

File stem

ISessions

Intended usage

The Resource interface defines a resource that needs to know about the completion of the sessions in which they are participating. Resource objects are registered with the ISessions::Coordinator for the current session, they should only be registered once per session they are participating in. The Resource will then be notified when the session checkpoints, resets, or ends.

Exceptions

NoSession
This exception is raised by methods that require session context to operate when there is none in the request.

NotProcessed
This exception can be raised by a resource if it can not perform a requested checkpoint or reset.

IDL syntax

```
interface Resource
{
  void checkpointResource()
    raises(NoSession, NotProcessed);
  void resetResource()
    raises(NoSession, NotProcessed);
  void endResource(in EndMode end_processing)
    raises(NoSession, NotProcessed);
  boolean isSameResource(in Resource session_resource);
  unsigned long hashResource();
};
```

Supported operations

Resource::checkpointResource
Resource::endResource
Resource::hashResource
Resource::isSameResource
Resource::resetResource

Resource::checkpointResource

This method instructs the Resource to checkpoint its internal state to its persistent store.

Original interface

ISessions

IDL syntax

```
void checkpointResource()
  raises(NoSession, NotProcessed);
```

Exceptions

NoSession
NotProcessed

Remarks

This method instructs the Resource to checkpoint its internal state to its persistent store. If the state could not be checkpointed, this method should be implemented to raise a NotProcessed exception along with an appropriate minor code identifying more precisely what prevented the resource from being checkpointed. The minor code is always specific to the resource and should be documented for use by client programs.

Resource::endResource

This operation is invoked to inform the resource that the session has ended.

Original interface

ISessions

IDL syntax

```
void endResource(in EndMode end_processing)
  raises(NoSession, NotProcessed);
```

Parameters

end_processing
The type of end session operation to be performed. The potential *EndMode* values are listed in Chapter 42. ISessions in the Session Service.

Exceptions

NoSession
NotProcessed

Remarks

This operation is invoked to inform the resource that the session has ended. Depending on the implications of a session-context to this Resource, it may perform any number of actions in response to this method, including pushing down any cached updates, removing itself from memory, or merely recording its lack of further involvement in this session. However, it should perform this action in consideration for the EndMode argument passed in on the request.

If the EndMode is set to EndModeCheckPoint, then the Resource should save any state changes that may have occurred since the last checkpointResource request to its persistent store, if any. This can be accomplished succinctly by invoking the checkpointResource operation on the object itself.

If the EndMode is set to EndModeReset then the Resource can be implemented in a way that is appropriate to it. In some cases, this may require actually transitioning state from the persistent data-system to pick up any relevant side-effects that occur in that transition, even though the state transitioned into memory will be freed within the ensuing endResource operation. This can be accomplished succinctly by invoking the resetResource operation on the object itself. In other cases, it may not be necessary to do anything, letting the current state of the resource object to essentially be lost in the ensuing endSession operation.

After acting on the end-mode processing, the resource object should free itself from memory (or arrange to be freed using its application adapter). However, some care may be appropriate here. If you have designed in any inter-dependencies between objects in your object implementation, freeing one object before other objects that are dependent on it have been freed may be problematic. It may be important to free objects in a particular order to avoid dangling local references to objects. The session services provides some assistance in this area by allowing resources to be registered with a specific priority. This priority can govern the order in which resources are operated on during session events, including the endResource event, to assist in solving the inter-dependency problem.

In addition, if your Resource can be participating in multiple session contexts concurrently, then freeing the resource at the first endResource request will almost certainly be problematic. It may be necessary for you to implement a use-counting mechanism in your resource to ensure it is not freed prematurely.

As with other method requests, the session context that is being ended is available on the thread of execution during this request. Thus, if the Resource is involved in multiple concurrent sessions, it can deduce which session is being ended by obtaining the session context from the thread of execution (using an ISessions::Current) on which this method is requested.

Resource::hashResource

Returns a hash value representing an approximate identity for the Resource.

Original interface

ISessions

IDL syntax

```
unsigned long hashResource();
```

Remarks

Returns a hash value representing an approximate identity for the Resource. Each resource should be implemented to create a hash value that remains constant throughout the life of that resource. While the hash value may not be unique across two resources, it can be used as a first-order check for whether two resources are the same. When comparing two hash-values, if they are different then the resources from which they were obtained are necessarily different. If they are the same, then resources may be different or the same, you can only be certain by invoking the isSameResource operation. This hash value can be used to increase the efficiency with which resource collections operate.

Resource::isSameResource

Returns a boolean indicating whether two references to a Resource are in fact to the same or different resource instances.

Original interface

ISessions

IDL syntax

```
boolean isSameResource(in Resource session_resource);
```

Parameters

session_resource
The Resource object with which the target object is being compared.

Return value

TRUE The objects are the same.

FALSE The objects are not the same.

Remarks

Returns a boolean indicating whether two references to a Resource are in fact to the same or different resource instances. This is a simple way that the session coordinator can detect whether the same resource has already been registered with the resource, or to determine which resource to unregister in either of those corresponding requests.

Resource::resetResource

This operation instructs the Resource to discard its current state and, if possible, restore back to the state that it last saved to its persistent store.

Original interface

ISessions

IDL syntax

```
void resetResource()
  raises(NoSession, NotProcessed);
```

Exceptions

NoSession
NotProcessed

Remarks

This operation instructs the Resource to discard its current state and, if possible, restore back to the state that it last saved to its persistent store. If the Resource can not revert back to its saved state, it should raise the NotProcessed exception along with an appropriate minor code identifying more precisely what prevented the resource from being checkpointed. The minor code is always specific to the resource and should be documented for use by client programs.

SessionableObject Interface

The SessionableObject interface is used by an object to indicate that it is sessionable.

File stem

ISessions

Intended usage

The SessionableObject interface is used by an object to indicate that it is sessionable. By inheriting from the SessionableObject interface, an object indicates that it wants the session context associated with the client thread to be propagated on requests to the object. If an object does not support the SessionableObject interface, the ORB is not required to propagate the transaction context on requests to the object, although the session context is always propagated in Component Broker, irrespective of whether the target object is a SessionableObject.

IDL syntax

```
interface SessionableObject {};
```

Chapter 43. NamingStringSyntax in the Naming Service

The other modules in the Naming Service are:
- CosNaming
- IExtendedNaming
- IExtendedNamingStringSyntax

NamingStringSyntax Module

Used in conjunction with the IExtendedNaming::NamingContext Interface to simplify the use of a name when invoking methods on a naming context. This module provides a more simple and familiar name format than the CORBA::Name structure.

File name

NamingStringSyntax.idl

Intended usage

The NamingStringSyntax module consists of two interfaces: the StandardSyntaxModule Interface and the StringName Interface. The StringName interface defines operations for converting a string name to a CosNaming name and vice-versa. The StandardSyntaxModule class defines rules for parsing the string name.

Types

```
typedef CosNaming::Istring NSS_Istring;
typedef NSS_Istring NameString;
typdef sequence<NSS_Istring> NSS_IstringList;
```

Interfaces

NamingStringSyntax::StandardSyntaxModel Interface
NamingStringSyntax::StringName Interface

StandardSyntaxModel Interface

Defines a set of rule attributes that are used to control parsing according to a set of defined rules. In the Component Broker implementation, these attributes cannot be modified. They can only be queried to determine the values being used.

File name

NamingStringSyntax.idl

Intended usage

Allows the user to query the object to determine the values being used to control the parsing behavior.

Ancestor interfaces

NamingStringSyntax::StringName Interface

Types

```
enum Direction { kLeftToRight, kRightToLeft };
enum CodeSet { kISOLatin1 };
enum Locale { kUS_ENG };
```

Attributes

StandardSyntaxModel::syntax_absolute_prefix
StandardSyntaxModel::syntax_begin_quote
StandardSyntaxModel::syntax_code_set
StandardSyntaxModel::syntax_delimiter
StandardSyntaxModel::syntax_direction
StandardSyntaxModel::syntax_end_quote
StandardSyntaxModel::syntax_escape
StandardSyntaxModel::syntax_locale_info
StandardSyntaxModel::syntax_reserved_names
StandardSyntaxModel::syntax_separator

StandardSyntaxModel::syntax_absolute_prefix

The string(s) that define absolute prefixes. Absolute prefixes are used to denote an absolute name that is applied to a local root context. The default value of this attribute is: ″/″, ″\″.

Original interface

NamingStringSyntax::StandardSyntaxModel Interface

IDL syntax

```
attribute sequence<NSS_Istring> syntax_absolute_prefix;
```

Remarks

The value for this attribute can be obtained using the getter method. Any value passed to the setter method will be ignored.

StandardSyntaxModel::syntax_begin_quote

The string(s) that define the beginning of a quoted string. If more than one character is specified then either can be used interchangeably, but each must be matched with a corresponding syntax_end_quote character (as determined by its index position within the character set). The default value of this attribute is: ″″″ (double-quote), ″’″ (single-quote).

Original interface

NamingStringSyntax::StandardSyntaxModel Interface

IDL syntax

```
attribute sequence<NSS_Istring> syntax_begin_quote;
```

Remarks

The value for this attribute can be obtained using the getter method. Any value passed to the setter method will be ignored.

StandardSyntaxModel::syntax_code_set

Defines the code set used for parsing the string. The default value of this attribute is: kISOLatin1.

Original interface

NamingStringSyntax::StandardSyntaxModel Interface

IDL syntax

```
enum CodeSet { kISOLatin1 };
attribute CodeSet syntax_code_set;
```

Remarks

The value for this attribute can be obtained using the getter method. Any value passed to the setter method will be ignored.

StandardSyntaxModel::syntax_delimiter

The string(s) that are used to separate name-components. If more than one character is specified then either can be used interchangeably. The default value of this attribute is: ″/″, ″\″.

Original interface

NamingStringSyntax::StandardSyntaxModel Interface

IDL syntax

```
attribute sequence<NSS_Istring> syntax_delimiter;
```

Remarks

The value for this attribute can be obtained using the getter method. Any value passed to the setter method will be ignored.

StandardSyntaxModel::syntax_direction

Indicates the direction in which the parsed string is loaded into components of the CosNaming::Name. The direction could be either from left to right (denoted by kLeftToRight) or from right to left (denoted by kRightToleft). The default value of the attribute is: kLeftToRight.

Original interface

NamingStringSyntax::StandardSyntaxModel Interface

IDL syntax

```
enum Direction { kLeftToRight, kRightToLeft }
attribute Direction syntax_direction;
```

Remarks

The value for this attribute can be obtained using the getter method. Any value passed to the setter method will be ignored.

StandardSyntaxModel::syntax_end_quote

The string(s) that define the end of a quoted string. The number of characters specified must match the number of characters specified in syntax_begin_quote. If more than one character is specified, then either can be used interchangeably, but must match the corresponding syntax_begin_quote character used. The default value of this attribute is: ″″″ (double-quote), ″″ (single-quote).

Original interface

NamingStringSyntax::StandardSyntaxModel Interface

IDL syntax

```
attribute sequence<NSS_Istring> syntax_end_quote;
```

Remarks

The value for this attribute can be obtained using the getter method. Any value passed to the setter method will be ignored.

StandardSyntaxModel::syntax_escape

The character that is used to escape the parsing rules. Only one character can be specified. The default value of this attribute is: '\'.

Original interface

NamingStringSyntax::StandardSyntaxModel Interface

IDL syntax

```
attribute NSS_Istring syntax_escape;
```

Remarks

The value for this attribute can be obtained using the getter method. Any value passed to the setter method will be ignored.

StandardSyntaxModel::syntax_locale_info

Defines the locale used for parsing the string. The default value of this attribute is: kUS_ENG.

Original interface

NamingStringSyntax::StandardSyntaxModel Interface

IDL syntax

```
enum Locale { kUS_ENG }
attribute Locale syntax_locale_info
```

Remarks

The value for this attribute can be obtained using the getter method. Any value passed to the setter method will be ignored.

StandardSyntaxModel::syntax_reserved_names

The string(s) that define reserved-names. Reserved-names are not parsed, even if they contain separator characters. The default value of this attribute is: ″.:″, ″...″.

Original interface

NamingStringSyntax::StandardSyntaxModel Interface

IDL syntax

```
attribute sequence<NSS_Istring> syntax_reserved_names;
```

Remarks

The value for this attribute can be obtained using the getter method. Any value passed to the setter method will be ignored.

StandardSyntaxModel::syntax_separator

The string(s) that are used to separate the id and kind fields of a name-component. If more than one character is specified, either can be used interchangeably. The default value of this attribute is: ″.″.

Original interface

NamingStringSyntax::StandardSyntaxModel Interface

IDL syntax

```
attribute sequence<NSS_Istring> syntax_separator;
```

Remarks

The value for this attribute can be obtained using the getter method. Any value passed to the setter method will be ignored.

StringName Interface

The StringName interface supports operations for converting a string-name to a CosNaming-name or vice versa.

File name

NamingStringSyntax.idl

Types

```
typedef CosNaming::Istring Istring
typedef Istring NameString
```

Exceptions

CORBA standard exceptions and the following user exceptions:

NamingStringSyntax::IllegalStringSyntax
Indicates that the string does not follow the syntax rule.

NamingStringSyntax::UnMatchedQuote
Indicates that a begin-quote is not matched by an end-quote or vice versa.

Supported operations

StringName::name_to_string
StringName::string_to_name

StringName::name_to_string

Converts a CosNaming::Name name into a NameString name.

Original interface

NamingStringSyntax::StringName Interface

IDL syntax

```
NameString name_to_string(in CosNaming::Name name)
```

Parameters

name The CosNaming::Name name to be converted.

Return value

<NameString>
The name of the NameString.

Exceptions

CORBA standard exceptions.

Remarks

This operation is used by the operations defined in the interfaces of the Chapter 25. IExtendedNaming in the Naming Service. It converts a CosNaming::Name name into a NameString name.

StringName::string_to_name

Converts a NameString name into a CosNaming::Name name. Syntax rules of the NameString are described in the “Naming Service” chapter of the *Component Broker Advanced Programming Guide*.

Original interface

NamingStringSyntax::StringName

IDL syntax

```
CosNaming::Name string_to_name(in NameString name)
```

Parameters

name The NameString name to be converted.

Return value

CosNaming::Name
The name of the CosNaming::Name name.

Exceptions

CORBA standard exceptions and the following user exceptions:

NamingStringSyntax::IllegalStringSyntax
Indicates that the string does not follow the syntax rule.

NamingStringSyntax::UnMatchedQuote
Indicates that a begin-quote is not matched by an end-quote or vice versa.

Remarks

This operation is used by the operations defined in the interfaces of the IExtendedNaming module.

Chapter 44. Security in the Security Service

The other modules in the Security Service are:
- IExtendedSecurity
- IExtendedSecurityClient
- SecurityLevel1
- SecurityLevel2

Security Module

The Security module contains data types and definitions used in the CORBA Security interface definitions and elsewhere. It has been necessary to extend these data types and definitions in some cases for the Component Broker. These extensions are indicated by comments in the module.

File name

Security.idl

Types

```
typedef string              SecurityName;

   typedef sequence<octet>  Opaque;

   // Extensible families for standard data types

   struct ExtensibleFamily {
       unsigned short          family_definer;
       unsigned short          family;
       };

   // Security association mechanism type

   typedef string MechanismType;
   struct SecurityMechandName {
       MechanismType           mech_type;
       SecurityName            security_name;
       };

   typedef sequence<MechanismType>         MechanismTypeList;
   typedef sequence<SecurityMechandName>   SecurityMecandNameList;

   // Security attributes

   typedef unsigned long       SecurityAttributeType;

   struct AttributeType {
       ExtensibleFamily        attribute_family;
       SecurityAttributeType   attribute_type;
       };
```

```
typedef sequence<Security::AttributeType>  AttributeTypeList;

struct Attribute {
    AttributeType           attribute_type;
    Opaque                  defining_authority;
    Opaque                  value;  // This can be interpreted only with
                                    // knowledge of type.
    };

typedef sequence<Security::Attribute>  AttributeList;

// Authentication return status

enum AuthenticationStatus {
    SecAuthSuccess,
    SecAuthFailure,
    SecAuthContinue,
    SecAuthExpired
    };

// Association return status

enum AssociationStatus {
    SecAssocSuccess,
    SecAssocFailure,
    SecAssocContinue
    };

// Authentication method

typedef unsigned long AuthenticationMethod;

// Credential types

enum CredentialType {
    SecInvocationCredentials,
    SecOwnCredentials,
    SecNRCredentials
    };

// Declarations related to rights.

struct Right {
    ExtensibleFamily        rights_family;
    string                  right;
    };

typedef sequence<Right>  RightsList;

enum RightsCombinator {
    SecAllRights,
    SecAnyRight
    };
```

```
// Delegation related

enum DelegationState {
    SecInitiator,
    SecDelegate
    };

// Time

//typedef TimeBase::UtcT UtcT;
//typedef TimeBase::IntervalT IntervalT;
//typedef TimeBase::TimeT TimeT;

typedef unsigned long       UtcT;
typedef unsigned long       IntervalT;
typedef unsigned long       TimeT;

// Security features available on credentials

enum SecurityFeature {
    SecNoDelegation,
    SecSimpleDelegation,
    SecCompositeDelegation,
    SecNoProtection,
    SecIntegrity,
    SecConfidentiality,
    SecIntegrityAndConfidentiality,
    SecDetectReplay,
    SecDetectMisordering,
    SecEstablishTrustInTarget
    };

struct SecurityFeatureValue {
    SecurityFeature         feature;
    boolean                 value;
    };

typedef sequence<SecurityFeatureValue>  SecurityFeatureValueList;

// Quality of protection

enum QOP {
    SecQOPNoProtection,
    SecQOPIntegrity,
    SecQOPConfidentiality,
    SecQOPIntegrityAndConfidentiality
    };

// Secure association options

typedef unsigned short AssociationOptions;

// Specifies whether association option is required or supported

enum RequiresSupports {
```

```
    SecRequires,
    SecSupports
};

// Communication direction

enum CommunicationDirection {
    SecDirectionBoth,
    SecDirectionRequest,
    SecDirectionReply
    };

// AssociationOptions-Direction pair

struct OptionsDirectionPair {
    AssociationOptions      options;
    CommunicationDirection  direction;
};

typedef sequence<OptionsDirectionPair>  OptionsDirectionPairList;

// Delegation mode

enum DelegationMode {
    SecDelModeNoDelegation,         // Use own credentials
    SecDelModeSimpleDelegation,     // delegate received credentials
    SecDelModeCompositeDelegation   // delegate both
    };

// Association options supported by a given mechanism type

struct MechandOptions {
    MechanismType           mechanism_type;
    AssociationOptions      options_supported;
    };

typedef sequence<MechandOptions>  MechandOptionsList;

// Audit

struct AuditEventType {
    ExtensibleFamily        event_family;
    unsigned short          event_type;
    };

typedef sequence<AuditEventType>  AuditEventTypeList;

typedef unsigned long SelectorType;

struct SelectorValue {
    SelectorType            selector;
    any                     value;
    };

typedef sequence<SelectorValue>  SelectorValueList;
```

Constants

```
// Identity attributes: family definer = 0, family = 0

   const SecurityAttributeType AuditId          = 1;
   const SecurityAttributeType AccountingId     = 2;
   const SecurityAttributeType NonRepudiationId = 3;

   // Privilege attributes: family definer = 0, family = 1;

   const SecurityAttributeType Public         = 1;
   const SecurityAttributeType AccessId       = 2;
   const SecurityAttributeType PrimaryGroupId = 3;
   const SecurityAttributeType GroupId        = 4;
   const SecurityAttributeType Role           = 5;
   const SecurityAttributeType AttributeSet   = 6;
   const SecurityAttributeType Clearance      = 7;
   const SecurityAttributeType Capability     = 8;

   const AssociationOptions NoProtection           = 1;
   const AssociationOptions Integrity              = 2;
   const AssociationOptions Confidentiality        = 4;
   const AssociationOptions DetectReplay           = 8;
   const AssociationOptions DetectMisordering      = 16;
   const AssociationOptions EstablishTrustInTarget = 32;
   const AssociationOptions EstablishTrustInClient = 64;

   const SelectorType InterfaceRef   = 1;
   const SelectorType ObjectRef      = 2;
   const SelectorType Operation      = 3;
   const SelectorType Initiator      = 4;
   const SelectorType SuccessFailure = 5;
   const SelectorType Time           = 6;
```

Exceptions

```
   // The CORBA Security standard does NOT define exceptions.

   enum InvalidEnumTypeReason {
       OutOfRange,
       NotSupported
       };

   exception InvalidCredentialType {
       InvalidEnumTypeReason    reason;
       Security::CredentialType type;
       };

   exception InvalidCommDirection {
       InvalidEnumTypeReason   reason;
       CommunicationDirection  direction;
       };

   enum InvalidAttributeTypeReason {
       InvalidType,
       InvalidFamily,
```

```
    InvalidFamilyDefiner
    };

exception InvalidAttributeType {
    InvalidAttributeTypeReason  reason;
    AttributeType               type;
    };

exception DuplicateAttributeType {
    AttributeType           type;
    };

exception InvalidSecurityFeature {
    InvalidEnumTypeReason   reason;
    SecurityFeature         feature;
    };

exception DuplicateSecurityFeature {
    SecurityFeature         feature;
    };

exception InvalidAuthnMethod {
    InvalidEnumTypeReason reason;
    AuthenticationMethod authn_method;
    };

exception InvalidTargetName {
    Security::SecurityName  name;
    };

exception InvalidAssociationOption {
    InvalidEnumTypeReason   reason;
    AssociationOptions       options;
    };

exception DuplicateAssociationOption {
    AssociationOptions       options;
    };

exception InvalidDelegationMode {
    InvalidEnumTypeReason   reason;
    DelegationMode          mode;
    };

exception InvalidQOP {
    InvalidEnumTypeReason   reason;
    QOP                     qop;
    };

exception InvalidMechType {
    MechanismType           type;
    };
```

Chapter 45. SecurityLevel1 in the Security Service

The other modules in the Security Service are:
- IExtendedSecurity
- IExtendedSecurityClient
- Security
- SecurityLevel2

SecurityLevel1 Module

Contains the Current interface for CORBA Security Level1 conformance.

File name

SecurityLevel1.idl

Interfaces

SecurityLevel1::Current Interface

Current Interface

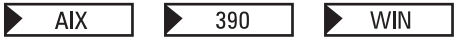
This class contains the implementation of the CORBA SecurityLevel1 Current. This class provides access to security level 1 function as defined in the Object Management Group (OMG) specification.

File name

SecurityLevel1.idl

Intended usage

The Current object in its most general form represents any context information attached to a thread of execution. This can include security credentials, transaction context, or context information attached by other services and facilities (such as the Component Broker server adapter).



For AIX, OS/390, and Windows NT:

Invoking the CORBA::ORB::resolve_initial_references request will return a CORBA::Object. The returned CORBA::Object can be specialized by each service that maintains context information on the thread. The application can narrow the CORBA::Object to the specific service interface of interest for which there is context information attached to the thread.

In particular, the security service introduces a specialization of the Current that will be returned from CORBA::ORB::resolve_initial_references when, for example, security credentials are attached to the thread. The SecurityLevel1::Current specialization can be used to retrieve attributes like CredAttrSecName.

SOLARIS For Solaris:

Invoking the CORBA::ORB::get_current request will return a Current object. The returned CORBA::Current can be specialized by each service that maintains context information on the thread. The application can narrow the Current object to the specific service interface of interest for which there is context information attached to the thread.

In particular, the security service introduces a specialization of the Current that will be returned from CORBA::ORB::get_current when, for example, security credentials are attached to the thread. The SecurityLevel1::Current specialization can be used to retrieve attributes like CredAttrSecName.

Ancestor classes

CORBA::Current Class

Supported methods

Current::get_attributes

Current::get_attributes

Returns the privilege and other attributes from a client's credentials. In a client, the client's privileges are returned. In a server, the received privileges of the client are returned; the server's privileges are returned if the server is not communicating with a client.

Original interface

SecurityLevel1::Current Class

IDL syntax

```
Security::AttributeList* get_attributes(
  in Security::AttributeTypeList attributes
)
   raises (Security::InvalidAttributeType,
           Security::DuplicateAttributeType);
```

Parameters

attributes

The attributes whose values are to be returned. These may be credential or privilege attributes. The defining authority and family values of these attributes are defined in the general security data module. Credential attributes identify

credential information such as security name and host name. Privilege attributes identify privileges of a credential and the privileges for which access is authorized in an access control list.

Return value

Security::AttributeList
In a client process, the client's attributes are returned. In a server process, the attributes of the client with whom the server is communicating is returned; if the server is not communicating with a client, the server's own attributes are returned.

Exceptions

Security::InvalidAttributeType
Security::DuplicateAttributeType

Remarks

Two attributes are supported on Component Broker for workstations. These are CredAttrSecName and CredAttrHostName, which refer to the DCE username and the host machine's name for the currently active Principal. To access these attributes, the family_definer field needs to be 8 and the family field needs to be 2.

On OS/390 Component Broker, the two supported attributes are AccessId and GroupId. AccessId maps to an MVS user identity that may be used with the SAF-based authorization services such as RACF. GroupId maps to a group identity that may be used with SAF-based authorization services. To access these attributes, the family_definer field needs to be set to 0 and the family field needs to be 1.

Example

AIX 390 WIN For AIX, OS/390, and Windows NT:

```
CORBA::ORB_ptr pTheORB;
SecurityLevel1::Current_ptr pSecCurrent;
Security::AttributeTypeList attribute_types(1);
Security::AttributeList_ptr pAttributes;

attribute_types[0].attribute_family.family_definer = 8; // = 0 for OS/390
attribute_types[0].attribute_family.family = 2; // = 1 for OS/390
attribute_types[0].attribute_type =
   IExtendedSecurity::CredAttrSecName; // = Security::CredAttrSecName for OS/390

pTheORB = CORBA::ORB_init(argc, argv, "DSOM");
pSecCurrent = SecurityLevel1::Current::_narrow (
   pTheORB->resolve_initial_references("SecurityCurrent") );
pAttributes = pSecCurrent->get_attributes(attribute_types);
```

SOLARIS For Solaris:

```
CORBA::ORB_ptr pTheORB;
SecurityLevel1::Current_ptr pSecCurrent;
Security::AttributeTypeList attribute_types(1);
Security::AttributeList_ptr pAttributes;

attribute_types[0].attribute_family.family_definer = 8;
attribute_types[0].attribute_family.family = 2;
attribute_types[0].attribute_type = Security::CredAttrSecName;

pTheORB = CORBA::ORB_init(argc, argv, "DSOM");
pSecCurrent = pTheORB->get_current("SecurityLevel2::Current") );
pAttributes = pSecCurrent->get_attributes(attribute_types);
```

The following example, written in JAVA, is used to initialize an attributeTypeList instance before passing it to the get_attributes() of SecurityLevel1::Current interface, to get the attribute list returned.

```
org.omg.Security.AttributeType[] attributeTypeList =
  new org.omg.Security.AttributeType[2];

org.omg.Security.ExtensibleFamily attribute_family[0] =
  new org.omp.Security.ExtensibleFamily(
    com.ibm.IExtendedSecurity.IBM_BOSS_FAMILY_FAMILY_DEFINER.value,
      (short)2);

org.omg.Security.ExtensibleFamily attribute_family[1] =
  new org.omp.Security.ExtensibleFamily(
    com.ibm.IExtendedSecurity.IBM_BOSS_FAMILY_FAMILY_DEFINER.value,
      (short)2);

attrTypeList[0] = new org.omp.Security.AttributeType(
  attribute_family[0], com.ibm.IExtendedSecurity.CredAttrSecName.value );

attrTypeList[0] = new org.omp.Security.AttributeType(
  attribute_family[1], com.ibm.IExtendedSecurity.CredAttrHostName.value);
```

Chapter 46. SecurityLevel2 in the Security Service

The other modules in the Security Service are:
- IExtendedSecurity
- IExtendedSecurityClient
- Security
- SecurityLevel1

SecurityLevel2 Module

The SecurityLevel2 module contains the Credentials and the PrincipalAuthenticator interfaces.

File name

SecurityLevel2.idl

Types

```
typedef sequence Credentials CredentialsList;
```

Exceptions

```
enum InvalidCredentialReason {
  InvalidLoginContext,
  NoLoginContext,
  LoginContextExpired
};
exception InvalidCredential {
  InvalidCredentialReason reason;
};

enum LoginFailedReason {
  UnrecognizedPrincipalName,
  InvalidPassword,
  PasswordExpired,
  AccountDisabled,
  OutOfHours,
  CommunicationError,
  // ChangePassword - We may get this if we tried to login a
  //                  new account. Log a message so the user
  //                  can change his/her password.
}; // enum LoginFailedReason
exception LoginFailed {
  LoginFailedReason          reason;
  string           principal_name;
};
```

Interfaces

SecurityLevel2::Current Interface
SecurityLevel2::Credentials Interface

Credentials Interface

Represents a principal's current credential information for the session and therefore includes information such as that principal's security_name, host_name, etc.

File name

SecurityLevel2.idl

Intended usage

When a principal has been authenticated, the security service creates a credential for that principal. There are three different types of credentials. The one which is used depends on a particular context. Refer to "Acquiring a Credential on a Thread" in the *Component Broker Advanced Programming Guide* for more information.

Local-only

True

IDL syntax

```
interface Credentials {

   // Methods
   Credentials copy();
   void set_security_features(
     in Security::CommunicationDirection direction,
     in Security::SecurityFeatureValueList
                        security_features
   )
   raises (Security::InvalidCommDirection,
       Security::InvalidSecurityFeature,
       Security::DuplicateSecurityFeature);
   Security::SecurityFeatureValueList
     get_security_features(
       in Security::CommunicationDirection direction
   )
   raises (Security::InvalidCommDirection);
   Security::AttributeList get_attributes(
     in Security::AttributeTypeList attribute_types
   )
   raises (Security::InvalidAttributeType,
       Security::DuplicateAttributeType);
   boolean is_valid(
     out Security::UtcT expiry_time
   )
```

```
    raises (InvalidCredential);
    boolean refresh()
    raises (InvalidCredential);
  };
```

Supported operations

Credentials::copy
Credentials::get_attributes
Credentials::get_security_features
Credentials::is_valid
Credentials::refresh

Credentials::copy

Performs a ″deep copy″ of a Credentials object, returning the newly created duplicate object.

Not supported by OS/390 Component Broker

Original interface

SecurityLevel2::Credentials Interface

IDL syntax

```
Credentials copy();
```

Return value

The newly created duplicate object.

Remarks

This operation creates a new Credentials object, which is an exact duplicate (a ″deep copy″) of the Credentials object which is the target of the invocation. The return value is a reference to the newly created copy of the original Credentials object. This is typically used when an application wants to return a Credentials object to a caller and does not want the caller to be able to modify the original Credentials object.

Example

AIX WIN For AIX and Windows NT:

```
CORBA::ORB_ptr pTheORB;
CORBA::Object_ptr pObject;
SecurityLevel1::Current_ptr pSecCurrent;
```

```
SecurityLevel2::Credentials_ptr pCred;
SecurityLevel2::Credentials_ptr pCredCopy;

pTheORB = // use your CORBA::ORB_init operation with appropriate parameters
pObject = pTheORB->resolve_initial_references("SecurityCurrent");
pSecCurrent = SecurityLevel1::Current::_narrow(pObject);
pCred = pSecCurrent->get_credentials(Security::SecInvocationCredentials);
pCredCopy = pCred->copy();
```

SOLARIS For Solaris:

```
CORBA::ORB_ptr pTheORB;
CORBA::Current_ptr pOrbCurrent;
SecurityLevel1::Current_ptr pSecCurrent;
SecurityLevel2::Credentials_ptr pCred;
SecurityLevel2::Credentials_ptr pCredCopy;

pTheORB = // use your CORBA::ORB_init operation with appropriate parameters
pORBCurrent = pTheORB->get_current("SecurityLevel2::Current");
pSecCurrent = SecurityLevel1::Current::_narrow(pOrbCurrent);
pCred = pSecCurrent->get_credentials(Security::SecInvocationCredentials);

pCredCopy = pCred->copy();
```

Credentials::get_attributes

Returns the privilege and other attributes from a client's credentials. In a client, the client's privileges are returned. In a server, the received privileges of the client are returned. The server's privileges are returned if the server is not communicating with a client.

Original interface

SecurityLevel2::Credentials Interface

IDL syntax

```
Security::AttributeList get_attributes(
    in Security::AttributeTypeList attributes
  )
  raises (Security::InvalidAttributeType,
      Security::DuplicateAttributeType);
```

Parameters

attributes

The attributes whose values are to be returned. These attributes can be credential or privilege attributes. The defining authority and family values of these attributes are defined in the general security data module. Credential attributes identify credential information such as security name and host name. Privilege attributes identify privileges of a credential and the privileges for which access is authorized in an access control list.

Return value

In a client process, the client's attributes are returned. In a server process, the attributes of the client with whom the server is communicating is returned. If the server is not communicating with a client, the server's own attributes are returned.

Exceptions

Security::InvalidAttributeType
Security::DuplicateAttributeType

Remarks

Two attributes are supported on Component Broker for workstations. These are CredAttrSecName and CredAttrHostName, which refer to the DCE username and the host machine's name for the currently active Principal. To access these attributes, the family_definer field needs to be 8 and the family field needs to be 2.

On OS/390 Component Broker, the two supported attributes are AccessId and GroupId. AccessId maps to an MVS user identity that may be used with the SAF-based authorization services such as RACF. GroupId maps to a group identity that may be used with SAF-based authorization services. To access these attributes, the family_definer field needs to be set to 0 and the family field needs to be 1.

Example

AIX 390 WIN For AIX, OS/390, and Windows NT:

```
CORBA::ORB_ptr pTheORB;
CORBA::Object_ptr pObject;
SecurityLevel1::Current_ptr pSecCurrent;
SecurityLevel2::Credentials_ptr pCred;
Security::AttributeTypeList attribute_types(1);
Security::AttributeList_ptr pAttributes;

attribute_types[0].attribute_family.family_definer = 8; // = 0 for OS/390
attribute_types[0].attribute_family.family = 2; // = 1 for OS/390
attribute_types[0].attribute_type = Security::CredAttrSecName;

pTheORB =  // use your CORBA::ORB_init operation with appropriate parameters
pObject = pTheORB->resolve_initial_references("SecurityCurrent");
pSecCurrent = SecurityLevel1::Current::_narrow(pObject);
pCred = pSecCurrent->get_credentials(Security::SecInvocationCredentials);

pAttributes = pCred->get_attributes(attribute_types);
```

SOLARIS For Solaris:

```
CORBA::ORB_ptr pTheORB;
CORBA::Current_ptr pOrbCurrent;
SecurityLevel1::Current_ptr pSecCurrent;
SecurityLevel2::Credentials_ptr pCred;
```

```
Security::AttributeTypeList attribute_types(1);
Security::AttributeList_ptr pAttributes;

attribute_types[0].attribute_family.family_definer = 8;
attribute_types[0].attribute_family.family = 2;
attribute_types[0].attribute_type = Security::CredAttrSecName;

pTheORB =  // use your CORBA::ORB_init operation with appropriate parameters
pSecCurrent = pTheORB->get_current("SecurityLevel2::Current");
pCred = pSecCurrent->get_credentials(Security::SecInvocationCredentials);

pAttributes = pCred->get_attributes(attribute_types);
```

The following example, written in JAVA, is used to initialize an attributeTypeList instance before passing it to the get_attributes() of SecurityLevel2::Credentials interface, to get the attribute list returned.

```
org.omg.Security.AttributeType[] attributeTypeList =
  new org.omg.Security.AttributeType[2];

org.omg.Security.ExtensibleFamily attribute_family[0] =
  new org.omp.Security.ExtensibleFamily(
    com.ibm.IExtendedSecurity.IBM_BOSS_FAMILY_FAMILY_DEFINER.value,
      (short)2);

org.omg.Security.ExtensibleFamily attribute_family[1] =
  new org.omp.Security.ExtensibleFamily(
    com.ibm.IExtendedSecurity.IBM_BOSS_FAMILY_FAMILY_DEFINER.value,
      (short)2);

attrTypeList[0] = new org.omp.Security.AttributeType(
  attribute_family[0], com.ibm.IExtendedSecurity.CredAttrSecName.value );

attrTypeList[0] = new org.omp.Security.AttributeType(
  attribute_family[1], com.ibm.IExtendedSecurity.CredAttrHostName.value);
```

Credentials::get_security_features

Returns the security features associated with the Credentials.

Not supported by OS/390 Component Broker

Original interface

SecurityLevel2::Credentials Interface

IDL syntax

```
SecurityFeatureValueList get_security_features (
    in  CommunicationDirection direction
);
```

Parameters

direction
The communication direction (i.e., both, request, or reply) for which the security features should be retrieved. Normally set to both.

Return value

SecurityFeatureValueList
A sequence of required feature-value pairs. A boolean value of TRUE indicates that the feature is on. A value other than TRUE indicates that the feature is off.

Exceptions

Security::InvalidCommDirection
CORBA::NO_IMPLEMENT (OS/390)

Remarks

This operation is intended to be used by applications that need to know which set of security features is currently in use. The application passes in the direction parameter which indicates which set of security features (i.e., those set for request direction, reply direction, or both) should be returned.

The current release does not support the request direction or the reply direction. The both direction is supported.

Credentials::is_valid

Determines whether or not a credential has expired. It returns the expiration time of a credential.

Not supported by OS/390 Component Broker

Original interface

SecurityLevel2::Credentials Interface

IDL syntax

```
boolean is_valid(out Security::UtcT expiry_time)
   raises (InvalidCredential);
```

Parameters

expiry_time
The expiration time of the credential.

Return value

TRUE The credential has not expired.

FALSE The credential has expired.

Exceptions

SecurityLevel2::InvalidCredential
CORBA::NO_IMPLEMENT (OS/390)

Remarks

Credentials objects may have limited lifetimes. This operation is used to check if the Credentials are still valid. An application should typically invoke this operation immediately following a call to “Current::get_credentials” on page 797.

Example

AIX WIN For AIX and Windows NT:

```
CORBA::ORB_ptr pTheORB;
CORBA::Object_ptr pObject;
SecurityLevel1::Current_ptr pSecCurrent;
SecurityLevel2::Credentials_ptr pCred;
int rc=0;
Security::UtcT expiry_time;

pTheORB =  // use your CORBA::ORB_init operation with appropriate parameters
pObject = pTheORB->resolve_initial_references("SecurityCurrent");
pSecCurrent = SecurityLevel1::Current::_narrow(pObject);
pCred = pSecCurrent->get_credentials(Security::SecInvocationCredentials);

rc = pCred->is_valid(expiry_time);
```

SOLARIS For Solaris:

```
CORBA::ORB_ptr pTheORB;
CORBA::Current_ptr pOrbCurrent;
SecurityLevel1::Current_ptr pSecCurrent;
SecurityLevel2::Credentials_ptr pCred;
int rc=0;
Security::UtcT expiry_time;

pTheORB =  // use your CORBA::ORB_init operation with appropriate parameters
pSecCurrent = pTheORB->get_current("SecurityLevel2::Current");
pCred = pSecCurrent->get_credentials(Security::SecInvocationCredentials);

rc = pCred->is_valid(expiry_time);
```

Credentials::refresh

Allows the application to possibly extend the lifetime of credentials.

Not supported by OS/390 Component Broker

Original interface

SecurityLevel2::Credentials Interface

IDL syntax

```
boolean refresh()
   raises (SecurityLevel2::InvalidCredential);
```

Return value

TRUE If the credential was successfully updated.

FALSE If the credential was not successfully updated.

Exceptions

SecurityLevel2::InvalidCredential
CORBA::NO_IMPLEMENT (OS/390)

Remarks

Allows the application to update expired credentials. Typically, applications will invoke is_valid() to get the expiry_time and then, if the Credentials are ″about to expire″, invoke refresh().

The Credentials::refresh operation is only available for the server process credential which has the CredentialType of Security::SecOwnCredentials in the Component Broker for Windows NT and AIX workstation implementation.

Example

AIX WIN For AIX and Windows NT:

```
CORBA::ORB_ptr pTheORB;
CORBA::Object_ptr pObject;
SecurityLevel1::Current_ptr pSecCurrent;
SecurityLevel2::Credentials_ptr pCred;
int rc=0;

pTheORB =  // use your CORBA::ORB_init operation with appropriate parameters
pObject = pTheORB->resolve_initial_references("SecurityCurrent");
```

```
pSecCurrent = SecurityLevel1::Current::_narrow(pObject);
pCred = pSecCurrent->get_credentials(Security::SecInvocationCredentials);

rc = pCred->refresh();
```

SOLARIS For Solaris:

```
CORBA::ORB_ptr pTheORB;
CORBA::Current_ptr pOrbCurrent
SecurityLevel1::Current_ptr pSecCurrent;
SecurityLevel2::Credentials_ptr pCred;
int rc=0;

pTheORB =  // use your CORBA::ORB_init operation with appropriate parameters
pSecCurrent = pTheORB->get_current("SecurityLevel2::Current");
pCred = pSecCurrent->get_credentials(Security::SecInvocationCredentials);

rc = pCred->refresh();
```

Current Interface

This class contains the implementation of the CORBA SecurityLevel2 Current. This class provides access to security level 2 function as defined in the Object Management Group (OMG) specification.

File name

SecurityLevel2.idl

Intended usage

The Current object in its most general form represents any context information attached to a thread of execution. This can include security credentials, transaction context, or context information attached by other services and facilities (such as the Component Broker server adapter). Invoking the CORBA::ORB::resolve_initial_references request will return a CORBA::Object_ptr. The returned CORBA::Object_ptr can be specialized by each service that maintains context information on the thread. The application can narrow the Current object to the specific service interface of interest for which there is context information attached to the thread.

In particular, the security service introduces a specialization of the Current that will be returned from CORBA::ORB::resolve_initial_references when a security context is attached to the thread. The SecurityCurrent specialization can be used to retrieve invocation credentials, owner credentials, or received credentials. The set_credentials and get_policy methods will not be implemented in the current release.

Ancestor interfaces

SecurityLevel1::“Current Interface” on page 783

Supported methods

Current::get_credentials
Current::principal_authenticator
Current::received_credentials
Current::received_security_features

Current::get_credentials

Retrieves the Current credential of the specified type.

Original interface

SecurityLevel2::Current Interface

IDL syntax

```
SecurityLevel2::Credentials get_credentials(
     in Security::CredentialType cred_type)
   raises (Security::InvalidCredentialType);
```

Parameters

cred_typ

The type of credential to be retrieved. NonRepudiation type is not supported. Supported types are:

- Security::CredentialType::InvocationCredentials - these are the effective credentials used to make requests, etc. They may or may not be the same as OwnCredentials. The two types of credentials are different, for example, when a server uses a client's credentials in simple delegation: the client's credentials become the InvocationCredentials of the server.
- Security::CredentialType::OwnCredentials - a client or a server's own credentials.

Return value

The credential of the specified type associated with the Current object. Null if there are no credentials.

Exceptions

Security::InvalidCredentialType

Remarks

This operation allows an application access to the credentials associated with Current. The application can ask for the default credentials for future invocations or its own credentials. An application will normally get invocation or other credentials when it wants to modify them.

Example

AIX 390 WIN For AIX, OS/390, and Windows NT:

```
CORBA::ORB_ptr pTheORB;
CORBA::Object_ptr pObject;
SecurityLevel1::Current_ptr pSecCurrent;
SecurityLevel2::Credentials_ptr pCred;
SecurityLevel2::Current_ptr pSecL2Current;

pTheORB = // use your CORBA::ORB_init operation with appropriate parameters
pObject = pTheORB->resolve_initial_references("SecurityCurrent");
pSecL2Current = SecurityLocalObjectBasePrivateImpl::Current::_narrow(pObject);
pCred = pSecL2Current->get_credentials(Security::SecInvocationCredentials);
```

SOLARIS For Solaris:

```
CORBA::ORB_ptr pTheORB;
SecurityLevel1::Current_ptr pSecCurrent;
SecurityLevel2::Credentials_ptr pCred;
SecurityLevel2::Current_ptr pSecL2Current;

pTheORB = // use your CORBA::ORB_init operation with appropriate parameters
pSecCurrent = pTheORB->get_current("SecurityLevel2::Current");
pSecL2Current =
   SecurityLocalObjectBasePrivateImpl::Current::_narrow(pSecCurrent);
pCred = pSecL2Current->get_credentials(Security::SecInvocationCredentials);
```

Current::principal_authenticator

Returns the principal_authenticator object for the thread represented by the Current object.

Not supported by OS/390 Component Broker

Original class

SecurityLevel2::Current Class

C++ syntax

```
SecurityLevel2::PrincipalAuthenticator_ptr principal_authenticator()
```

Return value

SecurityLevel2::PrincipalAuthenticator_ptr

Remarks

The principal_authenticator method returns a thread specific object. It is intended to be used by an application to authenticate principals and obtain Credentials containing their attributes.

Current::received_credentials

Returns the received_credentials for the thread represented by the Current object.

Original class

SecurityLevel2::Current Class

C++ syntax

```
SecurityLevel2::CredentialsList* received_credentials();
```

Return value

SecurityLevel2::CredentialsList*
One credential is returned in this list using the current release.

Remarks

The received_credentials method is intended to be used by implementations when executing a remote request within a server process to determine the attributes of the client process that has issued the remote request. Currently, the attributes supported are the user name of the client process (as it is known to DCE) and the host name of the machine from which the request originated. You must perform get_attributes on the received credentials to obtain the client's attributes.

Current::received_security_features

Returns a sequence of feature-value pairs which indicate whether the feature is on or not.

Not supported by OS/390 Component Broker

Original class

SecurityLevel2::Current Class

C++ syntax

```
SecurityLevel2::SecurityFeatureValueList* received_security_features()
```

Return value

SecurityLevel2::SecurityFeatureValueList*

A sequence of feature-value pairs. A boolean value of TRUE indicates that the feature is on. Values other than TRUE indicate that the feature is off.

Remarks

The received_security_features attribute is a thread specific attribute at the target application which provides the security features of the message sent by the client. This attribute is intended to be used by implementations when executing a remote request within a server process to determine the ″security features″ of the client process that has issued the remote request. Examples include delegation and quality of protection types.

PrincipalAuthenticator Interface

The principal authenticator is a pseudo object that is instantiated to perform authentication. The principal authenticator encapsulates the specifics of authentication for a particular authentication mechanism. The responsibility of the authenticator is to produce a credential for a principal. In the case of DCE, this means obtaining a ticket-granting-ticket for the principal.

Not supported by OS/390 Component Broker

File name

SecurityLevel2.idl

Intended usage

Used by LoginHelper or secure server to authenticate a user principal or server principal.

Local-only

True

Ancestor interfaces

IManagedLocal::ILocalOnly

IDL syntax

```
interface PrincipalAuthenticator : IManagedLocal::ILocalOnly {

       Security::AuthenticationStatus authenticate(
```

```
        in Security::AuthenticationMethod   method,
        in string                           security_name,
        in Security::Opaque                 auth_data,
        in Security::AttributeList          privileges,
        out Credentials                     creds,
        out Security::Opaque                continuation_data,
        out Security::Opaque                auth_specific_data
        )
    raises (LoginFailed, Security::InvalidAuthnMethod,
            Security::InvalidAttributeType,
            Security::DuplicateAttributeType);

    Security::AuthenticationStatus continue_authentication (
        in Security::Opaque     response_data,
        inout Credentials       creds,
        out Security::Opaque    continue_data,
        out Security::Opaque    auth_specific_data
        );
    };
```

Supported operations

PrincipalAuthenticator::authenticate
PrincipalAuthenticator::continue_authentication

PrincipalAuthenticator::authenticate

This operation authenticates a principal and creates a Credentials object including the required attributes.

Original interface

SecurityLevel2::PrincipalAuthenticator Interface

IDL syntax

```
Security::AuthenticationStatus authenticate(
          in Security::AuthenticationMethod   method,
          in string                           security_name,
          in Security::Opaque                 auth_data,
          in Security::AttributeList          privileges,
          out Credentials                     creds,
          out Security::Opaque                continuation_data,
          out Security::Opaque                auth_specific_data
          )
   raises (LoginFailed, Security::InvalidAuthnMethod,
           Security::InvalidAttributeType,
           Security::DuplicateAttributeType);
```

Parameters

method
The identifier of the authentication method used. Currently, the only method supported is: ISecurityLocalObjectDCEImpl::AuthnMethodDCE.

security_name
The principal's login name (DCE user id).

auth_data
The principal's authentication information (DCE password).

privileges
The privilege attributes requested. (ignored in the current release).

creds Object reference of the newly created Credentials object.

continuation_data
If the return parameter from the authenticate operation is 'Continue,' then this parameter contains challenge information for authentication continuation (not used in the current release).

auth_specific_data
Information specific to the particular authentication service used (not used in the current release).

Return value

SecAuthSuccess
Indicates that credentials were successfully created.

SecAuthFailure
Indicates that credentials were not created.

SecAuthContinue
Indicates presence of continuation_data (not available in the current release).

SecAuthExpired
Indicates that authentication data (i.e., password) has expired.

Exceptions

SecurityLevel2::LoginFailed
Security::InvalidAuthnMethod
Security::InvalidAttributeType
Security::DuplicateAttributeType

Remarks

This operation is intended to be used by applications that need to authenticate a principal and optionally request privilege and other attributes that the principal requires during its session with the system. For example, this operation is called by LoginHelper.

PrincipalAuthenticator::continue_authentication

Continues the authentication process for authentication procedures that annot complete in a single operation.

This operation is not implemented in the current release.

Original interface

SecurityLevel2::PrincipalAuthenticator Interface

IDL syntax

```
Security::AuthenticationStatus continue_authentication (
            in Security::Opaque     response_data,
            inout Credentials       creds,
            out Security::Opaque    continue_data,
            out Security::Opaque    auth_specific_data
            );
```

Parameters

response_data
The response data to the challenge.

creds Object reference of the partially initialized Credentials object. The Credentials object is fully initialized only when the return parameter is ′Success.'

continuation_data
If the return parameter from the authenticate operation is ′Continue,' then this parameter contains challenge information for authentication continuation (not used in the current release).

auth_specific_data
Information specific to the particular authentication service used (not used in the current release).

Return value

SecAuthSuccess
Indicates that the Credentials object whose reference was identified by the creds parameter is now fully initialized.

SecAuthFailure
Indicates that credentials were not created.

SecAuthContinue
Indicates presence of continuation_data (not used in current release).

SecAuthExpired
Indicates that authentication data (i.e., password) has expired.

Remarks

An example of this usage would be a challenge/response type of authentication procedure.

Chapter 47. Object-Oriented SQL

Identifiers

An *identifier* is a token used to form a name. An *OOSQL simple identifier* is an upper or lower-case letter followed by zero or more characters, each of which is an upper- or lower-case letter, a digit, or the underscore character. An *OOSQL identifier* is an optional "::" followed by a simple identifier followed by zero or more occurrences of "::" followed by a simple identifier.

The maximum length of an OOSQL identifier or constant is 2048 single byte or multibyte characters. However, the length of identifiers should be restricted to 128 bytes for portability.

Note: A *delimited identifier* is a sequence of one or more characters enclosed within escape characters. The escape character is the quotation mark. A delimited identifier can be used when the sequence of characters does not qualify as an ordinary identifier. Such a sequence, for example, could be an SQL reserved word.

Comparisons

Comparison operators are performed during the execution of statements that include predicates and other language elements such as MAX, MIN, DISTINCT, GROUP BY, and ORDER BY.

Numeric comparisons

Numbers are compared algebraically; that is, with regard to sign.

When one number is an integer and the other is a decimal, the comparison is made with a temporary copy of the integer which has been converted to decimal.

When decimal numbers with different scales are compared, the comparison is made with a temporary copy of one of the numbers that has been extended with trailing zeros so that its fractional part has the same number of digits as the other number.

When one number is double precision floating-point and the other is integer, decimal, or single-precision floating-point; the comparison is made with a temporary copy of the other number (converted to double-precision floating-point). However, if a single-precision floating-point is compared with a floating-point constant, the comparison is made with a single-precision form of the constant.

Two floating-point numbers are equal only if the bit configurations of their normalized forms are identical.

String comparisons

The query engine uses the portable wcscoll function to perform multibyte character comparisons for >, ≥, <, ≤. The multibyte characters are converted to wide characters before the comparisons. In order to be compatible with DB2, character comparisons are performed after removing trailing blanks.

Query push down can occur for equality and non-equality character comparisons. However, all other comparisons are performed in memory.

Datetime comparisons

DATE, TIME, or TIMESTAMP value may be compared with another value of the same data type or with a string representation of that data type. All comparisons are chronological. The later the time from January 1, 0001, the greater the value of that point in time.

Comparisons involving TIME values and string representation of time values always include seconds. If the string representation omits seconds, zero seconds are implied.

Comparisons involving TIMESTAMP values are chronological without regard to representations that might be considered equivalent. Thus, the following predicate is true:

```
TIMESTAMP('1990-02-23-00.00.00') > '1990-02-22-24.00.00'
```

Constants

A *constant* (also called a *literal*) specifies a value. Constants are classified into two categories: numeric constants and string constants. Numeric constants are further classified as integer constant, small integer constant, and floating-point constants. String constants are classified as character strings.

Integer constants

An *integer constant* specifies a binary integer as a signed or unsigned number without decimal. If the value is not within the range of a integer, an error is reported. The data type of an integer constant is `integer`.

Note: In syntax diagrams, the term “integer” is used for an integer constant that must not include a sign.

Small integer constants

A *small integer constant* specifies a binary integer as a signed or unsigned number without decimal. If the value is not within the range of a small integer, an error is reported. The data type of a small integer constant is `short`.

Floating-point constants

A *floating-point constant* specifies a floating-point as two numbers separated by an “E”. The first number can include a sign and a decimal point. The second number can include a sign (but no decimal point). The value of the constant is the product of the first number and the power of 10 specified by the second number. The data type of a floating-point constant is double-precision floating-point.

Decimal constants

A *decimal constant* is a signed or unsigned number that consists of no more than 38 digits and includes a decimal point. The precision is the total number of digits (including leading and trailing zeros); the scale is the number of digits to the right of the decimal point (including trailing zeros).

Examples: The following decimal constants have, respectively, precisions and scales of 5 and 2; 4 and 0; 2 and 0; 23 and 2:

```
025.50
1000.
-15.
+375893333333333333333.33
```

Character string constants

A *character string constant* specifies a varying-length character string. The character string constant is a sequence of characters that start and end with a sting delimiter (an apostrophe or single quotation mark). This form of string constant specifies the character string contained between the string delimiters. The number of bytes between the delimiters must not be greater than 2048. Two consecutive string delimiters are used to represent one string delimiter within the character string.

Note: At DBCS sites, a character string constant is classified as a multibyte string.

The character string constant can be the character string representation of dates, times, and timestamps. Examples are:

```
'12/14/1985'
'32'
'DON''T CHANGE'
''
```

The last example `''` represents an empty character string constant (a zero length string).

The transformation of IDL/CORBA types to result types is:

Table 4. Evaluation of IDL/CORBA types in OOSQL queries

IDL/CORBA Type	OOSQL Type	Result Type	Comments
char	varchar(1)	string	

Table 4. Evaluation of IDL/CORBA types in OOSQL queries (continued)

IDL/CORBA Type	OOSQL Type	Result Type	Comments
wchar	vargraphic(1)	wstring	wide character
unsigned short	integer	integer	
short	smallint	short	
long	integer	long	
long long	integer	long	exception raised if cannot fit in long
unsigned long	integer	long	exception raised if cannot fit in long
float	real	float	
double	double precision	double	
struct	struct	reference to struct	reference to allocated buffer
union	union	reference to union	not supported
array	array	reference to array	not supported
enum	integer	long	
boolean	smallint	short	zero=false, non-zero=true
octet	integer	long	
any	any	reference to any	not supported
sequence	sequence	reference to sequence	not supported
sequence<octet>	varchar(n) for bit data	sequence<octet>	
string	varchar	string	reference to char
wstring	vargraphic	wstring	wide character string
object type	object type	reference to object type	reference to interface definition

The mapping of Oracle and DB2 MVS types to the OOSQL types is:

Table 5. Evaluation of ORACLE, DB2 MVS and DB2 CS data types in OOSQL queries.

ORACLE	DB2 CS Type	DB2 MVS Type	IDL/CORBA Type	OOSQL Type	Comments
number(p,s), 5≤p<10, s=0	integer	integer	long	integer	
number(p,s), 1≤p<5, s=0	smallint	smallint	short	smallint	
		float (1-21) or real	float	float or real	4 byte float

Table 5. Evaluation of ORACLE, DB2 MVS and DB2 CS data types in OOSQL queries. (continued)

ORACLE	DB2 CS Type	DB2 MVS Type	IDL/CORBA Type	OOSQL Type	Comments
number(p,s), p>38 or s>38 or p=0, s=0	double or float	float (22-53) or float or double or double-precision	double	double-precision	8 byte float
number(p,s), 10≤p≤38, s≤38	decimal (p,s)	decimal (p,s)	string	decimal (p,s)	special handling of decimal as double
	date	date	string	date	special handling of date types
	time	time	string	time	special handling of time types
date	timestamp	timestamp	string	timestamp	special handling of timestamp types
	duration	duration	string	duration	special handling of duration types
char(n)	char(n)	char(n)	string	varchar(n)	null-terminated string
varchar2(n)	varchar(n)	varchar(n)	string	varchar(n)	null-terminated string
long	long varchar	long varchar	string	varchar	null-terminated string
nchar(n)	graphic(n)	graphic(n)	wstring	vargraphic(n)	wide character string. Not supported for ORACLE.
nvarchar2(n)	vargraphic(n)	vargraphic(n)	wstring	vargraphic(n)	wide character string. Not supported for ORACLE.
	long vargraphic	long vargraphic	wstring	vargraphic	wide character string
raw(n)	varchar(n) for bit data	varchar(n) for bit data	sequence<octet>	varchar(n) for bit data	
	char(n) for bit data	char(n) for bit data	sequence<octet>	character(n) for bit data	

Table 5. Evaluation of ORACLE, DB2 MVS and DB2 CS data types in OOSQL queries. (continued)

ORACLE	DB2 CS Type	DB2 MVS Type	IDL/CORBA Type	OOSQL Type	Comments
long raw	long varchar(n) for bit data	long varchar(n) for bit data	sequence<octet>	varchar for bit data	
blob	blob				not mapped
clob	clob				not mapped
nclob	dbclob				not mapped
bfile				bfile	not mapped

Member names

The meaning of a member name depends on its context. A member name is an attribute or method. A member name can be used to specify values of the member, as in the following contexts.

- In an *aggregate function*, a member name specifies all values of the member in the group on the intermediate result collection to which the function is applied. For example, `MAX(SALARY)` applies the function MAX to all values of the member SALARY in the group.
- In a GROUP BY or ORDER BY clause, a member name specifies all values in the intermediate result collection to which the clause is applied. For example, `ORDER BY DEPT` orders an intermediate result collection by the values of the member DEPT.
- In an expression, search condition, or scalar function a member name specifies a value for each object or group to which the construct is applied. For example, when the search condition `CODE=20` is applied to an object, the value specified by the member name CODE is the value of the member CODE in that object.

Qualified member names

A qualifier for a member name can be a collection name or a correlation name.

A member name is qualified depending on its context.

- Where the member name specifies values of the member, a member name can be qualified at the user option.
- Path expressions with more than one node need to be qualified.
- In all other contexts, a member name must not be qualified.

Where a qualifier is optional, it can serve two purposes: avoid ambiguity or correlation.

Correlation names

A *correlation name* can be defined in the FROM clause. For example, the clause `FROM X.empHome Z` establishes Z as a correlation name for X.empHome. With Z defined as a

correlation name for X.empHome, only Z should be used to qualify a reference to a member of X.empHome in the SELECT statement.

A correlation name is associated with a collection only within the context in which it is defined. Therefore, the same correlation name can be defined for different purposes in different statements or in different clauses of the same statement.

As a qualifier, a correlation name can be used to avoid ambiguity or to establish a correlated reference. It can also be used to shorten the name for a collection. In the example, Z might be used to avoid having to enter X.empHome more than once.

The correlation name is required for path expressions with multiple members or a single member that is a method.

Member name qualifiers to avoid ambiguity

In the context of a function, a GROUP BY clause, ORDER BY clause, an expression, or a search condition, a member name refers to values of a member in some collection. The collections that might contain the member are called the object collections of the context. Two or more object collections might contain members with the same name. One reason for qualifying a member name is to name the collection from which the member comes.

Collection designators

A qualifier that names a specific object collection is called a collection designator. The clause that identifies the object collections also establishes the collection designators for them. For example, the object collections of an expression in a SELECT

clause are named in the FROM clause that follows it, as in this statement:

```
SELECT DISTINCT Z.EMPNO, EMPTIME, PHONENO
  FROM EMP Z, EMPPROJACT
  WHERE WORKDEPT = 'D11'
    AND EMPTIME > 0.5
    AND Z.EMPNO = EMPPROJACT.EMPNO;
```

This example illustrates how to establish collection designators in the FROM clause:

- A name that follows a collection name is both a correlation name and a collection designator. Thus, Z is a collection designator and qualifies the first member name after SELECT.
- A collection name that is not followed by a correlation name is a collection designator. Thus, the collection name, EMPPROJACT is a collection designator and qualifies the EMPNO member.

Avoiding undefined or ambiguous references

When a member name refers to values of a member, exactly one object collection must include a member with that name. The following situations are considered errors:

- No object collection contains a member with the specified name. The reference is undefined.
- The member name is qualified by a collection designator, but the collection named does not include a member with the specified name. Again, the reference is undefined.
- The name is unqualified and more than one object collection includes a member with that name. The reference is ambiguous.

Avoid ambiguous references by qualifying a member name with a uniquely defined collection designator. If the member is contained in several object collections with different names, the collection names can be used as designators.

Two or more object collections can be instances of the same collection. A FROM clause that includes *n* references to the same collection should include at least n-1 unique correlation names.

For example, in the following FROM clause X and Y are defined to refer, respectively, to the first and second instances of the collection EMP.

```
SELECT X.LASTNAME, Y.LASTNAME
  FROM EMP X, EMP Y
  WHERE Y.JOB = 'MANAGER'
    AND X.WORKDEPT = Y.WORKDEPT
    AND X.JOB <> 'MANAGER';
```

Member name qualifiers in correlated references

A *subselect* is a form of a query that can be used as a component of various OOSQL statements. Refer to “Queries” on page 852 for more information on subselects. A subselect used within a search condition of any statement is called a *subquery*.

A subquery can include search conditions of its own, and these search conditions can, in turn, include subqueries. Thus, an OOSQL statement can contain a hierarchy of subqueries. Those elements of the hierarchy that contain subqueries are said to be at a higher level than the subqueries they contain.

Every element of the hierarchy has a clause that establishes one or more collection designators. This is the FROM clause. A search condition of a subquery can reference not only members of the collections identified by the FROM clause of its own element of the hierarchy, but also members of collections identified at any level along the path from its own element to the highest level of the hierarchy. A reference to a member of a collection identified at a higher level is called a correlated reference.

A correlated reference to member C of collection T can be of the form C, T.C, or Q.C, if Q is a correlation name defined for T. However, a correlated reference in the form of an unqualified member name is not good practice. The following explanation is based on the assumption that a correlated reference is always in the form of a qualified member name and that the qualifier is a correlation name.

A qualified member name, Q.C, is a correlated reference only if these three conditions are met:

- Q.C is used in a search condition of a subquery.
- Q does not name a collection used in the FROM clause of that subquery.
- Q does name a collection used at some higher level.

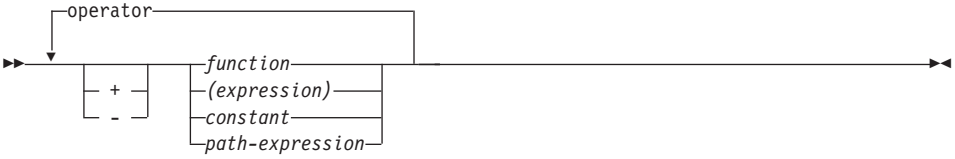
Q.C refers to member C of the collection at the level where Q is used as the collection designator of that collection. Because the same collection can be identified at many levels, unique correlation names are recommended as collection designators. If Q is used to name a collection at more than one level, Q.C refers to the lowest level that contains the subquery that includes Q.C.

For example, in the following statement, the correlated reference X.WORKDEPT (in the last line) refers to the value of WORKDEPT in collection EMP at the level of the first FROM clause (which establishes X as a correlation name for EMP.). The statement lists employees who make less than the average salary for their department.

```
SELECT EMPNO, LASTNAME, WORKDEPT
  FROM EMP X
  WHERE SALARY < (SELECT AVG(SALARY)
    FROM EMP
    WHERE WORKDEPT = X.WORKDEPT);
```

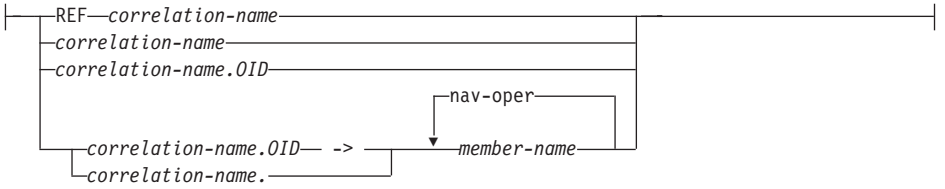
Expressions

An *expression* specifies a value. The form of an *expression* is as follows:



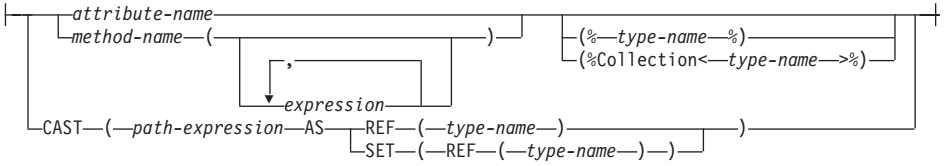
operator:

path-expression:

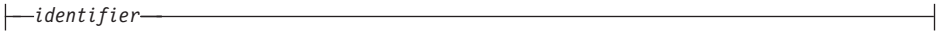


nav-oper:

member-name:



correlation-name:



Note: 390 SOLARIS OS/390 and Solaris do not support the CAST ..AS syntax , the keyword OID or the right arrow (->) operator.

Expression specifies a value. Expression can be:

without operators
If no operators are used, the result of the expression is the specified value.

with navigational operator
If the navigational operation "->" is used, the result of the expression is the value of the leaf of the expression. If any member in the expression has the null value, the result of the expression is the null value. The characters ".." can also be used for traversal. However, the traversal through OID is expressed only by "->" and "->" cannot be used for traversal through members that are not of the reference type. For example, the embedded structures and collections can only use ".." for traversal.

with cast operator
If the cast operator CAST AS or (*%type-name%*) is used, the result of the

expression is the value of the component derived by setting the type of the component to the type-name. If any member in the component selection has the null value, the result of the expression is the null value. If the attribute-name is of the reference type, then the expression CAST (*attribute-name* AS REF (*type-name*)) is used to set the type of *attribute-name* to *type-name* and is equivalent to *attribute-name*(*%type-name%*). If the attribute-name is of the collection type, then the expression CAST (*attribute-name* AS SET (REF (*type-name*))) is used to set the type of collection elements to *type-name* and is equivalent to *attribute-name*(*%Collection<type-name>%*).

with arithmetic operators

If arithmetic operators are used, the result of the expression is a value derived from the application of the operators to the values of the operands. If any operand can be null, or the expression is used in an outer SELECT list, the result can be null. If any operand has the null value, the result of the expression is the null value.

Arithmetic operators (except unary plus) must not be applied to strings. Fir example, USER+2 is invalid. Arithmetic operators can be applied to date/time values.

The prefix operator + (unary plus) does not change its operand.

The prefix operator - (unary minus) reverses the sign of a non-zero operand.

The infix operators +, -, *, and / specify addition, subtraction, multiplication, and division, respectively. The value of the second operand of division must not be zero.

Arithmetic with two integer operands

If both operands of an arithmetic operator are integers, the operation is performed in binary and the result is an integer. Any remainder of division is lost. The result of an integer arithmetic operation (including unary minus) must be within the range of integers.

Arithmetic with an integer and a decimal operand

If one operand is an integer and the other is decimal, the operation is performed in decimal using a temporary copy of the integer that has been converted to a decimal number with precision *p* and scale 0. *p* is 11 for an integer and 5 for a small integer.

Arithmetic with two decimal operands

If both operands are decimal, the operation is performed in decimal. The result of any decimal arithmetic operation is a decimal number with a precision and scale that are dependent on the operation and the precision and scale of the operands. If the operation is addition or subtraction and the operands do not have the small scale, the operation is performed with a temporary copy of one of the operands. The copy of the shorter operand is extended with trailing zeros so that its fractional part has the same number of digits as the longer operand.

The result of a decimal operation must not have a precision greater than 38. The result of decimal addition, subtraction, and multiplication is derived from a temporary result which may have a precision greater than 38. If the precision of the temporary result is not greater than 38, the final result is the same as the temporary result.

The following formulas define the precision and scale of the result of decimal operations in OOSQL. The symbols *p* and *s* denote the precision and scale of the first operand, and the symbols *p'* and *s'* denote the precision and scale of the second operand.

Decimal addition and subtraction

The precision of the result is the minimum of 38 and the quantity MAX(*p*-*s*,*p'*-*s'*)+MAX(*s*,*s'*)+1. The scale is MAX(*s*,*s'*).

Decimal multiplication

For multiplication, the precision of the result is MIN(38,*p*+*p'*), and the scale is MIN(38,*s*+*s'*).

Decimal division

The precision of the result of division is 38. The scale is 38-*p*+*s*-*s'*. The scale must not be negative.

Arithmetic with lloating-point operands

If either operand of an arithmetic operator is floating-point, the operation is performed in floating-point. Thus, if any element of an expression is a floating-point number, the result of the expression is a double-precision floating-point number.

An operation involving a floating-point number and an integer is performed with a temporary copy of the integer (converted to double-precision floating-point). An operation involving a floating-point number and a decimal number is performed with a temporary copy of the decimal number (converted to double-precision floating-point). The result of a floating-point operation must be within the range of floating-point numbers.

Datetime operands and durations

Datetime values can be incremented, decremented, and subtracted. These operations may involve decimal numbers called durations. A *duration* is a number representing an interval of time. There are three types of durations:

Date Duration
A *date duration* represents a number of years, months, and days expressed as a DECIMAL(8,0) number. To be properly interpreted, the number must have the format *yyyymmdd*, where *yyyy* represents the number of years, *mm* the

number of months, and *dd* the number of days. The result of subtracting one DATE value from another, as in the expression HIREDATE - BIRTHDATE, is a date duration.

Time Duration

A *time duration* represents a number of hours, minutes, and seconds expressed as a DECIMAL(6,0) number. To be properly interpreted, the number must have the format *hhmmss* where *hh* represents the number of hours, *mm* the number of minutes, and *ss* the number of seconds. The result of subtracting one TIME value from another is a time duration.

Timestamp Duration

A *timestamp duration* represents a number of years, months, days, hours, minutes, seconds, and microseconds expressed as a DECIMAL(20,6) number. To be properly interpreted, the number must have the format *yyyyxxddhhmmsszzzzzz,* where *yyyy, xx, dd, hh, mm,* and *ss* represent, respectively, the number of years, months, days, hours, minutes, and seconds, and zzzzzz represents the number of microseconds. The result of subtracting one TIMESTAMP value from another is a timestamp duration.

Datetime arithmetic in SQL

The only arithmetic operations that can be performed on datetime values are addition and subtraction. If a datetime value is the operand of addition, the other operand must be a duration. The specific rules governing the use of the addition operator with datetime values follow.

- If one operand is a date, the other operand must be a date duration.
- If one operand is a time, the other operand must be a time duration.
- If one operand is a timestamp, the other operand must be a duration.

The rules for the use of the subtraction operator on datetime values are not the same as those for addition because a datetime value cannot be subtracted from a duration, and because the operation of subtracting two datetime values is not the same as the operation of subtracting a duration from a datetime value. The specific rules governing the use of the subtraction operator with datetime values follow.

- If the first operand is a date, the second operand must be a date, a date duration, a string representation of a date.
- If the second operand is a date, the first operand must be a date, or a string representation of a date.
- If the first operand is a time, the second operand must be a time, a time duration, a string representation of a time.
- If the second operand is a time, the first operand must be a time, or string representation of a time.
- If the first operand is a timestamp, the second operand must be a timestamp, a string representation of a timestamp, or a duration.
- If the second operand is a timestamp, the first operand must be a timestamp or a string representation of a timestamp.

Date arithmetic

Dates can be subtracted, incremented, or decremented.

Subtracting dates: The result of subtracting one date (DATE2) from another (DATE1) is a date duration that specifies the number of years, months, and days between the two dates. The data type of the result is DECIMAL(8,0). If DATE1 is greater than or equal to DATE2, DATE2 is subtracted from DATE1. If DATE1 is less than DATE2, however, DATE1 is subtracted from DATE2, and the sign of the result is made negative. The following procedural description clarifies the steps involved in the operation RESULT = DATE1 - DATE2.

- If *DAY(DATE2)* <= *DAY(DATE1)*
  then *DAY(RESULT)* = *DAY(DATE1)* - *DAY(DATE2)*.

- If *DAY(DATE2)* > *DAY(DATE1)*
  then *DAY(RESULT)* = *N* + *DAY(DATE1)* - *DAY(DATE2)*
  where *N* = the last day of *MONTH(DATE2)*.
  MONTH(DATE2) is then incremented by 1.

- If *MONTH(DATE2)* <= *MONTH(DATE1)*
  then *MONTH(RESULT)* = *MONTH(DATE1)* - *MONTH(DATE2)*.

- If *MONTH(DATE2)* > *MONTH(DATE1)*
  then *MONTH(RESULT)* = *12* + *MONTH(DATE1)* - *MONTH(DATE2)*
  and *YEAR(DATE2)* is incremented by 1.

- *YEAR(RESULT)* = *YEAR(DATE1)* - *YEAR(DATE2)*.

For example, the result of DATE('3/15/2000') - '12/31/1999' is 215 (or, a duration of 0 years, 2 months, and 15 days).

Incrementing and decrementing dates: The result of adding a duration to a date, or of subtracting a duration from a date, is itself a date. (For the purposes of this operation, a month denotes the equivalent of a calendar page. Adding months to a date, then, is like turning the pages of a calendar, starting with the page on which the date appears.) The result must fall between the dates January 1, 0001 and December 31, 9999 inclusive. If a duration of years is added or subtracted, only the year portion of the date is affected. The month is unchanged, as is the day unless the result would be February 29 of a non-leap-year. Here the day portion of the result is set to 28.

Similarly, if a duration of months is added or subtracted, only months and, if necessary, years are affected. The day portion of the date is unchanged unless the result would be invalid (September 31, for example). In this case the day is set to the last day of the month.

Adding or subtracting a duration of days will, of course, affect the day portion of the date, and potentially the month and year.

Date durations, whether positive or negative, can also be added to and subtracted from dates.

When a positive date duration is added to a date, or a negative date duration is subtracted from a date, the date is incremented by the specified number of years, months, and days, in that order.

When a positive date duration is subtracted from a date, or a negative date duration is added to a date, the date is decremented by the specified number of days, months, and years, in that order.

Adding a month to a date gives the same day one month later, unless that day does not exist in the later month. In that case, the day in the result is set to the last day of the later month. For example, January 28 plus one month gives February 28; one month added to January 29, 30, or 31 results in either February 28 or, for a leap year, February 29. If one or more months is added to a given date and then the same number of months is subtracted from the result, the final date is not necessarily the same as the original date.

Time arithmetic

Times can be subtracted, incremented, or decremented.

Subtracting times: The result of subtracting one time (TIME2) from another (TIME1) is a time duration that specifies the number of hours, minutes, and seconds between the two times. The data type of the result is DECIMAL(6,0). If TIME1 is greater than or equal to TIME2, TIME2 is subtracted from TIME1. If TIME1 is less than TIME2, however, TIME1 is subtracted from TIME2, and the sign of the result is made negative. The following procedural description clarifies the steps involved in the operation RESULT = TIME1 - TIME2.

```
• If SECOND(TIME2) <=
SECOND(TIME1)
    then SECOND(RESULT) = SECOND(TIME1) - SECOND(TIME2).

• If SECOND(TIME2) > SECOND(TIME1)
    then SECOND(RESULT) = 60 + SECOND(TIME1) - SECOND(TIME2)
      and MINUTE(TIME2) is incremented by 1.

• If MINUTE(TIME2) <= MINUTE(TIME1)
    then MINUTE(RESULT) = MINUTE(TIME1) - MINUTE(TIME2).

• If MINUTE(TIME2) > MINUTE(TIME1)
    then MINUTE(RESULT) = 60 + MINUTE(TIME1) - MINUTE(TIME2)
      and HOUR(TIME2) is incremented by 1.

• HOUR(RESULT) = HOUR(TIME1) - HOUR(TIME2).
```

For example, the result of TIME('11:02:26') - '00:32:56' is 102930 (a duration of 10 hours, 29 minutes, and 30 seconds).

Incrementing and decrementing times: The result of adding a duration to a time, or of subtracting a duration from a time, is itself a time. Any overflow or underflow of hours is discarded, thereby ensuring that the result is always a time. If a duration of hours is added or subtracted, only the hours portion of the time is affected. Adding 24 hours to

the time '00:00:00' results in the time '24:00:00'. However, adding 24 hours to any other time results in the same time; for example, adding 24 hours to the time '00:00:59' results in the time '00:00:59'. The minutes and seconds are unchanged.

Similarly, if a duration of minutes is added or subtracted, only minutes and, if necessary, hours are affected. The seconds portion of the time is unchanged.

Adding or subtracting a duration of seconds affects the seconds portion of the time and may affect the minutes and hours.

Time durations, whether positive or negative, can also be added to and subtracted from times. The result is a time that has been incremented or decremented by the specified number of hours, minutes, and seconds, in that order.

Timestamp arithmetic

Timestamps can be subtracted, incremented, or decremented.

Subtracting timestamps: The result of subtracting one timestamp (TS2) from another (TS1) is a timestamp duration that specifies the number of years, months, days, hours, minutes, seconds, and microseconds between the two timestamps. The data type of the result is DECIMAL(20,6). If TS1 is greater than or equal to TS2, TS2 is subtracted from TS1. If TS1 is less than TS2, however, TS1 is subtracted from TS2 and the sign of the result is made negative. The following procedural description clarifies the steps involved in the operation RESULT = TS1 - TS2.

- If *MICROSECOND(TS2)* <= *MICROSECOND(TS1)*
  then *MICROSECOND(RESULT)* = *MICROSECOND(TS1)* - *MICROSECOND(TS2)*.

- If *MICROSECOND(TS2)* > *MICROSECOND(TS1)*
  then *MICROSECOND(RESULT)* = *1000000* + *MICROSECOND(TS1)*
  - *MICROSECOND(TS2)*
  and *SECOND(TS2)* is incremented by 1.

- The seconds and minutes part of the timestamps are subtracted as specified in the rules for subtracting times.

- If *HOUR(TS2)* <= *HOUR(TS1)*
  then *HOUR(RESULT)* = *HOUR(TS1)* - *HOUR(TS2)*.

- If *HOUR(TS2)* > *HOUR(TS1)*
  then *HOUR(RESULT)* = *24* + *HOUR(TS1)* - *HOUR(TS2)*
  and *DAY(TS2)* is incremented by 1.

- The date part of the timestamps is subtracted as specified in the rules for subtracting dates.

Incrementing and decrementing timestamps: The result of adding a duration to a timestamp, or of subtracting a duration from a timestamp, is itself a timestamp. Date and time arithmetic is performed as previously defined, except that an overflow or underflow of hours is carried into the date part of the result, which must be within the

range of valid dates. When the result of an operation is midnight, the time portion of the result can be '24.00.00'or '00.00.00' a comparison of those two values does not result in 'equal'.

Precedence of operations

Expressions within parentheses are evaluated first. When the order of evaluation is not specified by parentheses, prefix operators are applied before multiplication and division, and multiplication, division, and concatenation are applied before addition and subtraction. Operators at the same precedence level are applied from left to right. For example:

```
1.10 * (SALARY + BONUS) + SALARY / VAR3
          |        |        |       |
          2        1        4       3
```

Predicates

A *predicate* specifies a condition that is true, false, or unknown about a given object or group. The types of predicates are:

- basic
- quantified
- BETWEEN
- EXISTS
- IN
- LIKE
- NULL

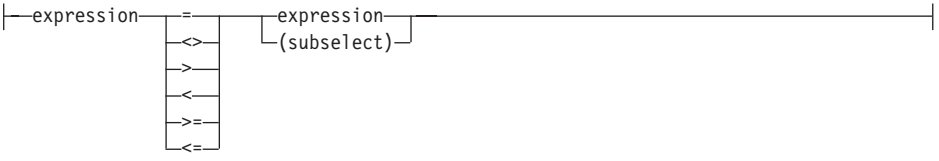
The following rules apply to predicates of any type:

- All values specified in a predicate must be compatible.
- Except for the first operand of LIKE, the operand of a predicate must not be a character string with a maximum length greater than 254.
- Except for EXISTS, a subselect in a predicate must specify a single expression.

Basic predicate

A basic predicate compares two values

Syntax statement



Remarks

A basic predicate compares two values. If the value of either operand is null, or the result of the subselect is empty; the result of the predicate is unknow. Otherwise, the result is either true or false.

A subselect in a basic predicate must not return more than one value, whether null or not null. This is also true for an expression of the form path-expression.

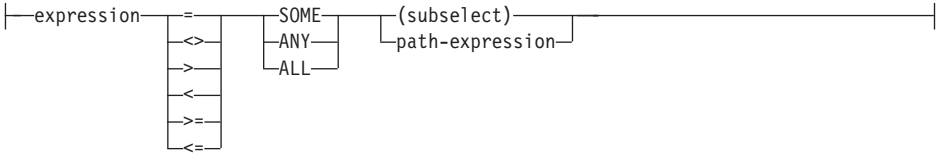
Examples

```
EMPNO = '123456'
SALARY < 20000
SALARY > (SELECT AVG(SALARY) FROM EMP)
```

Quantified predicate

A quantified predicate compares a value with the set of values produced by the subselect.

Syntax statement



Remarks

The quantified predicate compares a value with the set of values produced by the subselect. The subselect must specify a single result member. The subselect or path-expression can return any number of values, whether null or not null.

When ALL is specified, the result of the predicate is:

- True, if the result of the subselect is empty or if the specified relationship is true for every value returned by the subselect.
- False if the specified relationship is false for at least on value returned by the subselect.
- Unknown if the specified relationship is not false for any values returned by the subselect and at least one comparison is unknown because of a null value.

When SOME or ANY is specified, the result of the predicate is:

- True if the specified relationship is true for at least one value returned by the subselect.
- False if the result of the subselect is empty or if the specified relationship is false for every value returned by the subselect.

- Unknown if the specified relationship is not true for any of the values returned by the subselect and at least one comparison is unknown because of a null value.

Examples

Use the collections below when referring to the following examples. The collection COLA has 4 objects with member MEMA having values 1, 2, 3, and 4. The collection COLB has 2 object with members MEMB and MEMC. The member MEMB has values 2 and 3. The member MEMC has values 2 and NULL. In all examples, "object *n* of COLA" refers to the object in COLA for which MEMA has a value *n*.

```
COLA: MEMA    COLB:  MEMB  MEMC
        1              2     2
        2              3     -
        3
        4
```

- In the following predicate, the subselect returns the values 2 and 3. The predicate is false for objects 1, 2, and 3 of COLA, and is true for object 4.

  ```
  MEMA > ALL(SELECT MEMB FROM COLB)
  ```

- In the following predicate, the subselect returns the values 2 and 3. The predicate is false for objects 1 and 2 of COLA, and is true for objects 3 and 4.

  ```
  MEMA > ANY(SELECT MEMB FROM COLB)
  ```

- In the following predicate, the subselect returns the values 2 and null. The predicate is false for objects 1 and 2 of COLA, and is unknown for objects 2 and 4. The result is an empty collection.

  ```
  MEMA > ALL(SELECT MEMC FROM COLB)
  ```

- In the following predicate, the subselect returns the values 2 and null. The predicate is unknown for objects 1 and 2 of COLA, and is true for objects 3 and 4.

  ```
  MEMA > SOME(SELECT MEMC FROM COLB)
  ```

- In the following predicate, the subselect returns an empty result member. Therefore, the predicate is true for all objects of COLA.

  ```
  MEMA < ALL(SELECT MEMB FROM COLB WHERE MEMB>3)
  ```

- In the following predicate, the subselect returns an empty result member. Therefore, the predicate is false for all objects of COLA.

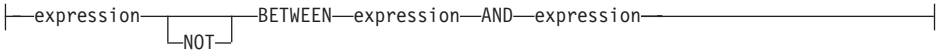
  ```
  MEMA < ANY(SELECT MEMB FROM COLB WHERE MEMB>3)
  ```

  If MEMA were null or more objects of COLA, the predicate would still be false for all objects of COLA.

BETWEEN predicate

The BETWEEN predicate determines whether a given value lies between two other given values specified in ascending order.

Syntax statement



Remarks

Each of the predicates forms has an equivalent search condition.

- The predicate:

  ```
  value1 BETWEEN value2 AND value3
  ```

  is equivalent to:

  ```
  value1 >= value2 AND value1 <= value3
  ```

- The predicate:

  ```
  value1 NOT BETWEEN value2 AND value3
  ```

  is equivalent to:

  ```
  NOT(value1 BETWEEN value2 AND value3
  ```

  and therefore also to:

  ```
  value1 < value2 OR value1 > value3
  ```

Examples

The predicate:

```
A BETWEEN B AND C
```

- Is true when B is 1, C is 3, and A is 1, 2, or 3.
- Is false when B is 1, C is 3, and A is 0 or 4.
- Is false when A is 0, B is 1, and C is null.
- Is false when A is 4, C is 3, and B is null.
- Is unknown when any of the following are true:
  - A is null.
  - A is 2, B is 1, and C is null.
  - A is 3, C is 4, and B is null.

EXISTS predicate

The EXISTS predicate tests for the existence of certain objects.

Syntax statement

Remarks

The result of the predicate is true if the result collection returned by the subselect or the path-expression contains at least one object; otherwise, false.

The SELECT clause in the subselect can specify any number of members, because the values returned by the subselect are ignored.

Unlike NULL, LIKE, and IN predicates, the EXISTS predicate has no form containing the word “NOT”. To negate an EXISTS predicate, precede it with the logical operator NOT; for example:

```
NOT EXISTS (subselect)
```

The result is false if the EXISTS predicate is true, and true if the predicate is false.

Examples

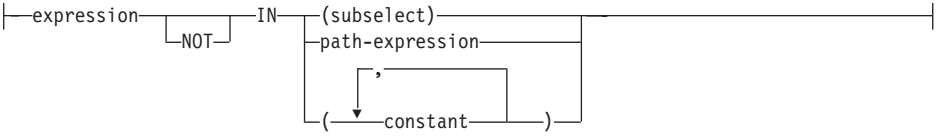
The following query lists the employee number of everyone represented in EMP who works in a department where at least one employee has a salary less than 20000. Like many EXISTS predicates, the one in this query involves a correlated variable.

```
SELECT EMPNO
  FROM EMP X
  WHERE EXISTS (SELECT * FROM EMP
                WHERE X.WORKDEPT=WORKDEPT AND SALARY<20000);
```

IN predicate

The IN predicate compares a value with a set of values.

Syntax statement



Remarks

In the subselect form, the subselect must specify a single result member. The subselect or the path-expression can return any number of values, whether null or not null. The IN predicate is equivalent to the quantified predicate as follows:

IN Predicate	Quantified Equivalent
expression IN (subselect)	expression = ANY (subselect)
expression NOT IN (subselect)	expression <> ALL (subselect)

In the non-subselect form of the IN predicate, the second operand is a set of one or more values specified by any combination of host variables, constants, or special registers. An In predicate of the form:

```
expression IN (value1, value2, ... valuen)
```

is logically equivalent to:

```
expression IN (SELECT * FROM R)
```

When T is a collection with a single object and R is a temporary collection formed by the following fullselect:

```
SELECT value1 FROM T
  UNION
SELECT value2 FROM T
  UNION
   ...
SELECT valuen FROM T
```

Each host variable must identify a structure or variable that is described in accordance with the rules for declaring host structures and variables.

Examples

The following predicate is true for any object whose employee is in department D11, B01, or C01.

```
WORKDEPT IN ('D11','B01','C01')
```

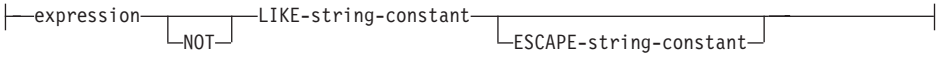
The following predicate is true for any object whose employee works in department E11.

```
EMPNO IN (SELECT EMPNO FROM EMP
WHERE WORKDEPT='E11')
```

LIKE predicate

The LIKE predicate searches for string that have a certain pattern.

Syntax statement



Remarks

The pattern is specified by a string in which the underscore and percent sign characters have special meanings.

Evaluating the predicate:

The first operand specifies the string to be tested. A parameter marker must

not be specified for or within the expression and the result of the expression must be a string. The second operand specifies the pattern. Let x denote the string to be tested and let y denote the pattern.

The following rules apply to predicates of the form `x LIKE y`. Predicates of the form `x NOT LIKE y` are equivalent to `NOT(x LIKE y)`.

- When x and y are both neither empty nor null, the result of the predicate is true if x matches the pattern in y and false if x does not match the pattern in y.
- When x or y is null, the result of the predicate is unknown.
- When y is empty and x is not, the result of the predicate is false.
- When x is empty and y is not, the result of the predicate is false unless y consists only of one or more percent signs.
- When x and y are both empty, the result of the predicate is true.

The pattern string

The pattern string and the string to be tested must be of the same type; that is, both x and y must be character strings. When x and y are character strings and x is not mixed data, a character is an SBCS character and y is interpreted as SBCS data regardless of its subtype. The pattern string can be specified as:

```
LIKE <string-constant>
```

includes the pattern as a string constant within the predicate.

The optional ESCAPE clause

Within the pattern, a percent sign or underscore can have a special meaning, or it can represent the literal occurrence of a percent sign or underscore. To have its literal meaning, it must be preceded by an escape character. If it is not preceded by an escape character, it has its special meaning.

The ESCAPE clause designates a single character. The character-and only that character-can be used multiple times within the pattern as an escape character. When the ESCAPE clause is omitted, no character serves as an escape character, so that percent signs and underscores in the pattern always have their special meaning.

The pattern

When the pattern does not include escape characters, a simple description of its meaning is:

- The underscore sign (_) represents a single arbitrary character.
- The percent sign (%) represents a string of zero or more arbitrary characters.
- Any other character represents a single occurrence of itself.

The more rigorous description follows:

- The string y is interpreted as a sequence of the minimum number of substring specifiers such that each character of y is part of exactly one

substring specifier. A substring specifier is an underscore, a percent sign, or any non-empty sequence of characters other than an underscore or percent sign.

- The string x matches the pattern y if there exists a partitioning of x into substrings such that:
  - A substring of x is a sequence of zero or more contiguous characters and each character of x is part of exactly one substring.
  - If the nth substring specifier is an underscore, the nth substring of x is a single character.
  - If the nth substring specifier is a percent sign, the nth substring of x is any sequence of zero or more characters.
  - If the nth substring specifier is neither an underscore nor a percent sign, the nth substring of x is equal to that substring specifier and has the same length as that substring specifier.
  - The number of substrings of x is thee same as the number of substring specifiers.
- When escape characters are present in the pattern string, an underscore, percent sign, or escape character represents a single occurrence of itself only if it is preceded by an odd number of successive escape characters.

Mixed data patterns

If x is mixed data, the pattern is assumed to be mixed data and its special characters are interpreted as follows:

- A single-byte underscore refers to one single-byte character; a double-byte underscore refers to one double-byte character.
- A percent sign, either single- or double-byte, refers to any number of characters of any type, either single- or double-byte.
- Redundant shift bytes in x or y are ignored.

Examples

The following predicate is true when the string to be tested in NAME had the value SMITH, NESSMITH, SMITHSON, or NESMITHY. It is not true when the string has the value SMYTHE:

```
NAME LIKE '%SMITH%'
```

Consider the predicate:

```
NAME LIKE 'AB+_C_%' ESCAPE '+'
```

Observer that in the pattern, the plus sign preceding the first underscore is an escape character. The predicate is true when the string being tested in NAME has the value AB_CD or AB_CDE. It is false when the string has the value AB, AB_, or AB_C.

The following two predicates are equivalent: three of the four percent signs in the first predicate are redundant:

```
NAME LIKE 'AB%%%%CD'
NAME LIKE 'AB%CB'
```

The following example illustrates the effect of successive occurrences of the escape character, which in this case is the plus (+) character.

When the pattern string is...	The pattern itself is...
+%	A percent sign.
++%	A plus sign followed by zero or more arbitrary characters.
+++%	A plus sign followed by a percent sign.

NULL predicate

The NULL predicate tests for null values.

Syntax statement

Remarks

The result of a NULL predicate cannot be unknown. If the value of the expression is null, the result is true. If the value is not null, the result if false. If NOT is specified, the result is reversed.

A parameter marker must not be specified for or within the expression.

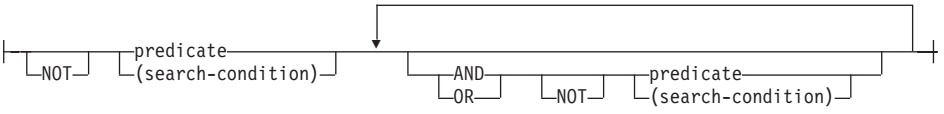
Examples

The following predicate is true whenever PHONENO has the null value; otherwise, false.

```
PHONENO IS NULL
```

Search

A *search condition* specifies a condition that is true, false, or unknown about a given object or group. When the condition is true, the object or group qualifies for the results. When the condition is false or unknown, the object or group does not qualify.



The result of a search condition is derived by application of the specified logical operator (AND, OR, NOT) to the result of each specified predicate. If logical operators are not specified, the result is the result of the specified predicate.

AND and OR are defined in the following table, in which P and Q are any predicates:

Table 6. Truth table for AND and OR

P	Q	P AND Q	P OR Q
True	True	True	True
True	False	False	True
True	Unknown	Unknown	True
False	True	False	True
False	False	False	False
False	Unknown	False	Unknown
Unknown	True	Unknown	True
Unknown	False	False	Unknown
Unknown	Unknown	Unknown	Unknown

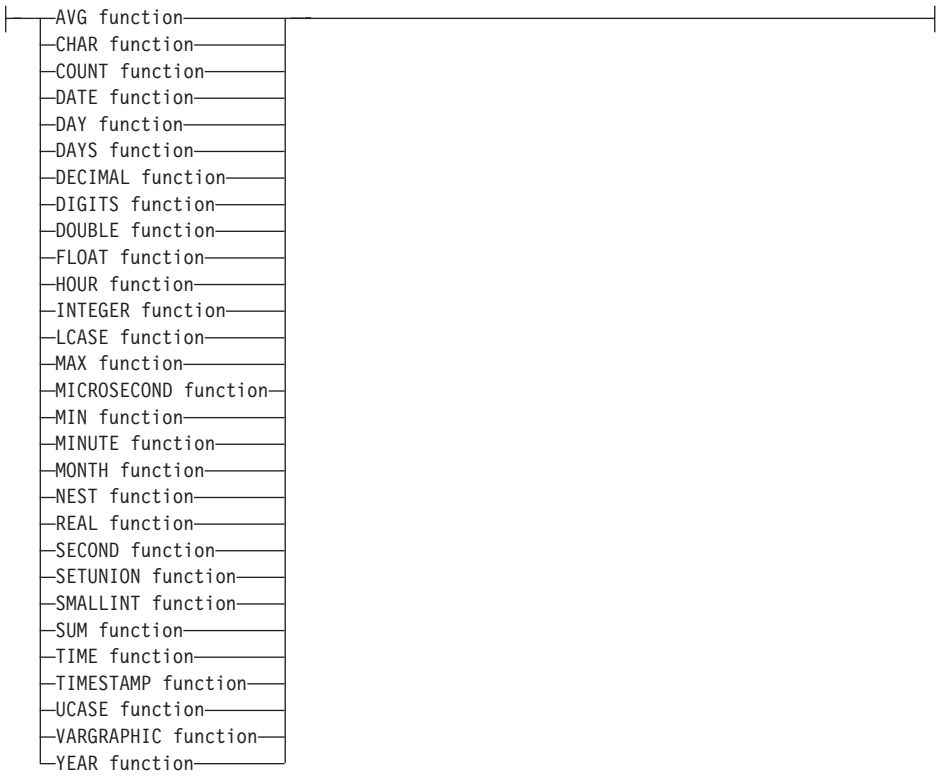
NOT(true) is false, NOT(false) is true, and NOT(unknown) is unknown.

Search conditions within parentheses are evaluated first. If the order of evaluation is not specified by parentheses, NOT is applied before AND, and AND is applied before OR. The order in which operators at the same precedence level are evaluated is undefined to allow for optimization of search conditions

Functions

A function is an operation denoted by a function name followed by one or more operands which are enclosed in parentheses. Most of the functions described in this section are scalar functions that take one or more scalar parameters and return a scalar result. Some of the functions, however, are aggregate functions which operate on a set of values or reduce it to a single value. The term column function is used in IBM documentation. The ANSI/ISO SQL92 Standard refers to this type of function as a set function. The term aggregate function is also used.

The operands of functions are called arguments. Most functions have a single argument that is specified by an expression. The result of a function is a single value derived by applying the function to the result of the expression.



In the syntax of OOSQL, the term function is used only in the definition of an expression. Thus, a function can be used only when an expression can be used.

The following scalar functions are supported:

- CHAR
- DATE
- DAY
- DAYS
- DECIMAL
- DIGITS
- DOUBLE
- FLOAT
- HOUR
- INTEGER
- LCASE
- MICROSECOND
- MINUTE
- MONTH
- REAL
- SECOND
- SMALLINT

- TIME
- TIMESTAMP
- UCASE
- VARGRAPHIC
- YEAR

The following aggregate functions are supported:
- AVG
- COUNT
- MAX
- MIN
- NEST
- SETUNION
- SUM

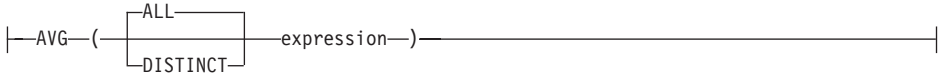
For Aggregate Functions:

1. This information applies to all aggregate functions except for the COUNT(*) variation of the COUNT function.
2. The argument of an aggregate function is a set of values derived from one or more members. The scope of the set is a group or an intermediate result collection.
3. The keyword DISTINCT is not considered an argument of the function but rather a specification of an operation that is performed before the function is applied. If DISTINCT is specified, duplicate values are eliminated. If ALL is implicitly or explicitly specified, duplicate values are not eliminated.
4. An expression in an aggregate function must include a member name and must not include an aggregate function. An aggregate function can be used in a WHERE clause only if the clause is part of a subquery of a HAVING clause and the member name specified in the expression is a correlated reference to a group. If the expression includes more than one member name, each member name must be a correlated to the same group.
5. The result of the COUNT function cannot be the null value. As specified in the description of AVG, MAX, MIN, and SUM, the result is the null value when the function is applied to an empty set.

AVG

The AVG function returns the average of a set of numbers.

Syntax statement



Remarks

The argument value must be numbers and their sum must be within the range of the data type of the result. The data type of the result is the same as the data type of the argument values, and the result is double-precision floating-point if the argument values are single-precision floating-point. The result can be null. If the data type of the argument values is decimal with precision *p* and scale *s*, the precision of the result is 38 and the scale is 38–*p*+*s*.

The function is applied to the set of values derived from the argument values by the elimination of null values. If DISTINCT is specified, duplicate values are also eliminated.

If the function is applied to an empty set, the result is a null value. Otherwise, the result is the average value of the set. If the type of the result is INTEGER the fractional part of the average is lost. The order in which the summation part of the operation is performed is undefined but every intermediate result must be within the range of the result data type.

Examples

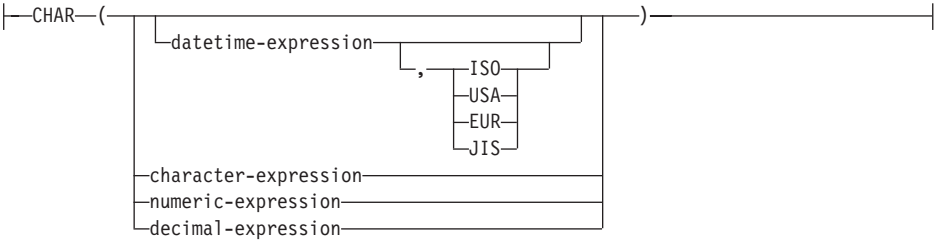
Select the average salary in department D11 of the employees in the sample collection EMP.

```
SELECT AVG(SALARY)
  FROM EMP
  WHERE WORKDEPT='D11';
```

CHAR

Returns a character string representation of a date/time, character, or numeric.

Syntax statement



Parameters

datetime-expression

An expression that is one of the following data types:

date The result is the character string representation of the date in the

format specified by the second argument. An error occurs if the second argument is specified and is not a valid value.

time The result is the character string representation of the time in the format specified by the second argument. An error occurs if the second argument is specified and is not a valid value.

timestamp
The second argument is not applicable and must not be specified. The result is the character representation of the timestamp.

character-expression
An expression that returns a value that is a string.

numeric-expression
An expression that returns a value that is an integer data type (either SMALLINT or INTEGER).

Note: REAL, FLOAT, or DOUBLE numeric expression result in a character string expression in the form of an SQL double constant.

decimal-expression
An expression that returns a value that is a decimal data type. If a different precision and scale is desired, the DECIMAL scalar function can be used first to make the change.

The result is the fixed-length character-string representation of the argument. The result includes *p* digits, where *p* is precision of the **decimal-expression** with a preceding minus sign if the argument is negative. The length of the result is 2+*p*, where *p* is the precision of the decimal-expression. This means that a positive value will always include one trailing blank.

Remarks

The CHAR function returns a character string representation of a:
- Date/Time value if the first argument is a date, time, or timestamp.
- Character string value if the first argument is anytype of character string.
- Number if the first argument is a numeric value.

The result of the function is a character string. If the first argument can be null, the result can be null. If the first argument is null, the result is the null value.

The result of the character string representation of the argument in the form of an SQL integer constant. The result consists of n characters that are the significant digits that represent the value of the argument with a preceding minus sign if the argument is negative. It is left justified.

Examples

Assume that the attribute PRSTDATE has an internal value equivalent to 1988-12-25:

```
CHAR (PRSTDATE, USA)
```

results in the value 12/25/1998.

Assume that the attribute STARTING has an internal value equivalent to 17.12.30:

```
CHAR (STARTING, USA)
```

results in the value 5:12 PM.

Assume that the attribute RECEIVED(timestamp) has an internal value equivalent to the combination of the PRSTDATE and STARTING attributes:

```
CHAR (RECEIVED)
```

results in the value 1988-12-25-17.12.30.000000.

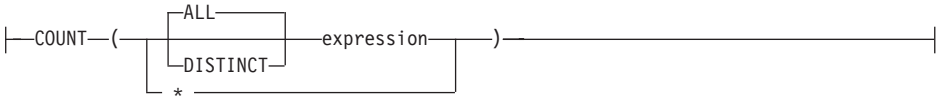
Use the CHAR function to return the values for EDLEVEL (Defined as smallint) as a character string:

```
SELECT CHAR (EDLEVEL) FROM EMPLOYEE
```

COUNT

Returns the number of objects or values in a set of objects or values.

Syntax statement



Remarks

The COUNT function returns the number of objects or values in a set of objects or values. The result of the function must be within the range of integers and cannot be null. The data type of the result is INTEGER.

The argument COUNT(*) is a set of objects. The result is the number of objects in the set. Any object that includes only null values is included in the count.

The argument COUNT(DISTINCT *expression*) is a set of values. The function is applied to the set of values derived from the argument values by the elimination of null values and duplicate values. The result is the number of values in the set.

Examples

Select the number of females represented in the sample collection EMP.

```
SELECT COUNT(*)
  FROM EMP
  WHERE SEX = 'F';
```

Select the number of departments that have at least one female as a member.

```
SELECT COUNT(DISTINCT WORKDEPT)
  FROM EMP
  WHERE SEX = 'F';
```

DATE

Returns a date from a value.

Syntax statement

Remarks

The DATE function retunes a date from a value. The argument must be a date, timestamp, a positive number less than or equal to 3,652,059, or valid character string representation of a date. (If the argument is a character string of length 7, it must represent a valid date in the form yyynnnn, where yyyy are digits denoting a year, and nnn are digits between 0001 and 366, denoting a day of that year.)

The result of the function is a date. If the argument is null, the result is the null value.

The other rules depend on the data type of the argument.

- If the argument is a date, timestamp, or valid string representation of a date, the result is the date part of the value.
- If the argument is a number, the result is the date that is n=1 days after January 1, 0001, where n is the integral part of the number.
- If the argument is a character string with a length of 7, the result is the date represented by the character string.

Examples

Both of the following statements result in an internal representation of `1988-12-25`.

```
DATE('1988-12-25')
DATE('25.12.1988;)
```

The following statement results in an internal representation of `0001-02-04`.

```
DATE(35)
```

DAY

Returns the day part of a value.

Syntax statement

Remarks

The DAY function returns the day part of a value. The argument must be a date, timestamp, date duration, or timestamp duration. The result of the function is a large integer. If the argument is null, the result is the null value.

The other rules depend on the data type of the argument.

- If the argument is a date or timestamp, the result is the day part of the value, which is an integer between 1 and 31.
- If the argument is a date duration or timestamp duration, the result is the day part of the value, which is an integer between —99 and 99. A non-zero result has the same sign as the argument.

Examples

Using the PROJECT collection, select the day that WELD LINE PLANNING project(PROJNAME) is scheduled to stop(PRENDATE).

```
SELECT DAY(PRENDATE)
FROM PROJECT
WHERE PROJNAME = 'WELD LINE PLANNING';
```

Assume that the attribute DATE1 has an internal value equivalent to 2000-03-15 and that the attribute DATE2 has an internal value equivalent to 1999-12-31.

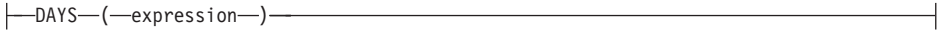
```
DAY(DATE1-DATE2)
```

results in the value 15.

DAYS

Returns an integer representation of a date.

Syntax statement



Remarks

The DAYS function returns an integer representation of a date. The argument must be a date. The result of the function is a integer. If the argument is null, the result is the null value.

The result is 1 more than the number of days from January 1, 0001 to D, where D is the date that would occur if the DATE function were applied to the argument.

Examples

Using the PROJECT collection, select the number of elapsed days(PRENDATE-PRSTDATE) estimated for the project(PROJNO)'IF2000'.

```
SELECT DAYS(PRENDATE) - DAYS(PRSTDATE)
  FROM PROJECT
  WHERE PROJNO = 'IF2000';
```

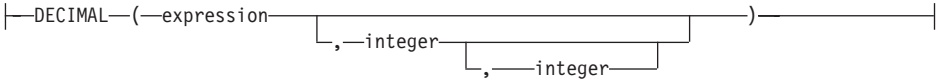
Using the PROJECT collection, select the sum of elapsed days (PRENDATE-PRSTDATE) estimated for all projects in department (DEPTNO)'E21'.

```
SELECT DAYS(PRENDATE) - DAYS(PRSTDATE)
  FROM PROJECT
  WHERE DEPTNO = 'E21';
```

DECIMAL

Returns a decimal representation of a value.

Syntax statement



Remarks

The DECIMAL function returns a decimal representation of a value. The first argument must be a character string or a number. The second argument, if specified, must range in value from 1 to 38. The third argument, if specified, must range in value from 1 to *p*, where *p* is the value of the second argument.

If the second argument is omitted, its value depends on the data type of the first argument. The default is:

- 5 if the first argument is a small integer
- 11 if the first argument is an integer
- 15 in all other cases

If the third argument is omitted, its value is zero.

If the first argument is a string, its length attribute must not exceed 2048. Leading and trailing blanks are eliminated from the string. The resulting substring must conform to the rules for forming an OOSQL integer or decimal constant.

The data type of the result is DECIMAL(*p*,*s*), where *p* and *s* are the second and third arguments. If the first argument is null, the result is null.

Assume that N denotes the number or numeric constant specified as the first argument. The result of the function is the number that would occur if N were assigned to a decimal column with precision *p* and scale *s*. Accordingly, an error occurs if the number of significant digits required to represent the whole part of the number is greater than *p-s*.

Examples

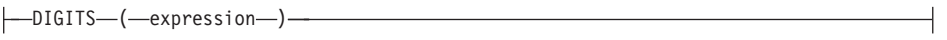
Represent the average salary of the employees in EMPLOYEE collection as an 8-digit decimal number with two of these digits to the right of the decimal point.

```
SELECT DECIMAL(AVG(SALARY),8,2)
  FROM EMPLOYEE;
```

DIGITS

Returns a character-string representation of a number.

Syntax statement



Remarks

The DIGITS function returns a character-representation of a number. The argument must be an expression that returns a value of type SMALLINT, INTEGER, or DECIMAL. If the argument is null, the result is the null value.

The result of the function is a fixed-length character string representing the absolute value of the argument without regard to its scale. The result does not include a sign or a decimal character. Instead, it consists exclusively of digits, including, if necessary, leading zeros to fill out the string. The length of the string is:

- 5 if the argument is a small integer.
- 10 if the argument is a large integer.
- *p* if the argument is a decimal number with a precision of *p*

Examples

Assume that TABLEX collection contains the INTCOL attribute that containing 10-digit numbers.

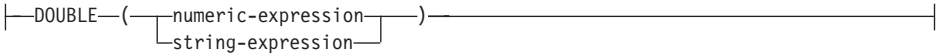
```
SELECT DIGITS(INTCOL)
  FROM TABLEX;
```

returns a 10 character result string for each entry in TABLEX.

DOUBLE

Returns a floating-point representation of a number.

Syntax statement



Remarks

The DOUBLE function returns a floating-point representation of a number. FLOAT can be used as a synonym for DOUBLE. The argument is an expression that returns a value of any built-in numeric data type. If the character-expression argument contains a valid representation of a floating point value, this value is returned. The result of the function is a double-precision floating-point number. If the argument is null, the result is the null value.

The result is the same number that would occur if the argument were assigned to a double-precision floating-point column or variable.

Examples

Using the EMPLOYEE collection, find the ratio of salary to commission for employees whose commission is not zero. The attributes involved (SALARY and COMM) have integer data types. DOUBLE is applied to SALARY so that the division is carried out in floating point:

```
SELECT EMPNO, DOUBLE(SALARY)/COMM
  FROM EMPLOYEE
  WHERE COMM > 0;
```

FLOAT

Returns a floating-point representation of a number.

Syntax statement

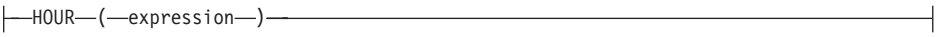
Remarks

The FLOAT function returns a floating-point representation of a number. FLOAT is a synonym for DOUBLE.

HOUR

The HOUR function returns the hour part of a value.

Syntax statement



Remarks

The HOUR function returns the hour part of a value. The argument must be a time, timestamp, time duration, or timestamp duration. The result of the function is a large integer. If the argument is null, the result is the null value.

The other rules depend on the data type of the argument:

- If the argument is a time or timestamp, the result is the hour part of the value, which is an integer between 0 and 24.
- If the argument is a time duration or timestamp duration, the result is the hour part of the value, which is an integer between -99 and 99. A nonzero result has the same sign as the argument.

Examples

Using the CL_SCHED sample collection, select all the classes that start in the afternoon.

```
SELECT * FROM CL_SCHED
  WHERE HOUR(STARTING) BETWEEN 12 AND 17;
```

INTEGER

The INTEGER function returns an integer representation of a number or character string in the form of an integer constant.

Syntax statement

Parameters

numeric-expression

An expression that returns a value of any built-in numeric data type.

If the argument is a numeric-expression, the result is the same number that would occur if the argument were assigned to a large integer attribute or

variable. If the whole part of the argument is not within the range of integers. an error occurs. The decimal part of the argument is truncated if present.

character-expression
An expression that returns a character string value of length not greater than the maximum length of a character constant. Leading blanks are eliminated and the resulting string must conform to the rules for forming an SQL integer constant.

If the argument is a character-expression, the result is the same number that would occur if the corresponding integer constant were assigned to a large integer attribute or variable.

Remarks

The INTEGER function returns an integer representation of a number or character string in the form of an integer constant. SMALLINT can be used as a synonym, but returns a small integer.

The result of the function is an integer. If the argument is null, the result is the null value.

Examples

- From the EMPLOYEE collection, select a list containing salary (SALARY) divided by education level (EDLEVEL). Truncate any decimal in the calculation. The list should also contain the value used in the calculation and employee number (EMPNO). The list should be in descending order of the calculated value.

```
SELECT INTEGER (SALARY / EDLEVEL), SALARY, EDLEVEL, EMPNO
  FROM EMPLOYEE
  ORDER BY 1 DESC;
```

- From the EMPLOYEE collection, select the EMPNO attribute in integer form:

```
SELECT INTEGER(EMPNO) FROM EMPLOYEE;
```

LCASE

The scalar function LCASE converts all upper case characters in *string-expression* to lower case.

Syntax statement

Remarks

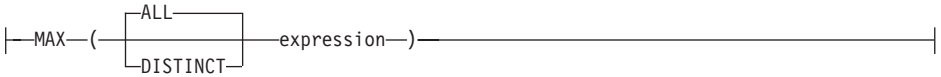
The scalar function LCASE converts all upper case characters in *string-expression* to lower case.

The LOWER function can be substituted for LCASE.

MAX

The MAX function returns the maximum value in a set of values.

Syntax statement



Remarks

The MAX function returns the maximum value in a set of values. The data type of the result and its other attributes (for example, the length) are the same as the data type and attributes of the argument values. The result can be null.

The function is applied to the set of values derived from the argument values by the elimination of null values.

If the function is applied to an empty set, the result is a null value. Otherwise, the result is the maximum value in the set.

The specification of DISTINCT has no effect on the result and is not advised.

Examples

- Select the maximum monthly salary of the employees represented in the sample collection EMP.

```
SELECT MAX(SALARY) / 12

FROM EMP;
```

- Select the surname that comes last in the collating sequence for the employees represented in the sample collection EMP.

```
SELECT MAX(LASTNAME)
  FROM EMP;
```

MICROSECOND

The MICROSECOND function returns the microsecond part of a value.

Syntax statement

Remarks

The MICROSECOND function returns the microsecond part of a value. The argument must be a timestamp or timestamp duration. The result of the function is a large integer. If the argument is null, the result is the null value.

The other rules depend on the data type of the argument:

- If the argument is a timestamp, the integer ranges from 0 through 999999.
- If the argument is a duration, the result reflects the microsecond part of the value, which is an integer between -999999 and 999999. A nonzero result has the same sign as the argument.

Examples

Assume that collection TABLEA contains two attributes, TS1 and TS2, of type TIMESTAMP. Select all attributes in which the microseconds portion of TS1 is not zero and the seconds portion of TS1 and TS2 are identical:

```
SELECT * FROM TABLE1
  WHERE MICROSECOND(TS1) <> 0 AND
  SECOND(TS1) = SECOND(TS2);
```

MIN

The MIN function returns the minimum value in a set of values.

Syntax statement

Remarks

The MIN function returns the minimum value in a set of values. The data type of the result and its other attributes are the same as the data type and attributes of the argument values. The result can be null.

The function is applied to the set of values derived from the argument values by the elimination of null values.

If the function is applied to an empty set, the result is a null value. Otherwise, the result is the minimum value in the set.

The specification of DISTINCT has no effect on the result and is not advised.

Examples

- Select the minimum monthly salary of the employees represented in the sample collection EMP.

```
SELECT MIN(SALARY) / 12
  FROM EMP;
```

- Select the surname that comes first in the collating sequence for the employees represented in the sample collection EMP.

```
SELECT MIN(LASTNAME)
  FROM EMP;
```

MINUTE

The MINUTE function returns the minute part of a value.

Syntax statement

Remarks

The MINUTE function returns the minute part of a value. The argument must be a time, timestamp, time duration, or timestamp duration. The result of the function is an integer. If the argument is null, the result is the null value.

The other rules depend on the data type of the argument:

- If the argument is a time or timestamp, the result is the minute part of the value, which is an integer between 0 and 59.
- If the argument is a time duration or timestamp duration, the result is the minute part of the value, which is an integer between -99 and 99. A nonzero result has the same sign as the argument.

Examples

Using the CL_SCHED sample collection, select all classes with a duration less than 50 minutes:

```
SELECT * FROM CL_SCHED
  WHERE HOUR(ENDING-STARTING) = 0 AND
  MINUTE(ENDING-STARTING) < 50;
```

MONTH

The MONTH function returns the month part of a value.

Syntax statement

Remarks

The MONTH function returns the month part of a value. The argument must be a date, timestamp, date duration, or timestamp duration. The result of the function is a large integer. If the argument is null, the result is the null value.

The other rules depend on the data type of the argument:

- If the argument is a date or timestamp, the result is the month part of the value, which is an integer between 1 and 12.
- If the argument is a date duration or timestamp duration, the result is the month part of the value, which is an integer between -99 and 99. A nonzero result has the same sign as the argument.

Examples

Select from the EMPLOYEE collection, people who were born (BIRTHDATE) in December:

```
SELECT * FROM EMPLOYEE
  WHERE MONTH(BIRTHDATE) = 12;
```

NEST

The NEST function returns a collection of pointers to objects of type specified by the parameter *pointer-type*.

Syntax statement

Remarks

The NEST function returns a collection of pointers to objects of type specified by the parameter *pointer-type*. The function returns an empty collection when the parameter is empty. The argument value is of pointer type. The result collections fabricated by NEST are temporary collections that are automatically dropped when the object that contains them is deleted.

The function is applied to the set of values derived from the argument values by the elimination of null values. The result is the number of values in the set.

The NEST(*pointer-type*) is a synonym for SETUNION(SET(*pointer-type*.OID)).

Examples

Select the distinct employee codes and a collection of pointers to employees having the employee code in Emp.

```
SELECT e.emp_code, NEST (e)
  FROM Emp e
  GROUP BY e.emp_code;
```

The above query is the same as the following query.

```
SELECT e.emp_code, SETUNION(SET(e.OID))
  FROM Emp e
  GROUP BY e.emp_code;
```

REAL

The scalar function REAL returns a single precision floating point representation of *numeric-exp.*

Syntax statement

SECOND

The SECOND function returns the seconds part of a value.

Syntax statement

Remarks

The SECOND function returns the seconds part of a value. The argument must be a time, timestamp, time duration, or timestamp duration. The result of the function is an integer. If the argument is null, the result is the null value.

The other rules depend on the data type of the argument:

- If the argument is a time or timestamp, the result is the seconds part of the value, which is an integer between 0 and 59.
- If the argument is a time duration or timestamp duration, the result is the seconds part of the value, which is an integer between -99 and 99. A nonzero result has the same sign as the argument.

Examples

Assume that the attribute RECEIVED(timestamp) has an internal value equivalent to 1988-12-25-17.12.30.000000:

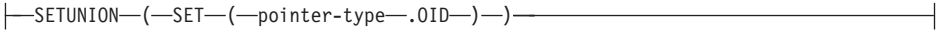
```
SECOND(RECEIVED)
```

returns the value 30.

SETUNION

The SETUNION function returns a collection of pointers to objects of type specified by the parameter pointer-type.

Syntax statement



Note: 390 SOLARIS OS/390 and Solaris do not support the SETUNION function. Use the NEST function instead.

Remarks

The SETUNION function returns a collection of pointers to objects of type specified by the parameter pointer-type. The function returns an empty collection when the parameter is empty. The argument value is of pointer type. The result collections fabricated by SETUNION are temporary collections that are automatically dropped when the object that contains them is deleted.

The function is applied to the set of values derived from the argument values by the elimination of null values. The result is the number of values in the set.

Examples

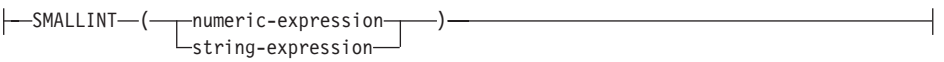
Select the distinct employee codes and a collection of pointers to employees having the employee code in Emp.

```
SELECT e.emp_code, SETUNION(SET(e.OID))
  FROM Emp e
  GROUP BY e.emp_code;
```

SMALLINT

The scalar function SMALLINT converts the argument to a small integer, by truncating of the decimal part if necessary. If the argument is of string data type, it must be a character-string representation of an integer between -32,768 and +32,767.

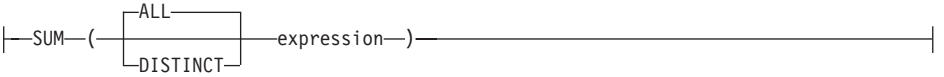
Syntax statement



SUM

The SUM function returns the sum of a set of numbers.

Syntax statement



Remarks

The SUM function returns the sum of a set of numbers. The argument values must be numbers and their sum must be within the range of the data type of the result. The data type of the result is the same as the data type of the argument values, except that the result is double precision floating-point if the argument values are single precision floating-point. The result can be null. If the data type of the argument values is decimal, the precision of the result is 38 and the scale is the same as the scale of the argument values.

The function is applied to the set of values derived from the argument values by the elimination of null values. If DISTINCT is specified, duplicate values are also eliminated.

If the function is applied to an empty set, the result is a null value. Otherwise, the result is the sum of the values in the set. The order in which the summation is performed is undefined but every intermediate result must be within the range of the result data type.

Examples

Select the total income from all sources (salaries, commissions, and bonuses) of the employees represented in sample collection EMP.

```
SELECT SUM(SALARY+COMM+BONUS)
  FROM EMP
```

TIME

The TIME function returns a time from a value.

Syntax statement

Remarks

The TIME function returns a time from a value. The argument must be a time or timestamp. The result of the function is a time. If the argument is null, the result is the null value.

The other rules depend on the data type of the argument:

- If the argument is a time, the result is that time.
- If the argument is a timestamp, the result is the time part of the timestamp.
- If the argument is a character string, the result is the time represented by the character string.

Examples

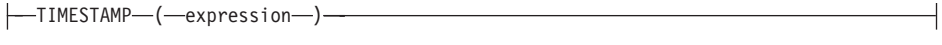
Select all notes from the IN_TRAY collection that were received later in the day (any day) than 12:30:

```
SELECT * FROM IN_TRAY
  WHERE TIME(RECEIVED) >= TIME('12.30.00');
```

TIMESTAMP

The TIMESTAMP function returns a timestamp from a value.

Syntax statement



Remarks

The TIMESTAMP function returns a timestamp from a value. The argument must be a timestamp, a valid character string representation of a timestamp, or a character string of length 14. (A character string of length 14 must be a string of digits that represents a valid date and time in the form yyyyxxddhhmmss, where yyyy is the year, xx is the month, dd is the day, hh is the hour, mm is the minute, and ss is the seconds.) The result of the function is a timestamp. If either argument is null, the result is the null value.

The other rules depend on the data type of the argument:

- If the argument is a timestamp, the result is that timestamp.

- If the argument is a character string, the result is the timestamp represented by that character string. If the argument is a character string of length 14, the timestamp has a microsecond part of zero.

Examples

Assume that the attribute START_TIMESTAMP has a value equivalent to 1988-12-25.17.12.30:

```
TIMESTAMP(START_TIMESTAMP)
```

returns the value '1988-12-25-17.12.30.000000'.

UCASE

The scalar function UCASE converts all lower case characters in *string-exp* to upper case.

Syntax statement

Remarks

The scalar function UCASE converts all lower case characters in *string-exp* to upper case.

The UPPER function can be substituted for UCASE.

VARGRAPHIC

The vargraphic function returns a graphic string representation of a character string.

Syntax statement

Remarks

The result of the function is a varying-length graphic string. If the argument is null, the result is the null value. If the argument is a emply string, the result is an empty string.

The length attribute of the result is equal to the length attribute of the argument.

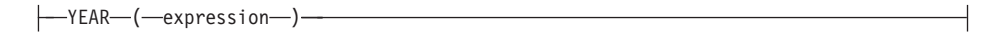
The query engine uses the portable NLS function *mbstowcs* to convert character string arguments into vargraphics string results.

Note: The input to the function can be either a string of single byte character or multibyte characters, and the result is always a string of wide characters.

YEAR

The YEAR function returns the year part of a value.

Syntax statement



Remarks

The YEAR function returns the year part of a value. The argument must be a date, timestamp, date duration, or timestamp duration. The result of the function is an integer. If the argument is null, the result is the null value.

The other rules depend on the data type of the argument:

- If the argument is a date or timestamp, the result is the year part of the value, which is an integer between 1 and 9999.
- If the argument is a date duration or timestamp duration, the result is the year part of the value, which is an integer between -9999 and 9999. A nonzero result has the same sign as the argument.

Examples

- Select all the projects in the PROJECT collection that are scheduled to start (PRSTDATE) and end (PRENDATE) in the same calendar year:

```
SELECT * FROM PROJECT
  WHERE YEAR(PRSTDATE) = YEAR(PRENDATE);
```

- Select all the projects in the PROJECT collection that are scheduled to take less than one year to complete:

```
SELECT * FROM PROJECT
  WHERE YEAR(PRENDATE - PRSTDATE) < 1;
```

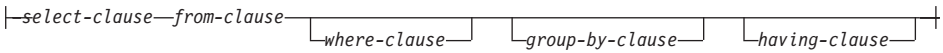
Queries

A *query* specifies a result collection. A query is a component of certain OOSQL statements. The forms of query are:

- subselect
- fullselect
- select-statement

A subselect is a subset of a fullselect. A fullselect is a subset of a select-statement.

subselect:



The *subselect* is a component of the fullselect statement. It is also a component of certain predicates which, in turn, are components of a subselect. A subselect that is a component of a predicated is called a *subquery*.

The subselect specifies a result collection derived from the result of its first FROM clause. The derivation can be described as a sequence of operations in which the result of each operation is input for the next. (This is only a way of describing the subselect. The method used to perform the derivation may be quite different from this description.)

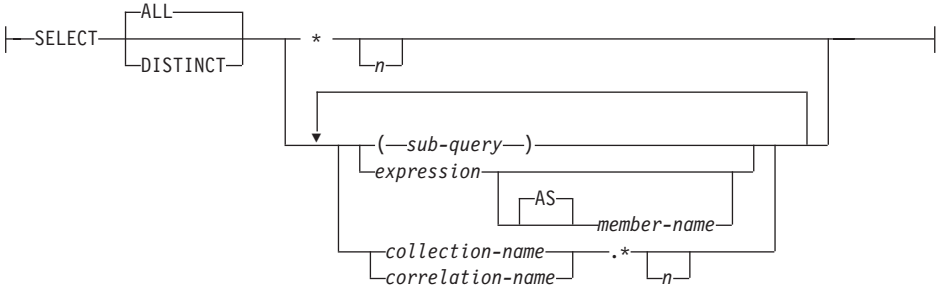
The sequence of the (hypothetical) operations is:

1. FROM clause
2. WHERE clause
3. GROUP BY clause
4. HAVING clause
5. SELECT clause

SELECT clause

The SELECT clause specifies the members of the final result collection.

Syntax statement



Parameters

ALL Retains all objects of the final result collection and does not eliminate duplicates. This is the default.

DISTINCT
Eliminates all but one of each set of duplicate objects of the final result

collection. DISTINCT must not be used more than once in a subselect. This restriction includes SELECT DISTINCT and the use of DISTINCT in an aggregate function of the select list or HAVING clause. It does not include occurrences of DISTINCT in subqueries of the subselect.

Two objects are duplicates of one another only if each value in the first object is equal to the corresponding value in the second object. For determining duplicate objects, two null values are considered equal.

***** Represents a list of names that identify the members of collection R. The first name in the list identifies the first member of R, the second name identifies the second member of R, and so on.

n Represents the depth that applies to the members that are of pointer and structure types. It does not apply to members that are of collection types.

*n represents a list of names that identify the members of collection R by following the pointer and structure types to a depth of n. The *1 is the same as *.

expression
Can be any expression of the type described in "Expressions" on page 813. Each *member-name* in the expression must unambiguously identify a member of R. The member name can be inherited from other classes. When a class inherits from multiple classes, OOSQL uses the left path precedence inheritance rule, that is, the member corresponds to the member name of the class that is on the left most path.

member-name
The member-name renames the result column. The name must not be qualified and does not have to be unique.

name.* Represents a list of names that identify the members of *name. name* can be a collection name or correlation name, and must designate a collection named in the FROM clause. The first name in the list identifies the first member of the collection, the second name in the list identifies the second member of the collection, and so on.

subquery
Represents a subquery that has a single element in its projection clause and must return at most one value. Subqueries in the projection clause cannot be embedded.

Remarks

The SELECT clause specifies the members of the final result collection. The member values are produced by the application of the select list to R. The select list is a list of names and expressions specified in the SELECT clause, and R is the result of the previous operation of the subselect. For example, if the only clauses specified are SELECT, FROM, and WHERE, then R is the result of that WHERE clause.

The number of members in the result of SELECT is the same as the number of expressions in the operational form of the select list (that is, the list established at the

time the statement is prepared). The result of a subquery must be a single member, unless the subquery is used in an EXISTS predicate.

Applying the select list

Some of the results of applying the select list to R depend on whether GROUP BY or HAVING is used.

If neither GROUP BY nor HAVING is used:

- The select list must not include aggregate functions, or it must be entirely a list of aggregate functions.
- If the select list does not include aggregate functions, it is applied to each object of R and the result contains as many objects as there are objects in R.
- If the select list is a list of aggregate functions, R is the source of the arguments of the functions and the result of applying the select list is one object, even when R consists of zero objects.

If GROUP BY or HAVING is used:

- Each *member-name* in the select list must either identify a grouping member or be specified within an aggregate function.
- The select list is applied to each group of R, and the result contains as many objects as there are groups in R. When the select list is applied to a group of R, that group is the source of the arguments of the aggregate functions in the select list.

In either case the *n*th member of the result contains the values specified by applying the nth expression in the operational form of the select list.

Null attributes of result members

Result members allow null values if they are derived from one of the following:

- Any aggregate function but COUNT
- A member that allows null values
- An arithmetic expression that allows nulls
- A scalar function or string expression that allows null values
- A result of a UNION if at least one of the corresponding items in the select list is nullable

Names of result members

The name of a result member of a subselect is determined as follows:

- If the result member is derived from a member name, the result member name is the unqualified name of that member.
- All other result members are named 1,2,3,.. depending on the position of the members.

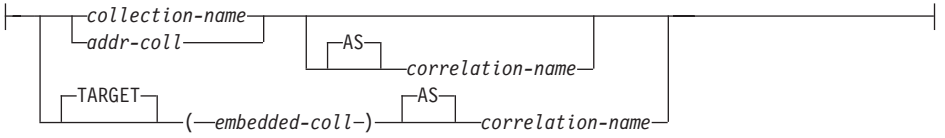
FROM clause

Specifies an intermediate result collection.

Syntax statement

FROM clause specifies an intermediate result collection, R. If a single *collection-spec* is specified, R is the result of that *collection-spec*. If more than one *collection-spec* is specified, R consists of all possible combinations of the objects of the result of each *collection-spec*. Each object of R is a object from the result of the first *collection-spec* concatenated with a object from the result of the second *collection-spec*, concatenated with a object from the result of the third *collection-spec*, and so on. The number of objects in R is the product of the number of objects in the result of each *collection-spec*. Thus, if the result of any *collection-spec* is empty, R is empty.

collection-spec:



Note: 390 SOLARIS OS/390 and Solaris do not support the TARGET.. AS syntax.

Remarks

A *collection-spec* specifies an intermediate result collection. If a single collection is identified, the intermediate result collection is simply that collection.

Each *collection-name* specified in every FROM clause of the same OOSQL statement must identify an existing collection.

Each *correlation-name* is defined as a designator of the intermediate result collection specified by the immediately preceding *collection-spec*.

The *embedded-coll* specifies an embedded collection. Parentheses must be used to declare correlation names for embedded collections. The embedded collection should be preceded by an optional keyword TARGET. Correlation names for embedded collections can be dependent upon other correlation names and can create an ordering in the declaration of correlation names. The only restriction is that if q2 is dependent upon q1, then the declaration of q1 must precede that of q2. Correlation name q2 is dependent upon q1 if q1 has a member that is the embedded collection for which q2 is defined.

The embedded collection for which a correlation name is defined can be part of a path of any depth, possibly containing collection members and methods. The only restriction is that the leaf node of the path has to be a collection member.

An exposed name is a correlation name or a name that is not followed by a correlation name. The exposed names in a FROM clause should be unique, and only exposed names should be used as qualifiers of member names. Thus, if the same collection name is specified twice, at least one specification of the collection name should be followed by a unique correlation name. That correlation name should be used to qualify references to members of that instance of the collection. For more information, see “Member name qualifiers in correlated references” on page 812.

WHERE clause

The WHERE clause specifies an intermediate result collection that consists of those objects of R for which the search condition is true.

Syntax statement

Remarks

The WHERE clause specifies an intermediate result collection that consists of those objects of R for which the search condition is true. R is the result of the FROM clause of the subselect.

The search condition must conform to the following rules:

- Each member name must unambiguously identify a member of R or be a correlated reference. A member name is a correlated reference if it identifies a member of a collection identified in an outer subselect.
- An aggregate function must not be specified unless the WHERE clause is specified in a subquery of a HAVING clause and the argument of the function is a correlated reference to a group.

Any subquery in the *search-condition* is effectively executed for each object of R and the results are used in the application of the *search-condition* to the given object of R. A subquery is actually executed for each object of R only if it includes a correlated reference. In fact, a subquery with no correlated references is executed just once, whereas a subquery with a correlated reference may have to be executed once for each object.

GROUP BY clause

The GROUP BY clause specifies an intermediate result collection that consists of a grouping of the objects of R.

Syntax statement

Remarks

The GROUP BY clause specifies an intermediate result collection that consists of a grouping of the objects of R. R is the result of the previous clause.

Each *member-name* must unambiguously identify a member of R other than a long string member. Each identified member is called a *grouping member*.

The result of GROUP BY is a set of groups of objects. In each group of more than one object, all values of each grouping member are equal; and all objects with the same set of values of the grouping members are in the same group. For grouping, all null values within a grouping member are considered equal.

Because every object of a group contains the same value of any grouping member, the name of a grouping member can be used in a search condition in a HAVING clause or an expression in a SELECT clause. In each case, the reference specifies only one value for each group.

HAVING clause

The HAVING clause specifies an intermediate result collection that consists of those groups of R for which the search-condition is true.

Syntax statement

Remarks

The HAVING clause specifies an intermediate result collection that consists of those groups of R for which the search-condition is true. R is the result of the previous clause. If this clause is not GROUP BY, R is considered a single group with no grouping members.

Each *member-name* in a *search-condition* must:

- Unambiguously identify a grouping member of R, or
- Be specified within an aggregate function, or
- Be a correlated reference. A member-name is a correlated reference if it identifies a member of a collection identified in an outer subselect.

A group of R to which the search condition is applied supplies the argument for each function in the search condition, except for any function whose argument is a correlated reference.

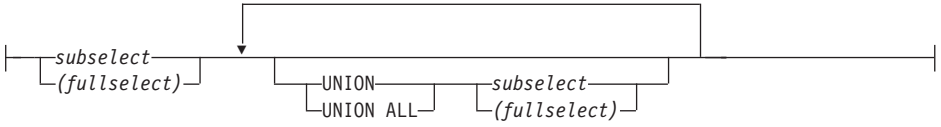
If the search condition contains a subquery, the subquery can be thought of as being executed each time the search condition is applied to a group of R, and the results used in applying the search condition. In actuality, the subquery is executed for each group only if it contains a correlated reference.

A correlated reference to a group of R must either identify a grouping member or be contained within an aggregate function.

fullselect

fullselect specifies a result collection.

Syntax statement



Remarks

UNION or UNION ALL

Derives a result collection by combining two other result collections, R1 and R2. If UNION ALL is specified, the result consists of all objects in R1 and R2. If UNION is specified without the ALL option, the result is the set of all objects in either R1 or R2, with duplicate objects eliminated.

If the *n*th member of R1 and the *n*th member of R2 have the same result member name, the *n*th member of R has the same result member name. In all other cases, the result member in R is unnamed.

Qualified member names cannot be used in the ORDER BY clause when UNION or UNION ALL is also specified. The member numbers can be used in the ORDER BY clause.

Two objects are duplicates if each value in the first is equal to the corresponding value of the second. For determining duplicates, two null values are considered equal.

UNION and UNION ALL are associative operations. However, when UNION and UNION ALL are used in the same statement, the result depends on the order in which the operations are performed. Operations within parentheses are performed first. The order of operations is from left to right.

Rules for members: R1 and R2 must have the same number of members and the data type of the nth member of R1 must be compatible with the data type of the nth member of R2.

The *n*th member of the result of UNION and UNION ALL is derived from the *n*th members of R1 and R2.

Examples

- Show all the objects from EMP.

```
SELECT * FROM EMP;
```

- Using sample collections EMP and EMPROJACT, list the employee numbers of all employees for which either of the following statements are true:
  - Their department numbers begin with 'D'
  - They are assigned to projects whose project numbers begin with 'AD'.

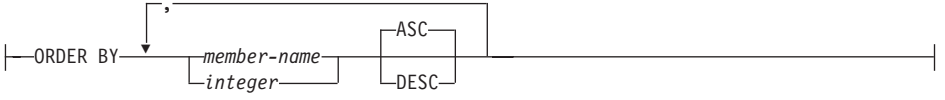
```
SELECT EMPNO FROM EMP
  WHERE WORKDEPT LIKE 'D%'
  UNION
  SELECT EMPNO FROM EMPPROJACT
  WHERE PROJNO LIKE 'AD%';
```

ORDER BY clause

The ORDER BY clause specifies an ordering of the objects of the result collection.

Syntax statement

ORDER-BY clause:



Parameters

member-name
Must unambiguously identify a member of the result collection.

integer Must be greater than 0 and not greater than the number of members in the result collection. The integer *n* identifies the *n*th member of the result.

ASC Uses the values of the member in ascending order. This is the default.

DESC Uses the values of the member in descending order.

Remarks

The collection specified by select-statement is the result of the fullselect. The ORDER BY clause specifies an ordering of the objects of the result collection.

If a single member is identified, the objects are ordered by the values of that member. If more than one member is identified, the objects are ordered by the values of the first identified member, then by the values of the second identified member, and so on.

An unnamed member must be identified by an integer. A member is unnamed if the AS clause is not specified and it is derived from a constant, an expression with operators, or a function. If the fullselect includes a UNION operator, the member numbers must be used to identify the members.

Ordering is performed in accordance with the comparison rules described in “Identifiers” on page 805. The null value is higher than all other values. If your ordering specification does not determine a complete ordering, objects with duplicate values of the last identified member have an arbitrary order. If the ORDER BY clause is not specified, the objects of the result collection have an arbitrary order.

Examples

- Select all the objects from EMP.

```
SELECT * FROM EMP;
```

- Select all the objects from EMP, arranging the result collection in chronological order by date of hiring.

```
SELECT * FROM EMP ORDER BY HIREDATE;
```

- Select the department number (WORKDEPT) and average departmental salary (SALARY) for all departments in the collection EMP. Arrange the result collection in ascending order by average departmental salary.

```
SELECT WORKDEPT, AVG(SALARY)
  FROM EMP
  GROUP BY WORKDEPT
  ORDER BY 2;
```

Chapter 48. Database Data Types Classes

Component Broker provides helper classes that emulate the semantics of the corresponding data types in DB2 for comparison and arithmetic. These classes are:

- ICBCDate Class (represents the DB2 Date type).
- ICBCDecimal Class (represents the DB2 Decimal type).
- ICBCDuration Class (represents the DB2 Date Duration, Time Duration, Timestamp Duration, and Labeled Duration types). The DB2 Duration data type is complementary to the Date, Time, and Timestamp classes, but is not, itself, stored in database tables.
- ICBCTime Class (represents the DB2 Time type).
- ICBCTimestamp Class (represents the DB2 Timestamp type).

The classes can be similarly used to manipulate Oracle DATE and NUMBER datatypes that originate from an Oracle table. The Component Broker Framework will represent the Oracle DATE datatype in a business object as a string attribute, which can be used to initialize a helper class of type ICBCTimestamp. The Oracle NUMBER datatype is also represented as a string which can, with certain retrictions, be used to initialize an instance of ICBCDecimal. Note that classes initialized from Oracle datatypes will be processed through the same functional interface as classes initialized from DB2 data and will exhibit the same DB2 semantics.

Operations performed on or between these classes will produce the same results as equivalent SQL expressions evaluated in DB2. For a description of the corresponding data types in DB2, refer to the *SQL Reference for DB2 Common Servers* or the *SQL Reference for DB2 for MVS/ESA*.

User-written application code (for example, business object methods) can use the classes to manipulate Dates, Decimal, Times, or Timestamps whenever it is desirable that the results should be consistent with those that would be produced by DB2 itself. The Component Broker framework will represent a Date, Decimal, Time, or Timestamp column from a DB2 table as a string attribute in a business object. The user-written application code can get the string attribute from the business object, and use the string to initialize an instance of a Component Broker helper class. The application can then perform comparisons and arithmetic on the instance, and optionally extract a string from the instance to update the attribute back in the business object.

One difference between these classes and DB2 processing of Dates and Times is that the classes cannot take a null value. Any attempt to initialize a Date/Time or Decimal class with a null value string will return an error. It is a responsibility of the application code to detect null attributes and deal with them appropriately without involving the class.

The classes are all built as Component Broker LocalOnly objects and their C++ and Java bindings are generated from their IDLs. The IDL interfaces are documented below, with their member functions. The classes are available for use in VisualAge C++ and Java applications, on the client or server.

Note: C++ applications using the helper classes should link with somdt00i.lib and somdt00i.dll.

ICBCDate Class

The class stores the value of a calendar date between 01/01/0001 and 12/31/9999. All access and manipulation of this date is through a public functional interface. The actual date value and its format are stored in private attributes.

Note that ICBCDate assumes the proleptic Gregorian calendar. The Gregorian calendar in use today is assumed to extend back to year 0001 and it is the responsibility of the application to apply any corrections that may be required for dates prior to the Gregorian calendar reform.

IDL interface description

```
interface ICBCDate :IManagedLocal::ILocalOnly
  {
    // Object initialization
    boolean  initializeFromString(in string aDate);
    boolean  initializeFromNumber(in long aDate);
    boolean  initializeFromTimestamp(in ICBCTimestamp aTimestamp);
    boolean  initializeFromValues(in long aYear, in long aMonth,
                                  in long aDay);

    // Comparisons
    boolean  equalTo(in ICBCDate aDate);
    boolean  notEqualTo(in ICBCDate aDate);
    boolean  lessThan(in ICBCDate aDate);
    boolean  lessThanOrEqualTo(in ICBCDate aDate);
    boolean  greaterThan(in ICBCDate aDate);
    boolean  greaterThanOrEqualTo(in ICBCDate aDate);

    // Query functions
    enum     OutputFormat {ISO, USA, EUR, JIS, LOC};
    string   formattedString(inout string aTarget,
                             in ICBCDate::OutputFormat aFormat);

    long     day();
    long     dayOfWeek();
    long     dayOfYear();
    long     julianDay();
    long     modifiedJulianDay();
    string   dayName();

    long     dayFromString(in string aString);
    long     dayOfWeekFromString(in string aDate);
    long     dayOfYearFromString(in string aDate);
    long     julianDayFromString(in string aDate);
    long     modifiedJulianDayFromString(in string aDate);
    string   dayNameFromString(in string aDate);
    string   dayNameFromNumber(in long aDayIndex);
```

```
    long     month();
    string   monthName();

    long     monthFromString(in string aDate);
    string   monthNameFromString(in string aDate)
    string   monthNameFromNumber(in long aMonthIndex);

    long     quarter();
    long     quarterFromstring(in string aDate);

    long     year();
    long     yearFromString(in string aString);
    boolean  isLeapYear(in long aYear);

    long     daysInObject();
    long     daysInString(in string aDate);
    long         daysInMonth(in long aYear, in long aMonth);
    long         daysInYear(in long aYear);

    boolean  isValidMonthDayYear(in long aMonth, in long aDay,
                                 in long aYear);
    boolean  isValidYearDay(in long aYear, in long aDay);

    boolean  isObjectChanged();

    // Manipulation functions
    void     assignFromDate(in ICBCDate aDate);
    void     increment(in ICBCDuration aDuration);
    void     decrement(in ICBCDuration aDuration);
    long     dateOverflow();

    ICBCDuration intervalFromDate(in ICBCDate aDate);
    ICBCDuration intervalFromString(in string aDate);
    ICBCDuration intervalFromLong(in long aDate);
};
```

Supported methods

ICBCDate::_create
ICBCDate::assignFromDate
ICBCDate::dateOverflow
ICBCDate::day
ICBCDate::dayFromString
ICBCDate::dayName
ICBCDate::dayNameFromNumber
ICBCDate::dayNameFromString
ICBCDate::dayOfWeek
ICBCDate::dayOfWeekFromString
ICBCDate::dayOfYear
ICBCDate::dayOfYearFromString
ICBCDate::daysInMonth
ICBCDate::daysInObject
ICBCDate::daysInString
ICBCDate::daysInYear

ICBCDate::decrement
ICBCDate::equalTo
ICBCDate::formattedString
ICBCDate::greaterThan
ICBCDate::greaterThanOrEqualTo
ICBCDate::increment
ICBCDate::initializeFromNumber
ICBCDate::initializeFromString
ICBCDate::initializeFromTimestamp
ICBCDate::initializeFromValues
ICBCDate::interval
ICBCDate::isObjectChanged
ICBCDate::isLeapYear
ICBCDate::isValidMonthDayYear
ICBCDate::isValidYearDay
ICBCDate::julianDay
ICBCDate::julianDayFromString
ICBCDate::lessThan
ICBCDate::lessThanOrEqualTo
ICBCDate::modifiedJulianDay
ICBCDate::modifiedJulianDayFromString
ICBCDate::month
ICBCDate::monthFromString
ICBCDate::monthName
ICBCDate::monthNameFromNumber
ICBCDate::monthNameFromString
ICBCDate::notEqualTo
ICBCDate::quarter
ICBCDate::quarterFromString
ICBCDate::year
ICBCDate::yearFromString

ICBCDate::_create

Returns a pointer to a new IDBCDate object. The object is intialized to the current system date.

ICBCDate::assignFromDate

Sets all state data in this ICBCData object equal to the second ICBCData object. The change flag of this object is set to `TRUE`.

IDL syntax

```
ICBCDate assignFromDate (in ICBCDate aDate);
```

ICBCDate::dateOverflow

Returns the number of overflow or underflow years of a preceding increment or decrement operation. For an increment, returns the years in excess of year 9999; for decrement, the number of years less than year 0001.

IDL syntax

```
long dateOverflow();
```

ICBCDate::day

Returns the day part of this object, as a number between 1 and 31.

IDL syntax

```
long day();
```

ICBCDate::dayFromString

Returns the day part Returns the day part of the date encoded in the input string parameter.

The input string parameter must conform to a format recognizable as a date. Refer to IBCDate::initializeFromString() for a description of supported formats. If the input parameter is not recognizable, a value of -1 is returned.

IDL syntax

```
long dayFromString(in string aString);
```

ICBCDate::dayName

Returns a string containing the name of the day of the week for this date object.

IDL syntax

```
string dayName();
```

ICBCDate::dayNameFromNumber

Returns a string containing the name of the day of the week for the day number within a week provided in the input parameter. The input parameter must be a number between 1 and 7. Sunday is day number 1.

If the input parameter is not recognizable, an empty string is returned.

IDL syntax

```
string dayNameFromNumber(in long aDayIndex);
```

ICBCDate::dayNameFromString

Returns a string containing the name of the day of the week for the date encoded in the input string parameter. The input string parameter must conform to a format recognizable as a date. Refer to “ICBCDate::initializeFromString” on page 872 for a description of supported formats.

If the input parameter is not recognizable, an empty string is returned.

IDL syntax

```
long dayNameFromString(in string aDate);
```

ICBCDate::dayOfWeek

Returns a number between 1 and 7 representing the day of the week for this date object. Sunday is day number 1.

IDL syntax

```
long dayOfWeek();
```

ICBCDate::dayOfWeekFromString

Returns a number between 1 and 7 representing the day of the week for the date encoded in the input string parameter. Sunday is day number 1.

The input string parameter must conform to a format recognizable as a date. Refer to “ICBCDate::initializeFromString” on page 872 for a description of supported formats. If the input parameter is not recognizable, a value of -1 is returned.

IDL syntax

```
long dayOfWeekFromString(in string aString);
```

ICBCDate::dayOfYear

Returns a number between 1 and 366 representing the day of the year for this date object. January 1 is day number 1.

IDL syntax

```
long dayOfYear();
```

ICBCDate::dayOfYearFromString

Returns a number between 1 and 366 representing the day of the year for the date encoded in the input string parameter. January 1 is day number 1.

The input string parameter must conform to a format recognizable as a date. Refer to “ICBCDate::initializeFromString” on page 872 for a description of supported formats. If the input parameter is not recognizable, a value of -1 is returned.

IDL syntax

```
long dayOfYearFromString(in string aString);
```

ICBCDate::daysInMonth

Returns the number of days in the month of the year provided in the input parameters. Month number must be between 1 and 12.

If the input parameters are not valid, a value of -1 is returned.

IDL syntax

```
long daysInMonth(in long aYear, in long aMonth);
```

ICBCDate::daysInObject

Returns the number of days from January 1, 0001 to the date represented by this object, plus one day. For example, an object holding the date January 2, 0001 returns the value 2.

IDL syntax

```
long daysInObject();
```

ICBCDate::daysInString

Returns the number of days from January 1, 0001 to the date represented by the string parameter, plus one day. For example, an object holding the date 00010102 returns the value 2. If the string is invalid, the value –1 is returned.

IDL syntax

```
long daysInString(in string aDate);
```

Parameters

aDate The string must be one of the following:

- A string of length 7 of the form yyyynnn.

- A string of length 6 of the form yyyymmdd.
- A punctuated string representation of a date.

For additional details on the *string* parameter, see "ICBCDate::initializeFromString" on page 872.

ICBCDate::daysInYear

Returns the number of days in the year provided in the input parameter.

If the input parameter is not valid, a value of -1 is returned.

IDL syntax

```
long daysInYear(in long aYear);
```

ICBCDate::decrement

Subtracts a ICBCDuration from the value of this ICBCDate object.

IDL syntax

```
void decrement(in ICBCDuration aDuration);
```

Remarks

This operation is performed essentially as the inverse of "ICBCDate::increment" on page 871. When the duration of type DATE is subtracted from a date, the date is decremented by the specified number of days, months, and years (in this order). If the years value of the result of an decrement operation is less than 0001 years, the years value of the result is set to `0001` and the number of underflow years can be retrieved as a negative number using "ICBCDate::dateOverflow" on page 867. The number of overflow years is cleared by the next assignFromDate or arithmetic operation on the object.

Decrementing with a ICBCDuration of type HOURS, MINUTES, SECONDS, TIME, or TIMESTAMP will have no effect on an object of type ICBCDate, even if, say, the number of hours in the duration object exceeds 24.

The change flag of this object is set to `TRUE`.

ICBCDate::equalTo

Returns TRUE if the date in this ICBCDate object is equal to the other ICBCDate object.

IDL syntax

```
boolean equalTo(in ICBCDate aDate);
```

ICBCDate::formattedString

Returns the date value of this object as a string in a specified format. The parameter OutputType must take one of the values: ISO, USA, EUR, JIS. Formats are as described for “ICBCDate::initializeFromString” on page 872. The LOC site-defined operation described in the DB2 manual is not supported by ICBCDate.

For the C++ implementation of ICBCDate, the application must supply a non-const pointer to a target storage area long enough to receive the formatted string. The function returns a reference to this area. For the Java implementation, this parameter is required, but plays no active role. It is sufficient to provide a null parameter of type org.omg.CORBA.StringHolder.

IDL syntax

```
string formattedString(inout string aOutput,
                       in ICBCDate::OutputFormat aFormat)
```

ICBCDate::greaterThan

Returns TRUE if the date stored in this ICBCDate object is greater than the other ICBCDate object.

IDL syntax

```
boolean greaterThan(in ICBCDate aDate);
```

ICBCDate::greaterThanOrEqualTo

Returns TRUE if the date stored in this ICBCDate object is greater than or equal to the other ICBCDate object.

IDL syntax

```
boolean greaterThanOrEqualTo(in ICBCDate aDate);
```

ICBCDate::increment

Adds a ICBCDuration to the value of this ICBCDate object. If the duration is negative, the operation is performed as a decrement. If a duration of type YEARS is added to the date, only the year portion of the date is affected. The month is unchanged, as is the day, unless the result would be February 29 of a non-leap-year. In this case, the day is changed to 28.

Similarly, if a duration of type MONTHS is added, only months and, if necessary, years are affected. Note that adding months to a Date is like turning the pages of a calendar, starting with the page on which the date appears. The day portion of the date is unchanged unless the result would be invalid (September 31, for example). In this case, the day is set to the last day of the month. If one or more months is added to a given date, and then the same number of months is subtracted from the result, the final date is not necessarily the same as the original date.

Adding a duration of type DAYS will, of course, affect the day portion of the date, and potentially the month and year.

When a duration of type DATE is added to a date, the date is incremented by the specified number of year, months, and days, in this order.

Incrementing with a ICBCDuration of type HOURS, MINUTES, SECONDS, TIME, or TIMESTAMP will have no effect on an object of type ICBCDate, even if, say, the number of hours in the duration object exceeds 24.

If the years value of the result of an increment operation is greater than 9999 years, then the years value of the result is set to 9999, and the number of overflow years can be retrieved using the dateOverflow() function. The number of overflow years is cleared by the next assignFromDate of arithmetic operation on the object.

The change flag of this object is set to TRUE.

IDL syntax

```
void increment(in ICBCDuration aDuration);
```

ICBCDate::initializeFromNumber

Accepts a numerical value for the date. Any existing value in the object is replaced. The parameter aDate must be positive, non-zero, and less than 3652059. Creates a date that is (aDate-1) days beyond January 1 0001, i.e., January 2 0001 is day number 2. Returns TRUE if the number is a valid date representation, FALSE otherwise. If FALSE is returned the state of the object is unchanged.

IDL syntax

```
boolean initializeFromNumber(in long aDate);
```

ICBCDate::initializeFromString

Accepts a value for the date in the form of a string. Returns TRUE if the string is a valid date representation and the object is successfully initialized. Any existing value in the object is replaced. Returns FALSE if the parameter is invalid. If FALSE is returned the state of the object is unchanged.

IDL syntax

```
boolean initializeFromString(in string aDate);
```

Parameters

aDate The string must be:

- A string of length 7 of the form yyyynnn.

  The yyyyddd string must consist of entirely numeric characters with padding zeros where necessary -- embedded signs or blanks are not allowed. However leading and trailing blanks are acceptable provided the string length is less than 20 characters. Only SBCS characters may be present in the string.
- A string of length 8 of the form yyyymmdd.

  The yyyymmdd string must consist of entirely numeric characters with padding zeros where necessary -- embedded signs or blanks are not allowed. However leading and trailing blanks around the string are acceptable provided the string length is less than 20 characters. Only SBCS characters may be present in the string.
- A valid punctuated string representation of a date, in one of the following valid forms:
  - International Standards Organization (ISO) (yyyy-mm-dd).
  - IBM USA standard (USA) (mm/dd/yyyy).
  - IBM European standard (EUR) (dd.mm.yyyy)
  - Japanese Industrial Standard Christian Era (JIS) (yyyy-mm-dd).

  Leading and trailing blanks are acceptable provided the string length is less than 20 characters. Leading zeros may be omitted from the month and day portions. Only SBCS characters may be present in the string.
- A string representation of a timestamp.

  Any of the string forms acceptable by the “ICBCTimestamp::initializeFromString” on page 929 function are acceptable. For details refer to that function. Briefly, the forms include:

  - A punctuated string: yyyy-mm-dd-hh.mm.ss.nnnnnn.
  - A 14 byte string without puctuation, of the form: yyyymmddhhmmss.

ICBCDate::initializeFromTimestamp

Initializes this object from the date portion of a Timestamp object.

IDL syntax

```
boolean initializeFromTimestamp(in ICBCTimestamp
                                aTimestamp);
```

ICBCDate::initializeFromValues

Accepts the date in the form of three separate value parameters for year, month, and day. The values must make a valid date. Returns TRUE if the values are a valid date representation, FALSE otherwise. If FALSE is returned the state of the object is unchanged.

IDL syntax

```
boolean initializeFromValues(in long aYear,in long aMonth,
                             in long aDay);
```

ICBCDate::interval

Calculates the Duration between two ICBCDate objects. The result is a pointer to a new ICBCDuration object of type DATE.

If the interval between DATE1 and DATE2 is required, and DATE1 is greater than or equal to DATE2, the operation is performed by subtracting DATE2 from DATE1. If, however, DATE1 is less than DATE2, DATE1 is subtracted from DATE2, and the sign of the result is made negative.

The following procedural description clarifies the steps involved in the operation result = DATE1 - DATE2.

```
If DAY(DATE2) <= DAY(DATE1)
   then DAY(RESULT) = DAY(DATE1) - DAY(DATE2).
If DAY(DATE2) > DAY(DATE1)
   then DAY(RESULT) = N + DAY(DATE1) - DAY(DATE2)
        where N = the last day of MONTH(DATE2).
        MONTH(DATE2) is then incremented by 1.
If MONTH(DATE2) <= MONTH(DATE1)
   then MONTH(RESULT) = MONTH(DATE1) - MONTH(DATE2).
If MONTH(DATE2) > MONTH(DATE1)
   then MONTH(RESULT) = 12 + MONTH(DATE1) - MONTH(DATE2).
        YEAR(DATE2) is incremented by 1.
YEAR(RESULT) = YEAR(DATE1) - YEAR(DATE2).
```

IDL syntax

```
ICBCDuration_ptr intervalFromDate(in ICBCDate aDate);
ICBCDuration_ptr intervalFromString(in string aDate);
ICBCDuration_ptr intervalFromLong(in long aDate);
```

In the cases with a string parameter, the parameter must be recognizable as a date. For details refer to the description of “ICBCDate::initializeFromString” on page 872. If parameters to this function cannot be recognized as a valid date, then a pointer value of -1 (for C++) or null (for Java) is returned..

ICBCDate::isObjectChanged

Returns a boolean indicating whether the value of this object has been changed since it was initialized.

IDL syntax

```
boolean isObjectChanged();
```

ICBCDate::isLeapYear

Returns TRUE if the year number in the input parameter is a leap year, FALSE otherwise. Year number must be between 1 and 9999.

If the input parameter is not in this range, FALSE is returned.

IDL syntax

```
boolean isLeapYear(in long aYear);
```

ICBCDate::isValidMonthDayYear

Returns TRUE if the provided values for month, day, and year make a valid date, and FALSE otherwise.

IDL syntax

```
boolean isValidMonthDayYear(in long aMonth, in longMonth, in longYear);
```

ICBCDate::isValidYearDay

Returns TRUE if the provided values for year, and day within year make a valid date, and FALSE otherwise.

IDL syntax

```
boolean isValidYearDay(in long aYear,in long aDay);
```

ICBCDate::julianDay

Returns the Julian day number corresponding to noon GMT on the date represented by this object. (Julian Day count started at noon, GMT, on January 1, 4712 BC

IDL syntax

```
long julianDay();
```

ICBCDate::julianDayFromString

Returns the Julian day number corresponding to noon GMT on the date encoded in the input string parameter. (Julian Day count started at noon, GMT, on January 1, 4712 BC). The input string parameter must conform to a format recognizable as a date. Refer to “ICBCDate::initializeFromString” on page 872 for a description of supported formats. If the input parameter is not recognizable, a value of -1 is returned.

IDL syntax

```
long julianDayFromString(in string aString);
```

ICBCDate::lessThan

Returns TRUE if the date stored in this ICBCDate object is less than the other ICBCDate object.

IDL syntax

```
boolean lessThan(in ICBCDate aDate);
```

ICBCDate::lessThanOrEqualTo

Returns TRUE if the date stored in this ICBCDate object is less than or equal to the other ICBCDate object.

IDL syntax

```
boolean lessThanOrEqualTo(in ICBCDate aDate);
```

ICBCDate::modifiedJulianDay

Returns the Modified Julian day number (MJD) corresponding to time 00:00:00 GMT on the date represented by this object. MJD is defined as Julian Day — 2,400,000.5 days. It thus starts at time 00:00:00 (i.e. midnight), in line with normal civil practice, and unlike Julian Day which extends from noon to noon.

IDL syntax

```
long modified JulianDay();
```

ICBCDate::modifiedJulianDayFromString

Returns the Modified Julian day number (MJD) corresponding to time 00:00:00 GMT on the date encoded in the input string parameter. MJD is defined as Julian Day - 2,400,000.5 days. It thus starts at time 00:00:00 (i.e., midnight), in line with normal civil practice, and unlike Julian Day which extends from noon to noon.

The input string parameter must conform to a format recognizable as a date. Refer to “ICBCDate::initializeFromString” on page 872 for a description of supported formats. If the input parameter is not recognizable, a value of -1 is returned.

IDL syntax

```
long modifiedJulianDayFromString(in string aString);
```

ICBCDate::month

Returns the month part of this object, as a number between 1 and 12.

IDL syntax

```
long month();
```

ICBCDate::monthFromString

Returns the month part of the date encoded in the input string parameter. The input string parameter must conform to a format recognizable as a date. Refer to “ICBCDate::initializeFromString” on page 872 for a description of supported formats.

If the input parameter is not recognizable, a value of -1 is returned.

IDL syntax

```
long monthFromString(in string aString);
```

ICBCDate::monthName

Returns a string containing the name of the month of this object.

IDL syntax

```
long monthName();
```

ICBCDate::monthNameFromNumber

Returns a string containing the name of the month for the month number within a year provided in the input parameter. The input parameter must be a number between 1 and 12.

If the input parameter is not recognizable, an empty string is returned.

IDL syntax

```
string monthNameFromNumber(in long aDayIndex);
```

ICBCDate::monthNameFromString

Returns a string containing the name of the month for the date encoded in the input string parameter. The input string parameter must conform to a format recognizable as a date. Refer to “ICBCDate::initializeFromString” on page 872 for a description of supported formats.

If the input parameter is not recognizable, an empty string is returned.

IDL syntax

```
string monthNameFromString(in string aDate);
```

ICBCDate::notEqualTo

The opposite of the operator== member.

IDL syntax

```
boolean notEqualTo(in ICBCDate aDate);
```

ICBCDate::quarter

Returns a number for the quarter within the year for this object. Dates between January 1 and March 31 are in the first quarter, and so on.

IDL syntax

```
long quarter();
```

ICBCDate::quarterFromString

Returns a number for the quarter within the year for the date encoded in the input string parameter. The input string parameter must conform to a format recognizable as a date. Refer to “ICBCDate::initializeFromString” on page 872 for a description of supported formats. Dates between January 1 and March 31 are in the first quarter, and so on.

If the input parameter is not recognizable, a value of -1 is returned.

IDL syntax

```
long quarterFromString(in string aDate);
```

ICBCDate::year

Returns the year part of the date.

IDL syntax

```
long year();
```

ICBCDate::yearFromString

Returns the year part of the date encoded in the input string parameter. The input string parameter must conform to a format recognizable as a date. Refer to “ICBCDate::initializeFromString” on page 872 for a description of supported formats.

If the input parameter is not recognizable, a value of -1 is returned.

IDL syntax

```
long yearFromString(in string aString);
```

ICBCDecimal Class

The class stores the value of a DB2 Decimal data type, or Oracle number type in which scale is less than or equal to precision, and positive with a maximum precision of 38. All access and manipulation of this value is through a public functional interface. The actual decimal value and its precision and scale are stored in private attributes.

Important

Throughout the description of ICBCDecimal, precision is the total number of digits in the decimal number; scale is the number of digits in the fractional part of the number — the number of digits that lie to the right of the decimal point.

IDL interface description

```
interface ICBCDecimal :IManagedLocal::ILocalOnly
  {
    // Object initialization (DB2 semantics)
    boolean  initializeFromString1(in string aString, in short aPrecision,
                                   in short aScale, in char aDecimalChar);
    boolean  initializeFromString2(in string aString, in char aDecimalChar);
    boolean  initializeFromDouble(in double aDouble,in short aPrecision,
                                  in short aScale);
    boolean  initializeFromFloat(in float aFloat, in short aPrecision,
                                 in short aScale);
    boolean  initializeFromLong(in long aLong, in short aPrecision,
                                in short aScale);
    boolean  initializeFromShort(in short aShort, in short aPrecision,
                                 in short aScale);
    boolean  initializeFromDecimal1(in ICBCDecimal aDecimal,
                                    in short aPrecision, in short aScale);
    boolean  initializeFromDecimal2(in ICBCDecimal aDecimal);
    boolean  initializeFromPackedDecimal(in string aString,
                                         in short aPrecision,
```

```
                                        in short aScale);

  //Comparisons
  boolean  equalTo(in ICBCDecimal aDecimal);
  boolean  notEqualTo(in ICBCDecimal aDecimal);
  boolean  lessThan(in ICBCDecimal aDecimal);
  boolean  lessThanOrEqualTo(in ICBCDecimal aDecimal);
  boolean  greaterThan(in ICBCDecimal aDecimal);
  boolean  greaterThanOrEqualTo(in ICBCDecimal aDecimal);

  //Query functions
  string   getAsFormattedString1(inout string aOutput,
                                 in char aDecimalCharacter);
  string   getAsFormattedString2(inout string aOutput,
                                 in char aDecimalCharacter);
  string   getAsFormattedString3(inout string aOutput,
                                 in char aDecimalCharacter);
  double   getAsDouble();
  float    getAsFloat();
  long     getAsLong();
  short    getAsShort();
  string   getAsDigits(inout string aOutput);
  string   getAsPackedDecimal(inout string aOutput,
                              inout short aPrecision,
                              inout short aScale);
  short    getPrecision();
  short    getScale();
  boolean  isNegative();
  boolean  isChanged();

  // Manipulation functions
  ICBCDecimal  assignFromDecimal(in ICBCDecimal aDecimal);
  ICBCDecimal  assignFromDouble(in double aDouble);
  ICBCDecimal  assignFromFloat(in float aFloat);
  ICBCDecimal  assignFromLong(in long aLong);
  ICBCDecimal  assignFromShort(in short aShort);

  ICBCDecimal  increment(in ICBCDecimal aDecimal);
  ICBCDecimal  decrement(in ICBCDecimal aDecimal);
  ICBCDecimal  multiplyThisObjectBy(in ICBCDecimal aDecimal);
  ICBCDecimal  divideThisObjectBy(in ICBCDecimal aDecimal);
  ICBCDecimal  remainderInThisObject(in ICBCDecimal aDecimal);

  ICBCDecimal  addWithNewObject(in ICBCDecimal aDecimal);
  ICBCDecimal  subtractWithNewObject(in ICBCDecimal aDecimal);
  ICBCDecimal  multiplyWithNewObject(in ICBCDecimal aDecimal);
  ICBCDecimal  divideWithNewObject(in ICBCDecimal aDecimal);
  ICBCDecimal  remainderWithNewObject(in ICBCDecimal aDecimal);

  ICBCDecimal  getPrecedingRemainder();

  void         swapSign();
  boolean      decimalOverflow();

};
```

Supported methods

ICBCDecimal::addWithNewObject
ICBCDecimal::assignFromDecimal
ICBCDecimal::assignFromDouble
ICBCDecimal::assignFromFloat
ICBCDecimal::assignFromLong
ICBCDecimal::assignFromShort
ICBCDecimal::_create
ICBCDecimal::decimalOverflow
ICBCDecimal::decrement
ICBCDecimal::divideThisObjectBy
ICBCDecimal::divideWithNewObject
ICBCDecimal::equalTo
ICBCDecimal::getAsDouble
ICBCDecimal::getAsFloat
ICBCDecimal::getAsFormattedString1
ICBCDecimal::getAsFormattedString2
ICBCDecimal::getAsFormattedString3
ICBCDecimal::getAsDigits
ICBCDecimal::getAsLong
ICBCDecimal::getAsPackedDecimal
ICBCDecimal::getAsShort
ICBCDecimal::getPreceedingRemainder
ICBCDecimal::getPrecision
ICBCDecimal::getScale
ICBCDecimal::greaterThan
ICBCDecimal::greaterThanOrEqualTo
ICBCDecimal::increment
ICBCDecimal::initializeFromDecimal1
ICBCDecimal::initializeFromDecimal2
ICBCDecimal::initializeFromDouble
ICBCDecimal::initializeFromFloat
ICBCDecimal::initializeFromLong
ICBCDecimal::initializeFromPackedDecimal
ICBCDecimal::initializeFromShort
ICBCDecimal::initializeFromString1
ICBCDecimal::initializeFromString2
ICBCDecimal::isChanged
ICBCDecimal::isNegative
ICBCDecimal::lessThan
ICBCDecimal::lessThanOrEqualTo
ICBCDecimal::multiplyWithNewObject
ICBCDecimal::multiplyThisObjectBy
ICBCDecimal::notEqualTo
ICBCDecimal::remainderInThisObject
ICBCDecimal::remainderWithNewObject
ICBCDecimal::subtractWithNewObject
ICBCDecimal::swapSign

ICBCDecimal::addWithNewObject

Adds the receiving object and the argument ICBCDecimal object and returns a new ICBCDecimal object containing the result. The original operand objects are unchanged. The precision of the result object is:

min(38, max(*p*1–*s*1, *p*2–*s*2) + max(*s*1, *s*2) + 1)

The scale is: max(*s*1, *s*2)

IDL syntax

```
ICBCDecimal addWithNewObject(in ICBCDecimal aDecimal);
```

Remarks

After a addWithNewObject operation, the change flag of the new object is set to TRUE. The change flag of the original object is unchanged.

It is the application's responsibility to delete the new object, freeing its memory, when no longer needed.

If the non-zero digits to the left of the decimal point in the result object would exceed (*p* - *s*) of the result, or if other errors occur that prevent the creation of the new object, an ICBCDecimal pointer with value -1 (for C++) or null (for Java) is returned.

ICBCDecimal::assignFromDecimal

Sets the value of the receiving ICBCDecimal object equal to that of the second ICBCDecimal object. The precision and scale of the receiving object are not changed.

IDL syntax

```
ICBCDecimal assignFromDecimal(in ICBCDecimal aDecimal);
```

Remarks

If the non-zero digits to the left of the decimal point in the source object exceed (*p*1 — *s*1) of the receiving object (that is, the whole number of the source object will not fit in the receiving object), the receiving object is unchanged, and a following invocation of decimalOverflow() on the receiving object function will return the value TRUE.

The scale of the source object is reduced to fit the scale of the receiving object, if necessary, by truncating digits from the right.

Any remainder value in the source object resulting from a preceding division operation is not transferred to the receiving object.

The change flag of the receiving object is set to TRUE.

Note that this operation preserves the precision and scale of the receiving object, as in a DB2 regular assignment between two decimal types. In contrast, the ICBCDecimal::initializeFromDecimal() function sets the precision and scale of the receiving object, as in the DB2 DECIMAL built-in function.

ICBCDecimal::assignFromDouble

Sets the value of a receiving ICBCDecimal object equal to a double variable. The precision and scale of the object is not changed.

IDL syntax

```
ICBCDecimal assignFromDouble(in double aDouble);
```

Remarks

The operand is converted to a temporary decimal object of precision 38. Numbers less than 0.5E-38 will be rounded to zero. The temporary decimal number is then converted to the precision and scale of the receiving ICBCDecimal object.

If the non-zero digits to the left of the decimal point in the source object exceed (*p*1 — *s*1) of the receiving object (that is, the whole number portion of the source object will not fit in the receiving object), the receiving object is unchanged, and a following invocation of decimalOverflow() function on the receiving object will return the value TRUE.

ICBCDecimal::assignFromFloat

Sets the value of a receiving ICBCDecimal object equal to a float variable. The precision and scale of the object is not changed.

IDL syntax

```
ICBCDecimal assignFromFloat(in float aFloat);
```

Remarks

The operand is converted to a temporary decimal object of precision 38. Numbers less than 0.5E-38 will be rounded to zero. The temporary decimal number is then converted to the precision and scale of the receiving ICBCDecimal object.

If the non-zero digits to the left of the decimal point in the source object exceed (*p*1 — *s*1) of the receiving object (that is, the whole number portion of the source object will not fit in the receiving object), the receiving object is unchanged, and a following invocation of decimalOverflow() function on the receiving object will return the value TRUE.

ICBCDecimal::assignFromLong

Sets the value of a receiving ICBCDecimal object equal to a long variable. The precision and scale of the object is not changed.

IDL syntax

```
ICBCDecimal assignFromLong(in long aLong);
```

Remarks

The operand is converted to a temporary decimal number, of precision 11 which is then converted to the precision and scale of the receiving object.

If the non-zero digits to the left of the decimal point in the source object exceed (*p*1 — *s*1) of the receiving object (that is, the whole number portion of the source object will not fit in the receiving object), the receiving object is unchanged, and a following invocation of decimalOverflow() function will return the value TRUE on the receiving object.

ICBCDecimal::assignFromShort

Sets the value of a receiving ICBCDecimal object equal to a short variable. The precision and scale of the object is not changed.

IDL syntax

```
ICBCDecimal assignFromShort(in short aShort);
```

Remarks

The operand is converted to a temporary decimal number, of precision 5 which is then converted to the precision and scale of the receiving object.

If the non-zero digits to the left of the decimal point in the source object exceed (*p*1 — *s*1) of the receiving object (that is, the whole number portion of the source object will not fit in the receiving object), the receiving object is unchanged, and a following invocation of decimalOverflow() function on the receiving object will return the value TRUE.

ICBCDecimal::_create

Returns a pointer to a new ICBCDecimal object. The object is initialized to zero, with precision=1 and scale=0.

ICBCDecimal::decimalOverflow

Following certain functions that potentially cause a precision overflow in a ICBCDecimal object, this function indicates whether an overflow did, in fact, take place. If the most recent initialization, assignment, or arithmetic operation caused an overflow, this function will return TRUE. Otherwise it returns FALSE. Intervening query or comparison operations will not affect the returned value.

IDL syntax

```
boolean decimalOverflow();
```

ICBCDecimal::decrement

Subtracts the argument ICBCDecimal object from the receiving ICBCDecimal object and sets the value of the receiving object to the result. The precision and scale of the receiving object are reset.

The precision is given by:
min(38, max(*p*1—*s*1, *p*2–*s*2) + max(*s*1, *s*2) + 1)

The scale is given by:
max(*s*1, *s*2)

If the non-zero digits to the left of the decimal point in the result exceed (*p* — *s*) of the result, the receiving object is not changed, and a following invocation of the function decimalOverflow() will return TRUE.

The change flag of the receiving object is set to TRUE.

IDL syntax

```
ICBCDecimal decrement(in ICBCDecimal aDecimal);
```

ICBCDecimal::divideThisObjectBy

Divides the receiving object by the argument ICBCDecimal object, and sets the receiving object to the quotient. The precision and scale of the receiving object are reset.

The precision is 38.

The scale is given by:
38 — *p*1 + *s*1 — *s*2

Scale may not be negative.

The division proceeds as far as possible within these limits for the quotient and then stops. Any remainder can be retrieved using the getPrecedingRemainder() function (see “ICBCDecimal::getPreceedingRemainder” on page 891).

This function does not cause any overflow conditions. However, the decimalOverflow() function is used to warn of zeroDivide situations, returning TRUE if a zeroDivide occured in the preceding divideThisObjectBy() function, and FALSE if it did not.

After a divideThisObjectBy operation, the change flag of the receiving object is set to TRUE.

For example, 200.00 (precision 5, scale 2) divided by 3.00 (precision 3, scale 2) returns a quotient of precision 38 and scale 33:

00066.66666666666666666666666666666666

A subsequent call to getPrecedingRemainder() on the same object returns a remainder with precision 38 and scale 35:

000.00000000000000000000000000000200

IDL syntax

```
ICBCDecimal divideThisObjectBy(in ICBCDecimal aDecimal);
```

ICBCDecimal::divideWithNewObject

Divides the receiving object and the argument ICBCDecimal object, and returns a new ICBCDecimal object containing the quotient. The original operand objects are unchanged.

The precision of the result object is 38.

The scale is: 38 — *p*1 + *s*1 — *s*2

IDL syntax

```
ICBCDecimal divideWithNewObject(in ICBCDecimal aDecimal);
```

Remarks

The division proceeds as far as possible within these limits and then stops. Any remainder can be retrieved using the getPrecedingRemainder() function (see “ICBCDecimal::getPreceedingRemainder” on page 891).For an example of a divide operation, see “ICBCDecimal::divideThisObjectBy” on page 885.

After a divideWithNewObject operation, the change flag of the new object is set to TRUE. The change flag of the original object is unchanged.

It is the application’s responsibility to delete the new remainder object, freeing its memory, when no longer needed.

This function does not cause any overflow conditions, by may encounter a zeroDivide situation. If so, or if errors occurred that prevented the creation of the new object, an ICBCDecimal pointer with value -1 (for C++) or null (for Java) is returned.

ICBCDecimal::equalTo

Returns TRUE if the value of the receiving ICBCDecimal object is equal to the other ICBCDecimal object.

IDL syntax

```
boolean equalTo(in ICBCDecimal aDecimal);
```

ICBCDecimal::getAsDigits

Returns a null terminated string of length equal to the precision of the object, containing the digits in the value of the object, left justified, and without either decimal point or sign. Leading zeros are provided if necessary.For example, an object of precision and scale of 6 and 2 respectively, containing a value of -6.28 would return a string containing '000628'.

IDL syntax

```
string getAsDigits(inout string aOutput);
```

Remarks

For the C++ implementation of ICBCDecimal, the application must supply a non-const pointer to a target storage area long enough to receive the formatted string. The function returns a reference to this area. For the Java implementation, this parameter is required, but plays no active role. It is sufficient to provide a null parameter of type org.omg.CORBA.StringHolder.

ICBCDecimal::getAsDouble

Returns the value of the receiving object as a double. Since the double representation has maximum precision less than that of a decimal data type in DB2, accuracy of the object value may be lost.

IDL syntax

```
double getAsDouble();
```

ICBCDecimal::getAsFloat

Returns the value of the receiving object as a float. Since the float representation has maximum precision less than that of a decimal data type in DB2, accuracy of the object value may be lost.

IDL syntax

```
double getAsFloat();
```

ICBCDecimal::getAsFormattedString1

Returns the value of the receiving object as a formatted string. This function is equivalent to the CHAR() built-in function of DB2.

For the C++ implementation of ICBCDecimal, the application must supply a non-const pointer to a target storage area long enough to receive the formatted string. The function returns a reference to this area. For the Java implementation, this parameter is required, but plays no active role. It is sufficient to provide a null parameter of type org.omg.CORBA.StringHolder.

IDL syntax

```
string getAsFormattedString1(inout string aOutput,
                             in char aDecimalCharacter);
```

Parameters

aDecimalCharacter

aDecimalCharacter must be a single character to be used as the “decimal point” character in the output.The characters plus (+) and minus (-) are not permitted, and blanks are not permitted.

The length of the output string is always precision + 2, where precision is the current precision of the object. One of the additional character positions is always used for the decimal point. If required, the the additional character position is used for a leading sign. If a sign is not required, the output string will have a trailing blank. Leading and trailing padding zeros are provided if necessary to fill the current precision and scale of the object.

For example, assuming an aDecimalCharacter of ’.’ :

Value	Precision	Scale	Prints as
45	2	0	"45. "
123.33	9	4	"00123.3300 "
-123.33	9	4	"-00123.3300"

ICBCDecimal::getAsFormattedString2

Returns the value of the receiving object as a string formatted without leading zero padding.

For the C++ implementation of ICBCDecimal, the application must supply a non-const pointer to a target storage area long enough to receive the formatted string. The function returns a reference to this area. For the Java implementation, this parameter is required, but plays no active role. It is sufficient to provide a null parameter of type org.omg.CORBA.StringHolder.

IDL syntax

```
string getAsFormattedString2(inout string aOutput,
                             in char aDecimalCharacter);
```

Parameters

aDecimalCharacter

aDecimalCharacter must be a single character to be used as the “decimal point” character in the output. The characters plus (+) and minus (-) are not permitted, and blanks are not permitted.

The length of the output string is always precision + 2, where precision is the current precision of the object. The leftmost character position is reserved for the sign, if necessary. The following positions contain the the number, with an embedded decimal point, which is always present, even if the scale of the number is zero. The digits in the number are padded with leading blanks and trailing zeros to fill the full current precision and scale of the object. Finally, if leading blanks are present, the sign moves to a position immediately adjacent to the first digit (or decimal point).

For example, assuming an aDecimalCharacter of ’.’, and showing blank as the character ’b’ :

```
Value          Precision       Scale       Prints as
-----          ---------       -----       ---------
45             2               0           "b45."
123.33         9               4           "bbb123.3300"
-123.33        9               4           "bb-123.3300"
```

ICBCDecimal::getAsFormattedString3

Returns the value of the receiving object formatted as a left justified string with a leading sign if required, and a decimal point only if there are digits that lie to the right of the decimal point. There are no leading or trailing zeros or blanks.

For the C++ implementation of ICBCDecimal, the application must supply a non-const pointer to a target storage area long enough to receive the formatted string. The

function returns a reference to this area. For the Java implementation, this parameter is required, but plays no active role. It is sufficient to provide a null parameter of type org.omg.CORBA.StringHolder.

IDL syntax

```
string getAsFormattedString3(inout string aOutput,
                             in char aDecimalCharacter);
```

Parameters

aDecimalCharacter

aDecimalCharacter must be a single character to be used as the “decimal point” character in the output. The characters plus (+) and minus (-) are not permitted, and blanks are not permitted.

The length of the output string is variable, containing only enough characters to hold the significant digits plus a sign and a decimal point if required. The precision and scale of the object are not reflected in the length output string.

This format is suitable for providing to the Oracle TO_NUMBER function without the need for an associated “NUMBER format element” parameter.

For example, assuming an aDecimalCharacter of ’.’:

Value	Precision	Scale	Prints as
45	2	0	"45."
123.33	9	4	"123.33"
-123.33	9	4	"-123.33"

ICBCDecimal::getAsLong

Returns the value of the receiving object as a long. All digits in the object value that lie to the right of the decimal point will be truncated. It is a user responsibility to ensure that the value of the object does not exceed the range for a long data type.

IDL syntax

```
double getAsLong();
```

ICBCDecimal::getAsPackedDecimal

Returns the value of the object as a packed decimal string.

The format of the returned packed string will one of the following:
[dd]..[ds] if the number of digits is odd
[0d]..[ds] if the number of digits is even

where [] denotes a single byte containing two 4-bit nibbles, d is a four-bit binary digit, 0 is a 4-bit binary zero, and s is a 4-bit sign representation. A hexadecimal value “C” denotes the positive sign, and “D” the negative sign. A sign is always present in the lower order four bits of the rightmost byte.

Parameters

There is no decimal point embedded in the string. Output parameters are updated to describe the precision (aPrecision), and scale (aScale), of the packed string aScale.

IDL syntax

```
string getAsPackedDecimal(inout string aOutput,
                          inout short aPrecision,
                          inout short aScale);
```

Remarks

Because of the presence of the sign, and the packing of two digits per byte, strings with an even number of bytes will have a leading zero padding character.

For the C++ implementation of ICBCDecimal, the application must supply a non-const pointer to a target storage area long enough to receive the packed string, with its sign and leading zero (if present). On return, a null terminator is appended to the end of the string. The function returns a reference to this area. For the Java implementation, this parameter is required, but plays no active role, and it is sufficient to provide a null parameter of type org.omg.CORBA.StringHolder. The function returns a reference to a new object of type String.

ICBCDecimal::getAsShort

Returns the value of the receiving object as a short. All digits in the object value that lie to the right of the decimal point will be truncated. It is a user responsibility to ensure that the value of the object does not exceed the range for a long data type.

IDL syntax

```
double getAsShort();
```

ICBCDecimal::getPreceedingRemainder

Returns a new object containing the remainder value of an immediately preceding division operation. This function may be invoked on an object that has been the subject of a divideThisObjectBy() operation, or on the new object produced by a divideWithNewObject() operation. This function will only return a meaningful result if used immediately after a divide operation. If used following any other manipulation function it will return an object with a zero value.

The precision of the result object is 38.

The scale is: 38 —*p*1 + *s*1

IDL syntax

```
ICBCDecimal getPreceedingRemainder();
```

Remarks

If errors occur that prevent the creation of the new object, a ICBCDecimal pointer with value -1 (for C++) or null (for Java) is returned.

If a division operation has already been performed, this function is a more efficient way of obtaining the remainder than issuing a remainderInThisObject() or remainderWithNewObject(). This function retrieves the remainder that was cached in the object as a by-product of the preceding divide operation, whereas the latter two functions perform an implicit divide to calculate the remainder.

It is the application's responsibility to delete the new remainder object, freeing its memory, when no longer needed.

ICBCDecimal::getPrecision

Returns the precision of the object - the maximum number of digits that make up the value of the object. Precision is conceptually similar to the precision of a DB2 column. For example, an object with precision and scale of 5 and 2 respectively can store numbers up to value 999.99, but could not store numbers 1000.00, nor 999.991, nor 99.991. Precision is set by an ICBCDecimal::initialize..() function, and is generally retained by the object through its lifetime unless changed by arithmetic functions.

IDL syntax

```
short getPrecision();
```

ICBCDecimal::getScale

Returns the scale of the object. Scale, like precision (see "ICBCDecimal::getPrecision"), is set by the ICBCDecimal::initialize..() functions, and is generally retained by the object through its lifetime unless changed by arithmetic operations.

IDL syntax

```
short getScale();
```

ICBCDecimal::greaterThan

Returns TRUE if the value of the receiving ICBCDecimal object is greater than the other ICBCDecimal object.

IDL syntax

```
boolean greaterThan(in ICBCDecimal aDecimal);
```

ICBCDecimal::greaterThanOrEqualTo

Returns TRUE if the value of the receiving ICBCDecimal object is greater than or equal to the other ICBCDecimal object.

IDL syntax

```
boolean greaterThanOrEqualTo(in ICBCDecimal aDecimal);
```

ICBCDecimal::increment

Adds the argument ICBCDecimal object to the receiving ICBCDecimal object and sets the value of the receiving object to the result. The precision and scale of the receiving object are reset. The precision is given by:

min(38, max(*p*1—*s*1, *p*2–*s*2) + max(*s*1, *s*2) + 1)

The scale is given by:

max(*s*1, *s*2)

If the non-zero digits to the left of the decimal point in the result exceed (*p* — *s*) of the result, the receiving object is not changed, and a following invocation of the function decimalOverflow() will return TRUE.

After an increment operation, the change flag of the receiving object is set to TRUE.

IDL syntax

```
ICBCDecimal increment(in ICBCDecimal aDecimal);
```

ICBCDecimal::initializeFromDecimal1

Initializes the receiving object with the value of the argument ICBCDecimal object, and sets its precision and scale to the values of the parameters.

IDL syntax

```
boolean initializeFromDecimal1(in ICBCDecimal aDecimal
                               in short precision,
                               in short scale);
```

Remarks

This function requires parameters to be explicitly specified for the precision and scale of the receiving object. precision must be a number between zero and 38. If zero, a default value of 15 is assumed. scale must be a number between 0 and the value of

precision. If precision is zero, the value of scale is ignored and the scale of the object is taken to be zero. The value of precision - scale must be sufficient to contain the digits to the left of the decimal point in the source ICBCDecimal object. If scale is insufficient to contain the digits to the right of the decimal point, those digits will be truncated from the right.

The difference between this function and ICBCDecimal::assignFromDecimal() is that this function sets the precision and scale of the receiving object, whereas ICBCDecimal::assignFromDecimal() leaves the existing precision and scale of the receiving object unchanged.

If the the object can be successfully initialized from the other Decimal object, TRUE is returned and any existing value in the object is replaced. If the initialization cannot be performed, FALSE is returned and the state of the object is unchanged, except that, if the FALSE return resulted from a numeric overflow, a following invocation of the decimalOverflow() function against the object will return the value TRUE.

ICBCDecimal::initializeFromDecimal2

Initializes the receiving object with the value of the argument ICBCDecimal object, and sets its precision and scale to the values of the parameters.

IDL syntax

```
boolean initializeFromDecimal2(in ICBCDecimal aDecimal);
```

Remarks

This function has no explicit parameters for precision and scale. The precision and scale of the receiving object are set to the same values as the source Decimal object.

The difference between this function and ICBCDecimal::assignFromDecimal() is that this function sets the precision and scale of the receiving object, whereas ICBCDecimal::assignFromDecimal() leaves the existing precision and scale of the receiving object unchanged.

If the the object can be successfully initialized from the other Decimal object, TRUE is returned and any existing value in the object is replaced. If the initialization cannot be performed, FALSE is returned and the state of the object is unchanged, except that, if the FALSE return resulted from a numeric overflow, a following invocation of the decimalOverflow() function against the object will return the value TRUE.

ICBCDecimal::initializeFromDouble

Initializes the receiving object with the value of an argument of type double, and sets its precision and scale.

IDL syntax

```
boolean initializeFromDouble(in double aDouble
                             in short precision,
                             in short scale);
```

Parameters

precision
Precision must be a number between zero and 38. A zero value implies that the application is not supplying values for precision or scale. In this case, a default value of 15 is assumed.

scale Scale must be a number between 0 and the value of precision. If precision is zero, the value of scale is ignored and a value of zero is assumed.

Remarks

If the the object can be successfully initialized with the numeric operand, TRUE is returned and any existing value in the object is replaced. If the initialization cannot be performed, FALSE is returned and the state of the object is unchanged, except that, if the FALSE return resulted from a numeric overflow, a following invocation of the decimalOverflow() function against the object will return the value TRUE.

Precision and scale must be such that (precision - scale) is sufficient to contain the digits to the left of the decimal point in the numeric operand. If scale is insufficient to contain the digits to the right of the decimal point, those digits will be truncated from the right.

ICBCDecimal::initializeFromFloat

Initializes the receiving object with the value of an argument of type float, and sets its precision and scale.

IDL syntax

```
boolean initializeFromFloat(in float aFloat
                            in short precision,
                            in short scale);
```

Parameters

precision
Precision must be a number between zero and 38. A zero value implies that the application is not supplying values for precision or scale. In this case, a default value of 15 is assumed.

scale Scale must be a number between 0 and the value of precision. If precision is zero, the value of scale is ignored and a value of zero is assumed.

Remarks

If the the object can be successfully initialized with the numeric operand, TRUE is returned and any existing value in the object is replaced. If the initialization cannot be performed, FALSE is returned and the state of the object is unchanged, except that, if the FALSE return resulted from a numeric overflow, a following invocation of the decimalOverflow() function against the object will return the value TRUE.

Precision and scale must be such that (precision - scale) is sufficient to contain the digits to the left of the decimal point in the numeric operand. If scale is insufficient to contain the digits to the right of the decimal point, those digits will be truncated from the right.

ICBCDecimal::initializeFromLong

Initializes the receiving object with the value of an argument of type long, and sets its precision and scale.

IDL syntax

```
boolean initializeFromLong(in long aLong
                           in short precision,
                           in short scale);
```

Parameters

precision
Precision must be a number between zero and 38. A zero value implies that the application is not supplying values for precision or scale. In this case, a default value of 11 is assumed.

scale Scale must be a number between 0 and the value of precision. If precision is zero, the value of scale is ignored and a value of zero is assumed.

Remarks

If the the object can be successfully initialized with the numeric operand, TRUE is returned and any existing value in the object is replaced. If the initialization cannot be performed, FALSE is returned and the state of the object is unchanged, except that, if the FALSE return resulted from a numeric overflow, a following invocation of the decimalOverflow() function against the object will return the value TRUE.

Precision and scale must be such that (precision - scale) is sufficient to contain the digits in the numeric operand.

ICBCDecimal::initializeFromPackedDecimal

Accepts a value in the form of a packed decimal string.

IDL syntax

```
boolean initializeFromPackedDecimal(in string aString,
                                    in short aPrecision,
                                    in short aScale);
```

Parameters

aString

must have one of the following forms:
[dd][dd]...[ds] if number of digits is odd
[0d][dd]...[ds] if number of digits is even

Where [] denotes a byte containing two 4-bit "nibbles", *d* is a four-bit binary digit, 0 is a 4-bit binary zero, and s is a 4-bit sign representation. The hexadecimal value "C" is interpreted as the positive sign, and "D" as the negative sign. There is no embedded decimal point. The scale of the object is defined by the scale input argument aScale. The number of digits unpacked from aString is defined by the aPrecision argument. The sign is always present in the lower order four bits of the rightmost byte. Because of the presence of the sign, and the packing of two digits per byte, strings with an even number of bytes will have a leading zero padding character.

This format is consistent with that used by DB2 for packed decimal variables passed to an application program.

precision

This must be a number between zero and 38, and must always be present. There is no default.

scale This parameter must be a number between 0 and the value of precision. It indicates the number of digits that lie to the right of the decimal point. It must always be present. There is no default. A null terminator character on the packed string is not required, but is tolerated if present.

Remarks

If the string is a valid packed decimal representation and the object is successfully initialized, TRUE is returned and any existing value in the object is replaced. Returns FALSE if the parameter is invalid. If FALSE is returned the state of the object is unchanged, except that, if the FALSE return resulted from a numeric overflow, a following invocation of the decimalOverflow() function against the object will return the value TRUE.

ICBCDecimal::initializeFromShort

Initializes the receiving object with the value of an argument of type short, and sets its precision and scale.

IDL syntax

```
boolean initializeFromShort(in short aShort
                            in short precision,
                            in short scale);
```

Parameters

precision
Precision must be a number between zero and 38. A zero value implies that the application is not supplying values for precision or scale. In this case, a default value of 5 is assumed.

scale Scale must be a number between 0 and the value of precision. If precision is zero, the value of scale is ignored and a value of zero is assumed.

Remarks

If the the object can be successfully initialized with the numeric operand, TRUE is returned and any existing value in the object is replaced. If the initialization cannot be performed, FALSE is returned and the state of the object is unchanged, except that, if the FALSE return resulted from a numeric overflow, a following invocation of the decimalOverflow() function against the object will return the value TRUE.

Precision and scale must be such that (precision - scale) is sufficient to contain the digits in the numeric operand.

ICBCDecimal::initializeFromString1

Accepts a value for this decimal object in the form of a string, specified values for precision and scale of the object, and the character that is used in the string as the decimal point (if any).

IDL syntax

```
boolean initializeFromString1(in string aString,
                              in short precision,
                              in short scale,
                              in char decimalChar);
```

Parameters

aString
The parameter aString must be of the form:
[spaces][sign][zeros][digits][decimal point][digits][zeros]&lbrk.spaces]

A negative sign is ignored if the value represented by the string is zero.

decimalChar
decimalChar must be a single character that is used in aString to represent the decimal point. The characters plus (+) and minus (-) are not permitted, and blanks are not permitted.

precision
Precision must be a number between zero and 38. If the value is zero, a default precision of 15 and a default scale of zero are used (the parameter for scale is ignored in this case). Otherwise the parameter values for precision and scale are used.

scale Scale must be a number between 0 and the value of precision. It indicates the number of digits that lie to the right of the decimal point. Note that if the parameter precision is zero, the value of scale is ignored and a default scale of zero is used.

Remarks

If the parameters are valid, TRUE is returned and any existing value in the object is replaced. If the parameter is invalid, FALSE is returned and the state of the object is unchanged, except that, if the FALSE return resulted from a numeric overflow, a following invocation of the decimalOverflow() function against the object will return the value TRUE.

Precision and scale must be such that (precision - scale) is sufficient to contain the non-zero characters to the left of the decimal point in aString. If scale is insufficient to contain the characters to the right of the decimal point, those characters will be truncated from the right.

ICBCDecimal::initializeFromString2

Accepts a value for this decimal object in the form of a string, and the character used in the string as a decimal point.

IDL syntax

```
boolean initializeFromString2(in string aString,
                              in char aDecimalChar);
```

Remarks

The precision and scale of the object are set to the actual precision and scale of the string parameter. The precision is the total number of digits, including leading and trailing zeros. The scale is the number of digits to the right of the decimal point, including trailing zeros. In other respects, this function behaves the same as ICBCDecimal::initializeFromString1().

If the input parameters are valid, TRUE is returned and any existing value in the object is replaced. If the parameter is invalid, FALSE is returned and the state of the object is unchanged, except that, if the FALSE return resulted from a numeric overflow, a following invocation of the decimalOverflow() function against the object will return the value TRUE.

ICBCDecimal::isChanged

Returns a boolean indicating whether the the value of the receiving object has been changed since its initialization. All initialization functions set a changed flag in the object to FALSE. All arithmetic and assignment functions set the flag TRUE. Query functions do not affect the changed flag.

IDL syntax

```
boolean isChanged();
```

ICBCDecimal::isNegative

Returns a boolean indicating whether the the value of the receiving object is less than zero.

IDL syntax

```
boolean isNegative();
```

ICBCDecimal::lessThan

Returns TRUE if the value of the receiving ICBCDecimal object is less than the other ICBCDecimal object.

IDL syntax

```
boolean lessThan(in ICBCDecimal aDecimal);
```

ICBCDecimal::lessThanOrEqualTo

Returns TRUE if the value of the receiving ICBCDecimal object is less than or equal to the other ICBCDecimal object.

IDL syntax

```
boolean lessThanOrEqualTo(in ICBCDecimal aDecimal);
```

ICBCDecimal::multiplyWithNewObject

Multiplies the receiving object by the argument ICBCDecimal object and returns a new ICBCDecimal object containing the result. The original operand objects are unchanged. The precision of the result object is:

min(38, *p*1 + *p*2)

The scale is: min(38, *s*1 + *s*2)

IDL syntax

```
ICBCDecimal multiplyWithNewObject(in ICBCDecimal aDecimal);
```

Remarks

After a multiplyWithNewObject operation, the change flag of the new object is set to TRUE. The change flag of the original object is unchanged.

It is the application's responsibility to delete the new object, freeing its memory, when no longer needed.

If the non-zero digits to the left of the decimal point in the result object would exceed (*p- s*) of the result, or if other errors occur that prevent the creation of the new object, an ICBCDecimal pointer with value -1 (for C++) or null (for Java) is returned.

ICBCDecimal::multiplyThisObjectBy

Multiplies the receiving object by the argument ICBCDecimal object, and sets the receiving object to the result. The precision and scale of the receiving object are reset. The precision is given by:

min(38, *p*1 + *p*2)

The scale is given by:

min(38, *s*l + *s*2)

If the non-zero digits to the left of the decimal point in the results exceed (*p* — *s*) of the result, the object is not changed, and a following invocation of the function decimalOverflow() will return TRUE.

The change flag of the receiving object is set to TRUE.

IDL syntax

```
ICBCDecimal multiplyThisObjectBy(in ICBCDecimal aDecimal);
```

ICBCDecimal::notEqualTo

The opposite of the operator== member.

IDL syntax

```
booleans notEqualTo(in ICBCDecimal aDecimal);
```

ICBCDecimal::remainderInThisObject

Divides the receiving object by the argument ICBCDecimal object, and sets the receiving object to the remainder of the division operation. The precision and scale of the receiving object are reset.

The change flag of the receiving object is set to TRUE.

The precision is: 38.

The scale is: 38 - *p*1 + *s*1

IDL syntax

```
ICBCDecimal remainderInThisObject(in ICBCDecimal aDecimal);
```

ICBCDecimal::remainderWithNewObject

Divides the receiving object by the argument ICBCDecimal object and returns a new ICBCDecimal object containing the remainder. The original operand objects are unchanged.

The precision of the result object is 38.

The scale is: 38 —*p*1 + *s*1

IDL syntax

```
ICBCDecimal remainderWithNewObject(in ICBCDecimal aDecimal);
```

Remarks

This function does not cause any overflow conditions, by may encounter a zeroDivide situation. If so, or if errors occurred that prevented the creation of the new object, a ICBCDecimal pointer with value -1 (for C++) or null (for Java) is returned.

After a remainderWithNewObject operation, the change flag of the new object is set to TRUE. The change flag of the original object is unchanged.

It is the application's responsibility to delete the new object, freeing its memory, when no longer needed.

ICBCDecimal::subtractWithNewObject

Subtracts the argument ICBCDecimal object from the receiving object and returns a new ICBCDecimal object containing the result. The original operand objects are unchanged. The precision of the result object is:

min(38, max(*p*1–*s*1, *p*2–*s*2) + max(*s*1, *s*2) + 1)

The scale is: max(*s*1, *s*2)

IDL syntax

```
ICBCDecimal subtractWithNewObject(in ICBCDecimal aDecimal);
```

Remarks

After a subtractWithNewObject operation, the change flag of the new object is set to TRUE. The change flag of the original object is unchanged.

It is the application's responsibility to delete the new object, freeing its memory, when no longer needed.

If the non-zero digits to the left of the decimal point in the result object would exceed (*p* - *s*) of the result, or if other errors occur that prevent the creation of the new object, an ICBCDecimal pointer with value -1 (for C++) or null (for Java) is returned.

ICBCDecimal::swapSign

This function is effectively a unary minus operator which reverses the sign of this ICBCDecimal object.

IDL syntax

```
void swapSign();
```

ICBCDuration Class

The class ICBCDuration represents the "Labeled Duration", "Date Duration", "Time Duration", and "Timestamp Duration" constructs of DB2. Duration is a complementary data type to Date, Time, and Timestamp, and participates in SQL expressions. However, it is not, itself, stored in DB2 tables.

The class holds the value of a duration, and a record of whether this duration value is in units of years, months, days, hours, minutes seconds, or microseconds. The value can also be of a composite type defining a "Date Duration" of the form yyyymmdd, a "Time Duration" of the form hhmmss, or a "Timestamp Duration" of the form yyyymmddhhmmsszzzzzz.

Arithmetic between two ICBCDuration classes is not supported (This is consistent with DB2 support of Durations). However, they can be compared, and a unary minus operator (which swaps the sign) is provided.

IDL interface description

```
interface ICBCDuration :IManagedLocal::ILocalOnly
{
  enum durationType {YEARS, MONTHS, DAYS, HOURS, MINUTES, SECONDS,
```

```
        MICROSECONDS, DATE, TIME, TIMESTAMP};

    // Object initialization
    // 1. Initializer for YEAR, MONTH, DAY, HOUR, MINUTE, SECOND
    // and MICROSECOND durations

    boolean    initializeLabeledDuration(in long aDuration,
                           in ICBCDuration::durationType aType);

    //Initializer for DATE and TIME durations
    boolean    initializeDateTimeDuration(in long aValue1, in long aValue2,
                           in long aValue3,
                           in boolean aNegativeFlag,
                           in ICBCDuration::durationType aType);
    //Initializer for TIMESTAMP durations
    boolean    initializeTimestampDuration(in long aValue1, in long aValue2,
                           in long aValue3, in long aValue4,
                           in long aValue5, in long aValue6,
                           in long aValue7,
                           in boolean aNegativeFlag);

    /* Comparisons */
    boolean  equalTo(in ICBCDuration aDuration);
    boolean  notEqualTo(in ICBCDuration aDuration);
    boolean  lessThan(in ICBCDuration aDuration);
    boolean  lessThanOrEqualTo(in ICBCDuration aDuration);
    boolean  greaterThan(in ICBCDuration aDuration);
    boolean  greaterThanOrEqualTo(in ICBCDuration aDuration);

    /* Query functions */
    ICBCDuration::durationType getType();
    boolean  isNegative();
    long     microsecond();
    long     second();
    long     minute();
    long     hour();
    long     day();
    long     month();
    long     year();
    long     size(in long aIntervalType);
    string   formattedString(inout string aString);

    /* Manipulation functions */
    void     assignFromDuration(in ICBCDuration aDuration);
    void     swapSign();        // unary minus
};
```

Supported methods

ICBCDuration::_create
ICBCDuration::initializeLabeledDuration
ICBCDuration::initializeDateTimeDuration
ICBCDuration::initializeTimestampDuration
ICBCDuration::equalTo
ICBCDuration::notEqualTo

ICBCDuration::lessThan
ICBCDuration::lessThanOrEqualTo
ICBCDuration::greaterThan
ICBCDuration::greaterThanOrEqualTo
ICBCDuration::formattedString
ICBCDuration::getType
ICBCDuration::isNegative
ICBCDuration::microSecond
ICBCDuration::second
ICBCDuration::minute
ICBCDuration::hour
ICBCDuration::day
ICBCDuration::month
ICBCDuration::year
ICBCDuration::assignFromDuration
ICBCDuration::size
ICBCDuration::swapSign

ICBCDuration::assignFromDuration

Sets all state data of this ICBCDuration object equal to the the other ICBCDuration object.

IDL syntax

```
void assignFromDuration(in ICBCDuration aDuration);
```

ICBCDuration::_create

Returns a pointer to a new ICBCDuration object. The object is initialized to a type of “YEARS” and a value of zero.

ICBCDuration::day

Returns the day part of the ICBCDuration.

IDL syntax

```
long day();
```

ICBCDuration::equalTo

Returns TRUE if all state data of this ICBCDuration object is equal to the other ICBCDuration object.

IDL syntax

```
boolean equalTo(in ICBCDuration aDuration);
```

ICBCDuration::formattedString

Returns the value of a DATE type duration as a string of the form yyyymmdd, a TIME type duration in the form hhmmss, or a TIMESTAMP duration in the form yyyymmddhhmmssmmmmmm. For other duration types this function returns a null string.

IDL syntax

```
string formattedString(inout string aString);
```

For the C++ implementation of ICBCDuration, the application must supply a non-const pointer to a target storage area long enough to receive the formatted string. The function returns a reference to this area. For the Java implementation, this parameter is required, but plays no active role. It is sufficient to provide a null parameter of type org.omg.CORBA.StringHolder.

ICBCDuration::greaterThan

Returns TRUE if the value of this ICBCDuration object is greater than the other ICBCDuration object.

IDL syntax

```
boolean greaterThan(in ICBCDuration aDuration);
```

ICBCDuration::greaterThanOrEqualTo

Returns TRUE if the value of this ICBCDuration object is greater than or equal to the other ICBCDuration object.

IDL syntax

```
boolean greaterThanOrEqualTo(in ICBCDuration aDuration);
```

ICBCDuration::getType

Returns the type of this ICBCDuration. Returned values can be DATE, TIME, YEARS, MONTHS, DAYS, HOURS, MINUTES, SECONDS, MICROSECONDS.

IDL syntax

```
ICBCDuration::durationType getType();
```

ICBCDuration::hour

Returns the hour part of the ICBCDuration.

IDL syntax

```
long hour();
```

ICBCDuration::initializeDateTimeDuration

Accepts three numeric values, an indicator of the sign of the duration (positive or negative) and an indicator of the duration type.

IDL syntax

```
boolean initializeDateTimeDuration(in long aValue1,
                                   in long aValue2,
                                   in long aValue3,
                                   in boolean aNegativeFlag,
                                   in ICBCDuration::durationType aType);
```

If the type indicator is DATE, then aValue1, aValue2, aValue3 must supply valid values for years, months, and days, respectively. Each value must be a positive number, or zero. If the supplied parameters are valid, returns TRUE, else returns FALSE.

If type indicator is TIME, then aValue1, aValue2, aValue3 must supply valid values for hours, minutes, and seconds, respectively.

Each value must be a positive number, or zero. If the supplied parameters are valid, returns TRUE, else returns FALSE.

ICBCDuration::initializeLabeledDuration

Accepts a single numeric value, and an indicator of the type of duration.

IDL syntax

```
boolean initializeLabeledDuration(in long aDuration,
                                  in ICBCDuration::durationType aType);
```

The type indicator can be: YEARS, MONTHS, DAYS, HOURS, MINUTES, SECONDS, MICROSECONDS. The supplied values must be a positive number or zero.If the numeric value is valid, returns TRUE, otherwise returns FALSE. If the values are valid, any existing value in the object is replaced.

ICBCDuration::initializeTimestampDuration

Accepts seven numeric values, and an indicator of the sign of the duration (positive or negative).

IDL syntax

```
boolean initializeTimestampDuration(in long aValue1,
                                    in long aValue2,
                                    in long aValue3,
                                    in long aValue4,
                                    in long aValue5,
                                    in long aValue6,
                                    in long aValue7,
                                    in boolean aNegativeFlag);
```

The numeric values represent the years, months, days, hours, minutes, seconds, and microseconds portions of the timestamp duration. They must be positive numbers or zeros. If the supplied parameters are valid, returns TRUE, else returns FALSE.

ICBCDuration::isNegative

Returns a Boolean TRUE is this object's value is negative. Otherwise returns FALSE;

IDL syntax

```
boolean isNegative();
```

ICBCDuration::lessThan

Returns TRUE if the value of this ICBCDuration object is less than the other ICBCDuration object.

IDL syntax

```
boolean lessThan(in ICBCDuration aDuration);
```

ICBCDuration::lessThanOrEqualTo

Returns TRUE if the value of this ICBCDuration object is less than or equal to the other ICBCDuration object.

IDL syntax

```
boolean lessThanOrEqualTo(in ICBCDuration aDuration);
```

ICBCDuration::microSecond

Returns the microseconds part of the ICBCDuration.

IDL syntax

```
long microSecond();
```

ICBCDuration::minute

Returns the minute part of the ICBCDuration.

IDL syntax

```
long minute();
```

ICBCDuration::month

Returns the month part of the ICBCDuration.

IDL syntax

```
long month();
```

ICBCDuration::notEqualTo

The opposite of the equalTo operation.

IDL syntax

```
boolean notEqualTo(in ICBCDuration aDuration);
```

ICBCDuration::second

Returns the seconds part of the ICBCDuration.

IDL syntax

```
long second();
```

ICBCDuration::size

Equivalent to the DB2 TIMESTAMPDIFF built-in function.

IDL syntax

```
long size(in long aIntervalType);
```

Remarks

Returns the value of the timestamp duration converted to the interval type specified in the input parameter. If the parameter is not valid, a value of -1 is returned. Valid parameter values, and associated interval types are:

1 Fractions of a second.
2 Seconds.
4 Minutes.
8 Hours.
16 Days.
32 Weeks.
64 Months.
128 Quarters.
256 Years.

The conversion is not precise. A standard length year and month are assumed, taking no account of leap years. Conversion factors used are as follows:

365 days per year.
30 days per month.
24 hours per day.
60 minutes per hour.
60 seconds per minute.

ICBCDuration::swapSign

This function is effectively a unary minus operator which reverses the sign of this ICBCDuration object.

IDL syntax

```
void swapSign();
```

ICBCDuration::year

Returns the year part of the ICBCDuration.

IDL syntax

```
long year();
```

ICBCTime Class

The class stores the value of a time-of-day. All access and manipulation of this time is through a public functional interface. The actual time value and its format are stored in private attributes.

IDL interface description

```
interface ICBCTime :IManagedLocal::ILocalOnly
  {
  /* Object initialization (DB2 semantics)*/
  boolean  initializeFromString(in string aTime);
  boolean  initializeFromTimestamp(in ICBCTimestamp aTimestamp);
  boolean  initializeFromValues(in long aHour,
                                in long aMinute,
                                in long aSecond);

  // Comparisons
  boolean  equalTo(in ICBCTime aTime);
  boolean  notEqualTo(in ICBCTime aTime);
  boolean  lessThan(in ICBCTime aTime);
  boolean  lessThanOrEqualTo(in ICBCTime aTime);
  boolean  greaterThan(in ICBCTime aTime);
  boolean  greaterThanOrEqualTo(in ICBCTime aTime);

  // Query functions (DB2 semantics)
  enum     OutputFormat {ISO, USA, EUR, JIS, LOC};
  string   formattedString(inout string aOutput,
              in ICBCTime::OutputFormat aFormat);
  long     second();
  long     minute();
  long     hour();
  boolean  isObjectChanged();

  // Manipulation functions (DB2 semantics)
  void     assignFromTime(in ICBCTime aTime);
  void     increment(in ICBCDuration aDuration);
  void     decrement(in ICBCDuration aDuration);
  long     timeOverflow();

  ICBCDuration intervalFromTime(in ICBCTime aTime);
  ICBCDuration intervalFromString(in string aTime);
};
```

Supported methods

ICBCTime::_create
ICBCTime::initializeFromString
ICBCTime::initializeFromTimestamp
ICBCTime::initializeFromValues
ICBCTime::equalTo
ICBCTime::notEqualTo
ICBCTime::lessThan
ICBCTime::lessThanOrEqualTo
ICBCTime::greaterThan
ICBCTime::greaterThanOrEqualTo
ICBCTime::formattedString
ICBCTime::second
ICBCTime::minute
ICBCTime::hour
ICBCTime::isObjectChanged

ICBCTime::assignFromTime
ICBCTime::increment
ICBCTime::decrement
ICBCTime::timeOverflow
ICBCTime::interval

ICBCTime::assignFromTime

Sets all state data in this ICBCTime object equal to the second ICBCTime object. The change flag of this object is set to TRUE.

IDL syntax

```
void assignFromTime(in ICBCTime aTime);
```

ICBCTime::_create

Returns a pointer to a new ICBCTime object. The object is initialized to the current local time.

ICBCTime::decrement

Subtracts a ICBCDuration from the value of this ICBCTime object.

IDL syntax

```
void decrement(in ICBCDuration aDuration);
```

Remarks

This operation is performed essentially as the inverse of the increment operation.

When a duration of type TIME is subtracted from a time, the time is decremented by the specified number of hours, minutes, and seconds, in that order.

Incrementing with a ICBCDuration of type YEARS, MONTHS, DAYS, DATE, or TIMESTAMP will have no effect on an object of type ICBCTime.

If the hours value of the result of a decrement operation is negative, then enough whole periods of 24 hours are added to the value to make it positive. The number of 24 hour periods (i.e., days) thus added can be retrieved as a negative number using the timeOverflow() function.

Although a ICBCTime object with a value (expressed in ISO format) of 24.00.00 can be input to an increment or decrement operation, a time of 24.00.00 will never be returned as the result of the operation. Thus 24.00.00 plus or minus a zero duration will return the time 00.00.00.

The change flag of this object is set to TRUE.

ICBCTime::equalTo

Returns TRUE if the time in this ICBCTime object is equal to the other ICBCTime object.

Note: Although 24.00.00 and 00.00.00 both represent midnight, ICBCTime considers that 24.00.00 > 00.00.00.

IDL syntax

```
boolean equalTo(in ICBCTime aTime);
```

ICBCTime::formattedString

Returns the time value of this object as a string in a specified format. The parameter OutputType must take one the values: ISO, USA, EUR, JIS. Formats are as described for the function ICBCTime::initializeFromString(). The LOC site-defined option described in the DB2 manual is not supported by ICBCTime.

IDL syntax

```
string formattedString(inout string aOutput,
                       in ICBCTime::OutputFormat aFormat);
```

For the C++ implementation of ICBCTime, the application must supply a non-const pointer to a target storage area long enough to receive the formatted string. The function returns a reference to this area. For the Java implementation, this parameter is required, but plays no active role. It is sufficient to provide a null parameter of type org.omg.CORBA.StringHolder.

ICBCTime::greaterThan

Returns TRUE if the time stored in this ICBCTime object is greater than the other ICBCTime object.

IDL syntax

```
boolean greaterThan(in ICBCTime aTime);
```

ICBCTime::greaterThanOrEqualTo

Returns TRUE if the time stored in this ICBCTime object is greater than or equal to the other ICBCTime object.

IDL syntax

```
boolean greaterThanOrEqualTo(in ICBCTime aTime);
```

ICBCTime::hour

Returns the hour part of the time.

IDL syntax

```
long hour();
```

ICBCTime::increment

Adds a ICBCDuration to the value of this ICBCTime object.

IDL syntax

```
void increment(in ICBCDuration aDuration);
```

Remarks

If a duration of type HOURS is added to a time, only the hours portion of the time is affected. The minutes and seconds are unchanged.

Similarly, if a duration of type MINUTES is added to the time, only minutes and, if necessary, hours are affected.

Adding a duration of type SECONDS will, of course, affect the seconds portion of the time, and potentially the minutes and hours.

When a duration of type TIME is added to a time, the time is incremented by the specified number of hours, minutes, and seconds, in that order.

Incrementing with a ICBCDuration of type YEARS, MONTHS, DAYS, DATE, or TIMESTAMP will have no effect on an object of type ICBCTime.

If the hours value of the result of an increment operation is greater than 24 hours, then the hours value is divided by 24 and the remainder stored in the hours value. The quotient of the division (the number of whole 24 hour periods or days in the original hours value) can be retrieved using the timeOverflow() function.

Although a ICBCTime object with a value (expressed in ISO format) of 24.00.00 can be input to an increment or decrement operation, a time of 24.00.00 will never be returned as the result of the operation. Thus 24.00.00 plus or minus a zero duration will return the time 00.00.00.

The change flag of this object is set to TRUE.

ICBCTime::initializeFromString

Accepts a value for the time in the form of a string. Any existing value in the object is replaced. Returns TRUE if the string is a valid time representation and the object is successfully initialized. Returns FALSE if the parameter is invalid. If FALSE is returned the state of the object is unchanged.

IDL syntax

```
boolean initializeFromString(in string aTime);
```

Parameters

aTime The string must be either:

- A valid string representation of a time, in one of the following valid forms.
  - International Standards Organization (ISO) (hh.mm.ss).
  - IBM USA standard (USA) (hh:mm AM or PM).
  - IBM European standard (EUR) (hh.mm.ss).
  - Japanese Industrial Standard Christian Era (JIS) (hh:mm:ss).

  Leading and trailing blanks are acceptable provided the string length is less than 20 characters. Leading zeros can be omitted from the hours portion, and in ISO, EUR and JIS the seconds portion can be omitted completely. In the USA format, the minutes can be omitted. Zeros are assumed for omitted portions. In the USA time format, the hour cannot be greater than 12 and cannot be 0 except for the special case of 00:00 AM. Only SBCS characters may be present in the string.
- A string representation of a timestamp.

  Any of the string forms acceptable by the ICBCTimestamp::initializeFromString() function are acceptable. For details refer to that function. Briefly, the forms include:
  - A punctuated string: yyyy-mm-dd-hh.mm.ss.nnnnnn.
  - A 14 byte string without punctuation, of the form: yyyymmddhhmmss.

ICBCTime::initializeFromTimestamp

Initializes this object from the time portion of a Timestamp object.

IDL syntax

```
boolean initializeFromTimestamp(in ICBCTimestamp aTimestamp);
```

ICBCTime::initializeFromValues

Accepts the time in the form of three separate value parameters for hour, minute, and second.

The values must make a valid time.

Returns TRUE if the values are a valid time representation, FALSE otherwise. If FALSE is returned, the state of the object is unchanged.

IDL syntax

```
boolean initializeFromValues(in long aHour, in long aMinute,
                             in long aSecond);
```

ICBCTime::interval

Calculates the Duration between two ICBCTime objects. The result is a pointer to a new ICBCDuration object of type TIME.

If the interval between TIME1 and TIME2 is required, and TIME1 is greater than or equal to TIME2, the operation is performed by subtracting TIME2 from TIME1. If, however, TIME1 is less than TIME2, TIME1 is subtracted from TIME2, and the sign of the result is made negative.

The following procedural description clarifies the steps involved in the operation result = TIME1 - TIME2.

```
If SECOND(TIME2) <= SECOND(TIME1)
   then SECOND(RESULT) = SECOND(TIME1) - SECOND(TIME2).
If SECOND(TIME2) > SECOND(TIME1)
   then SECOND(RESULT) = 60 + SECOND(TIME1) - SECOND(TIME2).
        MINUTE(TIME2) is then incremented by 1.
If MINUTE(TIME2) <= MINUTE(TIME1)
   then MINUTE(RESULT) = MINUTE(TIME1) - MINUTE(TIME2).
If MINUTE(TIME2) > MINUTE(TIME1)
   then MINUTE(RESULT) = 60 + MINUTE(TIME1) - MINUTE(TIME2).
        HOUR(TIME2) is then incremented by 1.
HOUR(RESULT) = HOUR(TIME1) - HOUR(TIME2).
```

IDL syntax

```
ICBCDuration_ptr intervalFromTime(in ICBCTime aTime);
ICBCDuration_ptr intervalFromString(in string aTime);
```

In the case with a string parameter, the parameter must be a valid string representation of a time. For details see “ICBCTime::initializeFromString” on page 915.

If the string parameter to this function cannot be recognized as a valid time, then a pointer value of -1 (for C++) or null (for Java) is returned.

ICBCTime::isObjectChanged

Returns a boolean indicating whether the value of this object has been changed since it was initialized.

IDL syntax

```
boolean isObjectChanged();
```

ICBCTime::lessThan

Returns TRUE if the time stored in this ICBCTime object is less than the other ICBCTime object.

IDL syntax

```
boolean lessThan(in ICBCTime aTime);
```

ICBCTime::lessThanOrEqualTo

Returns TRUE if the time stored in this ICBCTime object is less than or equal to the other ICBCTime object.

IDL syntax

```
boolean lessThanOrEqualTo(in ICBCTime aTime);
```

ICBCTime::minute

Returns the minute part of the time.

IDL syntax

```
long minute();
```

ICBCTime::notEqualTo

The opposite of the operator== member.

IDL syntax

```
boolean notEqualTo(in ICBCTime aTime);
```

ICBCTime::second

Returns the second part of the time.

IDL syntax

```
long second();
```

ICBCTime::timeOverflow

Returns the number of overflow or underflow days of a preceding increment or decrement operation. For an increment, returns the number of whole 24 hour periods (days) in the hours value of the result. For a decrement, returns the number of 24 hour periods necessary to restore the hours value of the result to a positive value.

IDL syntax

```
long timeOverflow();
```

ICBCTimestamp Class

The class stores the value of a DB2 Timestamp or an Oracle DATE type. All access and manipulation of this timestamp is through a public functional interface. The actual timestamp value is stored in a private attributes.

A Timestamp attribute in a Component Broker business object that originates from a DB2 database is represented by Component Broker as a 26 character string of form yyyy-mm-dd-hh.mm.ss.mmmmmm This string can be used to initialize an ICBCTimestamp instance through the ICBCTimestamp::initializeFromString function. A DATE attribute that originates from an Oracle database is represented by Component Broker in the same 26 character form yyyy-mm-dd-hh.mm.ss.mmmmmm, but with the final six characters (the microseconds field) always set to zeros. This string can be used to initialize an ICBCTimestamp via ICBCTimestamp::initializeFromString. However, Oracle dates earlier than 0001-01-01 are rejected. Whether the ICBCTimestamp object is initialized from data originating in DB2 or in Oracle, ICBCTimestamp applies DB2 semantics in its processing of the data.

IDL interface description

```
interface ICBCTimestamp :IManagedLocal::ILocalOnly
      {

        // Object initialization
  boolean     initializeFromString(in string aTimestamp);
  boolean     initializeFromDateTime(in ICBCDate aDate,
                                     in ICBCTime aTime,
                                     in long aMicroseconds);
  void        initializeFromDate(in ICBCDate aDate);  // Equivalent to
                                                      // TIMESTAMP_ISO
  void        initializeFromTime(in ICBCTime aTime);  // Equivalent to
                                                      // TIMESTAMP_ISO

  enum        DatastoreType {ORCL};
  boolean     initializeFromDatastoreFormat(in string aTimestamp,
                 in ICBCTimestamp::DatastoreType aDatastore);

        // Comparisons
  boolean     equalTo(in ICBCTimestamp aTimestamp);
  boolean     notEqualTo(in ICBCTimestamp aTimestamp);
```

```
boolean     lessThan(in ICBCTimestamp aTimestamp);
boolean     lessThanOrEqualTo(in ICBCTimestamp aTimestamp);
boolean     greaterThan(in ICBCTimestamp aTimestamp);
boolean     greaterThanOrEqualTo(in ICBCTimestamp aTimestamp);

      // Query functions
enum        OutputFormat {ISO, USA, EUR, JIS, LOC};

// gets whole timestamp as a string:
string      formattedString(inout string aOutput);

string  getAsDateFormattedString(inout string aTarget,
                                 in ICBCTimestamp::OutputFormat aFormat);
string  getAsTimeFormattedString(inout string aTarget,
                                 in ICBCTimestamp::OutputFormat aFormat);

string  getAsDatstoreFormat(inout string aTarget,
                            in ICBCTimestamp::DatastoreType aDatastore);

long         microsecond();
long         second();
long         minute();
long         hour();
long         day();
long         month();
long         year();

long        dayOfWeek();
long        dayOfYear();
long        julianDay();
long        modifiedJulianDay();
string      dayName();

long        dayFromString(in string aTimestamp);
long        dayOfWeekFromString(in string aTimestamp);
long        dayOfYearFromString(in string aTimestamp);
long        julianDayFromString(in string aTimestamp);
long        modifiedJulianDayFromString(in string aTimestamp);
string      dayNameFromString(in string aTimestamp);
string      dayNameFromNumber(in long aDayIndex);

string      monthName();

long        monthFromString(in string aTimestamp);
string      monthNameFromString(in string aTimestamp);
string      monthNameFromNumber(in long aMonthIndex);

long        quarter();
long        quarterFromString(in string aTimestamp);

long        yearFromString(in string aTimestamp);
boolean     isLeapYear(in long aYear);

long        daysInObject();
long        daysInString(in string aTimestamp);
```

```
    long        daysInMonth(in long aYear, in long aMonth);
    long        daysInYear(in long aYear);

    boolean     isObjectChanged();


          // Manipulation functions
    void         assignFromTimestamp(in ICBCTimestamp aTimestamp);

    void         increment(in ICBCDuration aDuration);
    void         decrement(in ICBCDuration aDuration);
    long         timestampOverflow();


    ICBCDuration intervalFromTimestamp(in ICBCTimestamp aTimestamp);
    ICBCDuration intervalFromString(in string aTimestamp);
};
```

Supported methods

ICBCTimestamp::_create
ICBCTimestamp::initializeFromString
ICBCTimestamp::initializeFromDateTime
ICBCTimestamp::initializeFromDate
ICBCTimestamp::initializeFromTime
ICBCTimestamp::equalTo
ICBCTimestamp::notEqualTo
ICBCTimestamp::lessThan
ICBCTimestamp::lessThanOrEqualTo
ICBCTimestamp::greaterThan
ICBCTimestamp::greaterThanOrEqualTo
ICBCTimestamp::formattedString
ICBCTimestamp::microsecond
ICBCTimestamp::second
ICBCTimestamp::minute
ICBCTimestamp::hour
ICBCTimestamp::day
ICBCTimestamp::month
ICBCTimestamp::year
ICBCTimestamp::dayOfWeek
ICBCTimestamp::dayOfYear
ICBCTimestamp::julianDay
ICBCTimestamp::modifiedJulianDay
ICBCTimestamp::dayName
ICBCTimestamp::dayFromString
ICBCTimestamp::dayOfWeekFromString
ICBCTimestamp::dayOfYearFromString
ICBCTimestamp::julianDayFromString
ICBCTimestamp::modifiedJulianDayFromString
ICBCTimestamp::dayNameFromString
ICBCTimestamp::dayNameFromNumber
ICBCTimestamp::monthName

ICBCTimestamp::monthFromString
ICBCTimestamp::monthNameFromString
ICBCTimestamp::monthNameFromNumber
ICBCTimestamp::quarter
ICBCTimestamp::quarterFromString
ICBCTimestamp::yearFromString
ICBCTimestamp::isLeapYear
ICBCTimestamp::daysInObject
ICBCTimestamp::daysInString
ICBCTimestamp::daysInMonth
ICBCTimestamp::daysInYear
ICBCTimestamp::isObjectChanged
ICBCTimestamp::assignFromTimestamp
ICBCTimestamp::increment
ICBCTimestamp::decrement
ICBCTimestamp::timestampOverflow
ICBCTimestamp::interval

ICBCTimestamp::assignFromTimestamp

Sets all state data in this ICBCTimestamp object equal to the the second ICBCTimestamp object. The change flag of this object is set to TRUE.

IDL syntax

```
ICBCTimestamp assignFromTimestamp(in ICBCTimestamp aTimestamp);
```

ICBCTimestamp::_create

Returns a pointer to a new ICBCTimestamp object. The object is initialized to today's date and the current local time with a microseconds value of zero.

ICBCTimestamp::day

Returns the day part of the timestamp.

IDL syntax

```
long day();
```

ICBCTimestamp::dayFromString

Returns the day part of the timestamp encoded in the input string parameter. The input string parameter must conform to a format recognizable as a timestamp. Refer to ICBCTimestamp::initializeFromString() for a description of supported formats.

If the input parameter is not recognizable, a value of -1 is returned.

IDL syntax

```
long dayFromString(in string aString);
```

ICBCTimestamp::dayName

Returns a string containing the name of the day of the week for this timestamp object.

IDL syntax

```
long dayName();
```

ICBCTimestamp::dayNameFromNumber

Returns a string containing the name of the day of the week for the day number within a week provided in the input parameter. The input parameter must be a number between 1 and 7. Sunday is day number 1.

If the input parameter is not recognizable, an empty string is returned.

IDL syntax

```
long dayNameFromNumber(in long aDayIndex);
```

ICBCTimestamp::dayNameFromString

Returns a string containing the name of the day of the week for for the timestamp encoded in the input string parameter. The input string parameter must conform to a format recognizable as a timestamp. Refer to ICBCTimestamp::initializeFromString() for a description of supported formats.

If the input parameter is not recognizable, an empty string is returned.

IDL syntax

```
long dayNameFromString(in string aTimestamp);
```

ICBCTimestamp::dayOfWeek

Returns a number between 1 and 7 representing the day of the week for this timestamp object. Sunday is day number 1.

IDL syntax

```
long dayOfWeek();
```

ICBCTimestamp::dayOfWeekFromString

Returns a number between 1 and 7 representing the day of the week for the timestamp encoded in the input string parameter. Sunday is day number 1.

The input string parameter must conform to a format recognizable as a timestamp. Refer to ICBCtimestamp::initializeFromString() for a description of supported formats.

If the input parameter is not recognizable, a value of -1 is returned.

IDL syntax

```
long dayOfWeekFromString(in string aString);
```

ICBCTimestamp::dayOfYear

Returns a number between 1 and 366 representing the day of the year for this timestamp object. January 1 is day number 1.

IDL syntax

```
long dayOfYear();
```

ICBCTimestamp::dayOfYearFromString

Returns a number between 1 and 366 representing the day of the year for the timestamp encoded in the input string parameter. January 1 is day number 1.

The input string parameter must conform to a format recognizable as a timestamp. Refer to ICBCTimestamp::initializeFromString() for a description of supported formats.

If the input parameter is not recognizable, a value of -1 is returned.

IDL syntax

```
long dayOfYearFromString(in string aString);
```

ICBCTimestamp::daysInMonth

Returns the number of days in the month of the year provided in the input parameters. Month number must be between 1 and 12. If the input parameters are not valid, a value of -1 is returned.

IDL syntax

```
long daysInMonth(in long aYear, in long aMonth);
```

ICBCTimestamp::daysInObject

The result is the number of days from January 1, 0001 to the timestamp represented by this object, plus one day. i.e., an object holding a timestamp for date January 2 0001 returns the value 2.

IDL syntax

```
long daysInObject();
```

ICBCTimestamp::daysInString

The result is the number of days from January 1, 0001 to the timestamp represented by the string parameter, plus one day. i.e., an object holding a timestamp for date 00010102 returns the value 2.

If the string is invalid, returns a value of -1.

IDL syntax

```
long daysInString(in string aTimestamp);
```

The input string parameter must conform to a format recognizable as a timestamp. Refer to ICBCtimestamp::initializeFromString() for a description of supported formats. If the input parameter is not recognizable, a value of -1 is returned.

ICBCTimestamp::daysInYear

Returns the number of days in the year provided in the input parameter. If the input parameter is not valid, a value of -1 is returned. Returns the number of days in the year provided in the input parameter.

If the input parameter is not valid, a value of -1 is returned.

IDL syntax

```
long daysInYear(in long aYear);
```

ICBCTimestamp::decrement

Subtracts a ICBCDuration from the value of this ICBCTimestamp object. No value is returned (i.e., this is effectively a -= operator).

Decrementing of a timestamp with a duration (of any type) is carried out on the date and time portions of the timestamp as described for ICBCDate and ICBCTime, except that any underflow of microseconds is satisfied from the seconds portion of the result timestamp, and any underflow of hours is satisfied from the days portion.

If the years value of the result of an decrement operation is less than 0001 years, then the years value of the result is set to 0001, and the number of “underflow” years can be retrieved as a negative number with the dateOverflow() function.

The change flag of this object is set to TRUE.

Although an ICBCTimestamp object with an time portion equal to 24.00.00 can be initialized, or returned from an ICBCTimestamp::interval operation, and can be input to an increment or decrement operation, such a timestamp is never be returned as a result of the increment or decrement operation. For the purposes of the calculation, the time portion of the timestamp is considered to be 00.00.00, with the days part incremented by 1. For example, a timestamp value of 1998-11-2-24.00.00.000000 plus or minus a zero duration will return the timestamp result 1998-11-3-00.00.00.000000.

IDL syntax

```
void decrement(in ICBCDuration aDuration);
```

ICBCTimestamp::equalTo

Returns TRUE if the date and time this ICBCTimestamp object is equal to the other ICBCTimestamp object.

Note that because of the alternative representations of midnight it is possible to have two different timestamps that actually represent the same point in time, for example: 1997-11-02-24.00.00.000000 and 1997-11-03-00.00.00.000000. ICBCTimestamp compares timestamps by date portion, followed by time portion, and will consider that for timestamp pairs of this type, the timestamp with the higher day part is greater. e.g.: 1997-11-03-00.00.00.000000 > 1997-11-02-24.00.00.000000.

IDL syntax

```
boolean equalTo(in ICBCTimestamp aTimestamp);
```

ICBCTimestamp::formattedString

Returns the value of this timestamp object as a string in the form yyyy-mm-dd-hh.mm.ss.mmmmmm.

For the C++ implementation of ICBCTimestamp, the application must supply a non-const pointer to a target storage area long enough to receive the formatted string. The function returns a reference to this area. For the Java implementation, this parameter is required, but plays no active role. It is sufficient to provide a null parameter of type org.omg.CORBA.StringHolder.

IDL syntax

```
string formattedString(inout string aOutput);
```

ICBCTimestamp::getAsDatastoreFormat

Returns this timestamp object formatted in the internal format of some datastore. Parameter aDatastore must be set to the type of datastore that is to receive the string. The only accepted value at present is ICBCTimestamp::ORCL, for Oracle. The microseconds part of the timestamp is truncated.

For the C++ implementation of ICBCTimestamp, the application must supply a non-const pointer to a target storage area long enough to receive the formatted string. The function returns a reference to this area. For the Java implementation, this parameter is required, but plays no active role. It is sufficient to provide a null parameter of type org.omg.CORBA.StringHolder.

IDL syntax

```
  string  getAsDatstoreFormat(inout string aTarget,
in ICBCTimestamp::DatastoreType aDatastore);
```

ICBCTimestamp::getAsDateFormattedString

Returns the date portion of this timestamp object formatted according to the parameter aFormat. aFormat must be one of the enumerated values ISO, USA, EUR, JIS. For details of date formats, see “ICBCDate::formattedString” on page 871.

For the C++ implementation of ICBCTimestamp, the application must supply a non-const pointer to a target storage area long enough to receive the formatted string. The function returns a reference to this area. For the Java implementation, this parameter is required, but plays no active role. It is sufficient to provide a null parameter of type org.omg.CORBA.StringHolder.

IDL syntax

```
  string  getAsDateFormattedString(inout string aTarget,
in ICBCTimestamp::OutputFormat aFormat);
```

ICBCTimestamp::getAsTimeFormattedString

Returns the time portion of this timestamp object formatted according to the parameter aFormat. aFormat must be one of the enumerated values ISO, USA, EUR, JIS. For details of time formats, see “ICBCTime::formattedString” on page 913.

For the C++ implementation of ICBCTimestamp, the application must supply a non-const pointer to a target storage area long enough to receive the formatted string. The function returns a reference to this area. For the Java implementation, this parameter is required, but plays no active role. It is sufficient to provide a null parameter of type org.omg.CORBA.StringHolder.

IDL syntax

```
  string  getAsTimeFormattedString(inout string aTarget,
in ICBCTimestamp::OutputFormat aFormat);
```

ICBCTimestamp::greaterThan

Returns TRUE if the date and time stored in this ICBCTimestamp object is greater than the other ICBCTimestamp object.

IDL syntax

```
  boolean greaterThan(in ICBCTimestamp aTimestamp);
```

ICBCTimestamp::greaterThanOrEqualTo

Returns TRUE if the date and time stored in this ICBCTimestamp object is greater than or equal to the other ICBCTimestamp object.

IDL syntax

```
  boolean greaterThanOrEqualTo(in ICBCTimestamp aTimestamp);
```

ICBCTimestamp::hour

Returns the hour part of the timestamp.

IDL syntax

```
  long hour();
```

ICBCTimestamp::increment

Adds a ICBCDuration to the value of this ICBCTimestamp object.

Incrementing of a timestamp with a duration (of any type) is carried out on the date and time portions of the timestamp as described for ICBCDate and ICBCTime, except that any overflow of microseconds is carried into the seconds portion of the result timestamp, and any overflow of hours is carried into the days portion.

If the years value of the result of an increment operation is greater than 9999 years, then the years value of the result timestamp is set to 9999, and the number of overflow years can be retrieved using the dateOverflow() function.

The change flag of this object is set to TRUE.

Although an ICBCTimestamp object with an time portion equal to 24.00.00 can be initialized, or returned from an ICBCTimestamp::interval operation, and can be input to

an increment or decrement operation, such a timestamp is never be returned as a result of the increment or decrement operation. For the purposes of the calculation, the time portion of the timestamp is considered to be 00.00.00, with the days part incremented by 1. For example, a timestamp value of 1998-11-2-24.00.00.000000 plus or minus a zero duration will return the timestamp result 1998-11-3-00.00.00.000000.

IDL syntax

```
void increment(in ICBCDuration aDuration);
```

ICBCTimestamp::initializeFromDatastoreFormat

Initializes this timestamp object from a string in the internal format of some datastore. Parameter aDatastore must be set to the type of datastore that originates the string. The only accepted value at present is ICBCTimestamp::ORCL, for Oracle. The microseconds part of the timestamp is set to zero. Returns TRUE if the string is a valid timestamp representation and the object is successfully initialized. Returns FALSE if the string is invalid. If FALSE is returned the state of the object is unchanged.

IDL syntax

```
boolean initializeFromDatastoreFormat(in string aTimestamp,
in ICBCTimestamp::DatastoreType aDatastore);
```

ICBCTimestamp::initializeFromDate

Initializes the date part of this timestamp object from the provided ICBCDate parameter. The time portion and the microseconds portion of the timestamp are set to zero.

IDL syntax

```
void initializeFromDate(inICBCDate aDate);
```

ICBCTimestamp::initializeFromDateTime

Initializes this timestamp object from one ICBCDate and one ICBCTime object. The time portion the timestamp takes its value from the ICBCDate and the time portion from the ICBCTime object. The microseconds portion of the timestamp is set to zero.

IDL syntax

```
boolean initializeFromDateTime(in ICBCDate_ptr aICBCDate,
                               in ICBCTime_ptr aICBCTime);
```

ICBCTimestamp::initializeFromString

Accepts a value for the time in the form of a string. Any existing value in the object is replaced. Returns TRUE if the string is a valid timestamp representation and the object is successfully initialized. Returns FALSE if the parameter is invalid. If FALSE is returned the state of the object is unchanged.

IDL syntax

```
boolean initializeFromString(in string aTimestamp);
```

The string must be in one of two formats:

- A character string of length 14 numeric characters, yyyyxxddhhmmss. Leading and trailing blanks are acceptable provided the string length is less than 30 characters. The string must represent a valid date and time. The microseconds portion of the timestamp is set to zero. Only SBCS characters may be present in the string.
- A punctuated string of the form: yyyy-mm-dd-hh.mm.ss.mmmmmm. Leading and trailing blanks are acceptable provided the string length is less than 30 characters. The string must represent a valid date and time. Leading zeros can be omitted from month, day, and hour. Microseconds can be truncated, or omitted entirely. If omitted, microseconds are set to zero. Only SBCS characters may be present in the string.

ICBCTimestamp::initializeFromTime

Initializes the date portion of this timestamp object to the current date, and the time portion from the provided ICBCTime parameter. The microseconds portion of the timestamp is set to zero.

IDL syntax

```
void initializeFromTime(in ICBCTime aTime);
```

ICBCTimestamp::interval

Calculates the Duration between two ICBCTimestamp objects. The result is a pointer to a new ICBCDuration object of type TIMESTAMP.

If the interval between TIMESTAMP1 and TIMESTAMP2 is required, and TIMESTAMP1 is greater than or equal to TIMESTAMP2, the operation is performed by subtracting TIMESTAMP2 from TIMESTAMP1. If, however, TIMESTAMP1 is less than TIMESTAMP2, TIMESTAMP1 is is subtracted from TIMESTAMP2, and the sign of the result is made negative.

The following procedural description clarifies the steps involved in the operation result = TIMESTAMP1 - TIMESTAMP2.

```
If MICROSECOND(TS2) <= MICROSECOND(TS1)
   then MICROSECOND(RESULT) = MICROSECOND(TS1)- MICROSECOND(TS2).
```

```
If MICROSECOND(TS2) > MICROSECOND(TS1)
      then MICROSECOND(RESULT) = 1000000 +
           MICROSECOND(TS1) - MICROSECOND(TS2).
           and SECOND(TS2) is incremented by 1.

The seconds and minutes part of the timestamps are subtracted
as specified in rules for subtracting ICBCTime.

If HOUR(TS2) <= HOUR(TS1)
   then HOUR(RESULT) = HOUR(TS1) - HOUR(TS2).

IF HOUR(TS2) > HOUR(TS1)
   then HOUR(RESULT) = 24 + HOUR(TS1) - HOUR(TS2)
   and DAY(TS2) is incremented by 1

The date part of the timestamps is substracted as specified in the
rules for subtracting ICBCDate.
```

IDL syntax

```
ICBCDuration_ptr intervalFromTimestamp (in ICBCTimestamp aTimestamp);
ICBCDuration_ptr intervalFromString(in string aTimestamp);
```

In the case with a string parameter, the parameter must be a valid string representation of a timestamp, in the form yyyy-mm-dd-hh.mm.ss.mmmmmm For details refer to the description of “ICBCTimestamp::initializeFromString” on page 929.

If the string parameter to this function cannot be recognized as a valid timestamp, then a pointer value of -1 (for C++) or null (for Java) is returned.

ICBCTimestamp::isLeapYear

Returns TRUE if the year number in the input parameter is a leap year, FALSE otherwise. Year number must be between 1 and 9999.

If the input parameter is not in this range, FALSE is returned.

IDL syntax

```
boolean  isLeapYear(in long aYear);
```

ICBCTimestamp::isObjectChanged

Returns a boolean indicating whether the the value of this object has been changed since it was initialized.

IDL syntax

```
boolean isObjectChanged();
```

ICBCTimestamp::julianDay

Returns the Julian day number corresponding to noon GMT on the date of this timestamp object. (Julian Day count started at noon, GMT, on January 1 4712 BC.)

IDL syntax

```
long julianDay();
```

ICBCTimestamp::julianDayFromString

Returns the Julian day number corresponding to noon GMT on the timestamp encoded in the input string parameter.

(Julian Day count started at noon, GMT, on January 1 4712 BC) The input string parameter must conform to a format recognizable as a timestamp. Refer to ICBCTimestamp::initializeFromString() for a description of supported formats.

If the input parameter is not recognizable, a value of -1 is returned.

IDL syntax

```
long julianDayFromString(in string aString);
```

ICBCTimestamp::lessThan

Returns TRUE if the date and time stored in this ICBCTimestamp object is less than the other ICBCTimestamp object.

IDL syntax

```
boolean lessThan(in ICBCTimestamp aTimestamp);
```

ICBCTimestamp::lessThanOrEqualTo

Returns TRUE if the date and time stored in this ICBCTimestamp object is less than or equal to the other ICBCTimestamp object.

IDL syntax

```
boolean lessThanOrEqualTo(in ICBCTimestamp aTimestamp);
```

ICBCTimestamp::microsecond

Returns the microsecond part of the timestamp.

IDL syntax

```
long microsecond();
```

ICBCTimestamp::minute

Returns the minute part of the timestamp.

IDL syntax

```
long minute();
```

ICBCTimestamp::modifiedJulianDay

Returns the Modified Julian day number (MJD) corresponding to time 00:00:00 GMT on the date of this timestamp object. MJD is defined as Julian Day - 2,400,000.5 days. It thus starts at time 00:00:00 (i.e., midnight), in line with normal civil practice, and unlike Julian Day which extends from noon to noon.

IDL syntax

```
long modifiedJulianDay();
```

ICBCTimestamp::modifiedJulianDayFromString

Returns the Modified Julian day number (MJD) corresponding to time 00:00:00 GMT on the date of the timestamp encoded in the input string parameter. MJD is defined as Julian Day - 2,400,000.5 days. It thus starts at time 00:00:00 (i.e., midnight), in line with normal civil practice, and unlike Julian Day which extends from noon to noon. The input string parameter must conform to a format recognizable as a timestamp. Refer to ICBCTimestamp::initializeFromString() for a description of supported formats.

If the input parameter is not recognizable, a value of -1 is returned.

IDL syntax

```
long modifiedJulianDayFromString(in string aString);
```

ICBCTimestamp::month

Returns the month part of the timestamp.

IDL syntax

```
long month();
```

ICBCTimestamp::monthFromString

Returns the month part of the timestamp encoded in the input string parameter. The input string parameter must conform to a format recognizable as a timestamp. Refer to ICBCTimestamp::initializeFromString() for a description of supported formats.

If the input parameter is not recognizable, a value of -1 is returned.

IDL syntax

```
long monthFromString(in string aString);
```

ICBCTimestamp::monthName

Returns a string containing the name of the month of this timestamp object.

IDL syntax

```
string monthName();
```

ICBCTimestamp::monthNameFromNumber

Returns a string containing the name of the month for the month number within a year provided in the input parameter. The input parameter must be a number between 1 and 12.

If the input parameter is not recognizable, an empty string is returned.

IDL syntax

```
string monthNameFromNumber(in long aDayIndex);
```

ICBCTimestamp::monthNameFromString

Returns a string containing the name of the month for the timestamp encoded in the input string parameter. The input string parameter must conform to a format recognizable as a timestamp. Refer to ICBCTimestamp::initializeFromString() for a description of supported formats.

If the input parameter is not recognizable, an empty string is returned.

IDL syntax

```
string monthNameFromString(in string aTimestamp);
```

ICBCTimestamp::notEqualTo

The opposite of the operator== member.

IDL syntax

```
boolean notEqualTo(in ICBCTimestamp aTimestamp);
```

ICBCTimestamp::quarter

Returns a number for the quarter within the year for this timestamp object. timestamps between January 1 and March 31 are in the first quarter, and so on.

IDL syntax

```
long quarter();
```

ICBCTimestamp::quarterFromString

Returns a number for the quarter within the year for the timestamp encoded in the input string parameter. The input string parameter must conform to a format recognizable as a timestamp. Refer to ICBCTimestamp::initializeFromString() for a description of supported formats. Timestamps between January 1 and March 31 are in the first quarter, and so on.

If the input parameter is not recognizable, a value of -1 is returned.

IDL syntax

```
long quarterFromString(in string aTimestamp);
```

ICBCTimestamp::second

Returns the second part of the timestamp.

IDL syntax

ICBCTimestamp::timestampOverflow

Returns the number of overflow or underflow years of a preceding increment or decrement operation. For an increment, returns the years in excess of year 9999. For a decrement, returns the number of years less than year 0001.

IDL syntax

```
long timestampOverflow();
```

ICBCTimestamp::year

Returns the year part of the timestamp.

IDL syntax

```
long year();
```

ICBCTimestamp::yearFromString

Returns the year part of the timestamp encoded in the input string parameter. The input string parameter must conform to a format recognizable as a timestamp. Refer to ICBCTimestamp::initializeFromString() for a description of supported formats.

If the input parameter is not recognizable, a value of -1 is returned.

IDL syntax

```
long yearFromString(in string aString);
```

Appendix. Default Filter Constraint Language

Filters are supported in the Notification Service as objects, which can be associated with Proxy or Admin objects. These filter objects may or may not be colocated with the same server in which the Notification Service event channel resides. Each filter object has associated with it one or more constraints which have meaning in a particular filtering constraint grammar or language. The grammar applicable to this implementation is described in this section.

The IBM Component Broker default filter constraint language (IBM_NTF_CTG) is a subset of the default OMG Notification filter constraint language which is an extension of the OMG Trading Service Grammar as described in the OMG Notification Service specification. The characteristics of this grammar are described below:

- The run-time variable $<Ident> is supported for fixed_header, variable_header, and filterable_data. Note that the <Ident>'s are case sensitive.
- Comparative function: == (equality), != (inequality), >, >=, <, <=; the result of applying a comparative function is a boolean value.
- Mathematical operators: +, —, *, /.
- The following precedence relations hold in the absence of parentheses, in the order of highest to lowest:
  not
  *, /
  +, —
  ==, !=, <, <=, >, >=
  and
  or
- The following types are supported:
  boolean, short, unsigned short, long, unsigned long, float, double, char, string.
- The following operator restrictions are applied:
  ==, !=, <, <=, >, >= can only be applied if the left and right operands are of the same type.
  +, —, *, / can only be applied to numeric operands.
  <, <=, >, >= comparisons imply use of the appropriate collating sequence for characters and string.
  TRUE is greater than FALSE for boolean.
- The following representation of literals is supported:
  boolean - TRUE or FALSE
  integers - sequence of digits
  float - digits with decimal point, the exponential notation is not supported.
  characters - char or string
- Only ″OR″ operation are supported between all the constraints.
- Only ″AND″ operation are supported within a single constraint.
- Nesting (...(...)...) are supported.
- ″NOT″ before the '(' is supported
- String1 ~ String2 (String1 is contained within String2) is not supported.
- Leading plus sign is not supported.

- Backslash (\) is not supported.
- The groping operator ',' is not supported.
- $curtime is not supported.
- ':<Ident>:' is not supported.
- The struct member operator '.' is not supported.

Arithmetic conversions for mixed data types

In general, arithmetic conversions follow the Usual arithmetic conversion rules defined by C and C++. In the context of the Notification Service, however, it is not always possible to determine the data types of all operands at compile-time. Therefore, to simplify data conversion rules, most arithmetic operations are performed using CORBA::Long or CORBA::Double. The result of each operation is cast back to the data type of the most capacious of the operands, with its weak or strong type attribute.

The following rules govern mathematical operations with mixed data types.

- If either operand is a CORBA::LongDouble, the other is converted to CORBA::LongDouble and result is CORBA::LongDouble.
- Otherwise, if either operand is a CORBA::Double, the other is converted to CORBA::Double and the result is CORBA::Double.
- Otherwise, if either operand is a CORBA::Float, the other is converted to CORBA::Float and the result is CORBA::Float.
- Otherwise, if either operand is a CORBA::LongLong, the other is converted to CORBA::LongLong and the result is CORBA::LongLong.
- Otherwise the most strongly-typed of the two operands becomes the result type, and both operands are converted to either CORBA::Long or CORBA::ULong.
- When a shorter unsigned type is combined with a larger signed type, the unsigned property does not propagate to the result type.
- When a numeric constant is specified, it is treated as weakly-typed CORBA::Long or in the case of a floating point constant, a weakly-typed CORBA::Double.
- When a boolean operand is used in an arithmetic operation, is treated as weakly-typed CORBA::Long.

For the purpose of describing the operator restrictions, all operands may be classified as one of the following generic types: boolean, numeric, or string. Numeric operands include boolean and strings of length one (i.e., char). Operator restrictions are as follows:

- Comparison operations are valid only when both operands are either boolean, numeric or string.
- Numeric operations are valid only on numeric types.
- For a divide operation, zero is invalid as a denominator.
- A numeric value may not be substituted when a boolean is required.

When first handed a constraint, the Notification Service can only guarantee that it is syntactically correct. It is only when events are filtered, that it becomes possible to check that operands have valid data types. When invalid operands are encountered or when specified identifiers do not exist, the match operation must immediately return FALSE.

The implication of the above rule is that a Notification Service implementation run-time engine must implement short-circuiting of boolean 'and' and 'or' operations. Specifically, 'FALSE and <expression>' must yield FALSE. Similarly, 'TRUE or <expression>' must yield TRUE. In either case, it is not permissible to evaluate <expression>.

As an example, consider the following four events and the associated constraint:

Event 1: <$a, 'Hawaii'>, <$c, 5.0>

Event 2: <$a, 'H'>, <$c, 5.0>

Event 3: <$a, 5>, <$c, 5.0>

Event 4: <$a, 5>, <$b, 5.0>

Constraint: ($a + 1 > 32) or ($b == 5) or ($c > 3)

For the first event, the first expression becomes ('Hawaii' + 1 > 32). Since it is not possible to add '1' to a string data type, the constraint is invalid and the match operation immediately returns FALSE.

In the second event, the first expression becomes ('H' + 1 > 32). Since 'H' is a valid char data type, this yields TRUE (for the ASCII character set) and the match operation immediately returns TRUE. Note that here, the fact that '$b' is not part of the event is immaterial due to defined short-circuit semantics.

For the third event, the first expression yields FALSE and the second expression can not be resolved (since there is no '$b' member in the event). This is an error, so the match operation immediately returns FALSE. Note that, the constraint author could have dealt with the possibility of a missing '$.b' by rewriting the constraint as:

($a + 1 > 32) or (exists $b and .b == 5) or ($c > 3)

In the forth event, the first expression again yields FALSE, but this time '$b' is defined as a floating point '5.0' Following the arithmetic conversion rules, the constant '5' is also cast to floating point and the second expression yields TRUE. Here, the match operation returns TRUE even though the event has no '$c' member.

Short-hand notation for filtering a structured event

Any member of 'EventHeader.FixedEventHeader' or any property in the name-value pairs 'EventHeader.variable_header' and 'EventHeader.filterable_data' may be represented as run-time variables. For example:

$event_type == 'CommunicationsAlarm' and $priority < 2

The following rules govern translation of a run-time variable. '$variable', into a specific event field. The first matching translation is chosen respectively from: a member of 'EventHeader.FixedEventHeader', properties in 'EventHeader.variable_header', and properties in 'EventHeader.filterable_data'. If no match is found, an exception will be raised.

Examples of Notification Service constraints

This section provides annotated examples of constraints written in the Extended Trader Constraint Language defined by the Notification Service. The following examples intend to show the flexibility of this language.

```
$domain_type == 'dt' and not ($event_name == 'en')

$event_type != 'et' and $priority >= 1 and $priority <= 5

$a / 5.5 * ($b * 3.2 + $c * 1.25) > 50.0

$b * ($a + ($b + $c) /($c - $a)) > 10

$a > 1 and not ($b < 1) and $c > 1
```

Constraint language "BNF"

```
<constraint>        :=  /* empty */
                    |   <bool_and>
<bool_and>          :=  <bool_and> and <bool_compare>
                    |   <bool_compare>
<bool_compare>      :=  <expr> == <expr>
                    |   <expr> != <expr>
                    |   <expr> < <expr>
                    |   <expr> <= <expr>
                    |   <expr> > <expr>
                    |   <expr> >= <expr>
```

```
                    |   <expr>

<expr>              :=  <expr> + <term>

                    |   <expr> - <term>

                    |   <term>

<term>              :=  <term> * <factor_not>

                    |   <term> / <factor_not>

                    |   <factor_not>

<factor_not>        :=  not <factor>

                    |   <factor>

<factor>            :=  <Ident>

                    |   <Number>

                    |   - <Number>

                    |   <String>

                    |   TRUE

                    |   FALSE

<Ident>             :=  $ <String>
<String>            :=  <TextChars>

<TextChars>         :=  <TextChars> <TextChar>

<TextChar>          :=  <Alpha>

                    |   <Digit>

<Alpha>             :=  a, b, ..., z

<Number>            :=  <Digits>

                    |   <Digits>.

                    |   .<Digits>

                    |   <Digits>.<Digits>

<Digit>             :=  0, 1, ..., 9
```

Notices

This information was developed for products and services offered in the U.S.A. IBM may not offer the products, services, or features discussed in this document in other countries. Consult your local IBM representative for information on the products and services currently available in your area. Any reference to an IBM product, program, or service is not intended to state or imply that only that IBM product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any IBM intellectual property right may be used instead. However, it is the user's responsibility to evaluate and verify the operation of any non-IBM product, program, or service.

IBM may have patents or pending patent applications covering subject matter described in this document. The furnishing of this document does not give you any license to these patents. You can send license inquiries, in writing, to:

IBM Director of Licensing
IBM Corporation
North Castle Drive
Armonk, NY 10504-1785
U.S.A.

For license inquiries regarding double-byte (DBCS) information, contact the IBM Intellectual Property Department in your country or send inquiries, in writing, to:

IBM World Trade Asia Corporation Licensing
2-31 Roppongi 3-chome, Minato-ku
Tokyo 106, Japan

The following paragraph does not apply to the United Kingdom or any other country where such provisions are inconsistent with local law:

INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS DOCUMENT "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OR CONDITIONS OF NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Some states do not allow disclaimer of express or implied warranties in certain transactions, therefore, this statement may not apply to you.

This information could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the document. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this publication at any time without notice.

Any references in this information to non-IBM Web sites are provided for convenience only and do not in any manner serve as an endorsement of those Web sites. The materials at those Web sites are not part of the materials for this IBM product and use of those Web sites is at your own risk.

IBM may use or distribute any of the information you supply in any way it believes appropriate without incurring any obligation to you.

Licensees of this program who wish to have information about it for the purpose of enabling: (i) the exchange of information between independently created programs and other programs (including this one) and (ii) the mutual use of the information which has been exchanged, should contact:

For Component Broker:
IBM Corporation
Department LZKS
11400 Burnet Road
Austin, TX 78758
U.S.A.

For TXSeries:
IBM Corporation
ATTN: Software Licensing
11 Stanwix Street
Pittsburgh, PA 15222
U.S.A.

Such information may be available, subject to appropriate terms and conditions, including in some cases, payment of a fee.

The licensed program described in this document and all licensed material available for it are provided by IBM under terms of the IBM International Program License Agreement or any equivalent agreement between us.

Any performance data contained herein was determined in a controlled environment. Therefore, the results obtained in other operating environments may vary significantly. Some measurements may have been made on development-level systems and there is no guarantee that these measurements will be the same on generally available systems. Furthermore, some measurements may have been estimated through extrapolation. Actual results may vary. Users of this document should verify the applicable data for their specific environment.

Information concerning non-IBM products was obtained from the suppliers of those products, their published announcements or other publicly available sources. IBM has not tested those products and cannot confirm the accuracy of performance, compatibility or any other claims related to non-IBM products. Questions on the capabilities of non-IBM products should be addressed to the suppliers of those products.

All statements regarding IBM's future direction or intent are subject to change or withdrawal without notice, and represent goals and objectives only.

This information contains examples of data and reports used in daily business operations. To illustrate them as completely as possible, the examples may include the

names of individuals, companies, brands, and products. All of these names are fictitious and any similarity to the names and addresses used by an actual business enterprise is entirely coincidental.

If you are viewing this information softcopy, the photographs and color illustrations may not appear.

Trademarks and service marks

The following terms are trademarks or registered trademarks of the IBM Corporation in the United States, other countries, or both:

AFS
AIX
AS/400
CICS
CICS OS/2
CICS/400
CICS/6000
CICS/ESA
CICS/MVS
CICS/VSE
CICSPlex
DB2
DCE Encina Lightweight Client
DFS
Encina
IBM
IMS
MQSeries
MVS/ESA
OS/2
OS/390
OS/400
PowerPC
RISC System/6000
RS/6000
S/390
Transarc
TXSeries
VSE/ESA
VTAM
VisualAge
WebSphere

Microsoft, Windows, Windows NT, and the Windows 95 logo are trademarks of Microsoft Corporation in the United States and/or other countries.

Oracle and Oracle8 are registered trademarks of the Oracle Corporation in the United States and/or other countries.

UNIX is a registered trademark of The Open Group in the United States and/or other countries licensed exclusively through X/Open Company Limited.

OSF and Open Software Foundation are registered trademarks of the Open Software Foundation, Inc.

* HP-UX is a Hewlett-Packard branded product. HP, Hewlett-Packard, and HP-UX are registered trademarks of Hewlett-Packard Company.

Orbix is a registered trademark and OrbixWeb is a trademark of IONA Technologies Ltd.

Sun, SunLink, Solaris, SunOS, Java, all Java-based trademarks and logos, NFS, and Sun Microsystems are trademarks or registered trademarks of Sun Microsystems, Inc. in the United States and/or other countries.

Some of this documentation is based on material from Object Management Group bearing the following copyright notices:

Copyright 1995, 1996 AT&T/NCR
Copyright 1995, 1996 BNR Europe Ltd.
Copyright 1991, 1992, 1995, 1996 by Digital Equipment Corporation
Copyright 1996 Gradient Technologies, Inc.
Copyright 1995, 1996 Groupe Bull
Copyright 1995, 1996 Expersoft Corporation
Copyright 1996 FUJITSU LIMITED
Copyright 1996 Genesis Development Corporation
Copyright 1989, 1990, 1991, 1992, 1995, 1996 by Hewlett-Packard Company
Copyright 1991, 1992, 1995, 1996 by HyperDesk Corporation
Copyright 1995, 1996 IBM Corporation
Copyright 1995, 1996 ICL, plc
Copyright 1995, 1996 Ing. C. Olivetti &C.Sp
Copyright 1997 International Computers Limited
Copyright 1995, 1996 IONA Technologies, Ltd.
Copyright 1995, 1996 Itasca Systems, Inc.
Copyright 1991, 1992, 1995, 1996 by NCR Corporation
Copyright 1997 Netscape Communications Corporation
Copyright 1997 Northern Telecom Limited
Copyright 1995, 1996 Novell USG
Copyright 1995, 1996 02 Technolgies
Copyright 1991, 1992, 1995, 1996 by Object Design, Inc.
Copyright 1991, 1992, 1995, 1996 Object Management Group, Inc.
Copyright 1995, 1996 Objectivity, Inc.
Copyright 1995, 1996 Oracle Corporation
Copyright 1995, 1996 Persistence Software
Copyright 1995, 1996 Servio, Corp.
Copyright 1996 Siemens Nixdorf Informationssysteme AG
Copyright 1991, 1992, 1995, 1996 by Sun Microsystems, Inc.
Copyright 1995, 1996 SunSoft, Inc.
Copyright 1996 Sybase, Inc.
Copyright 1996 Taligent, Inc.
Copyright 1995, 1996 Tandem Computers, Inc.
Copyright 1995, 1996 Teknekron Software Systems, Inc.
Copyright 1995, 1996 Tivoli Systems, Inc.
Copyright 1995, 1996 Transarc Corporation
Copyright 1995, 1996 Versant Object Technology Corporation
Copyright 1997 Visigenic Software, Inc.
Copyright 1996 Visual Edge Software, Ltd.

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.

WHILE THE INFORMATION IN THIS PUBLICATION IS BELIEVED TO BE ACCURATE, THE OBJECT MANAGEMENT GROUP, AND THE COMPANIES LISTED ABOVE

MAKE NO WARRANTY OF ANY KIND WITH REGARDS TO THIS MATERIAL INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. The Object Management Group and the companies listed above shall not be liable for errors contained herein or for incidental or consequential damages in connection with the furnishing, performance, or use of this material.

This software contains RSA encryption code.

Other company, product, and service names may be trademarks or service marks of others.

Index of Methods, Operations, and Attributes

Special Characters

A

B

C

D

E

F

G

H

I

J

K

L

M

N

S

T

U

V

W

Y

Printed in the United States of America
on recycled paper containing 10%
recovered post-consumer fiber.

SC09-4446-00

Spine information:

WebSphere Programming Reference Version 3.0 SC09-4446-00