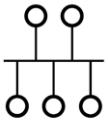


Orbix®

CORBA Programmer's Reference C++

Version 6.0, February 2003

Making Software Work Together™

Updated: 26-Nov-2003

M 3 0 3 6

Contents

List of Tables

Preface

Orbix is a software environment for building and integrating distributed object-oriented applications. Orbix is a full implementation of the Common Object Request Broker Architecture (CORBA) from the Object Management Group (OMG). Orbix fully supports CORBA version 2.3.

This document is based on the CORBA 2.3 standard with some additional features and Orbix-specific enhancements. If you need help with this or any other IONA products, contact IONA at `support@iona.com`. Comments on IONA documentation can be sent to `docs-support@iona.com`.

For the latest online versions of Orbix documentation, see the IONA website:

`http://www.iona.com/support/docs`

Audience

The reader is expected to understand the fundamentals of writing a distributed application with Orbix. Familiarity with C++ is required.

Organization of this Reference

This reference presents core-product modules in alphabetical order, disregarding IT_ prefixes in order to keep together related OMG-compliant and Orbix-proprietary modules. For example, modules `CORBA` and `IT_CORBA` are listed in sequence.

Modules that are specific to a service are also grouped together under the service's name—for example, modules `CosPersistentState`, `IT_PSS`, and `IT_PSS_DB` are listed under Persistent State Service.

Related Documentation

This document is part of a set that comes with the Orbix product. Other books in this set include:

- *Application Server Platform Administrator's Guide*
- *CORBA Programmer's Guide*
- *CORBA Code Generation Toolkit Guide*

Document Conventions

This guide uses the following typographical conventions:

`Constant width` Constant width (courier font) in normal text represents portions of code and literal names of items such as classes, methods, variables, and data structures. For example, text might refer to the `CORBA::Object` class.

Constant width paragraphs represent code examples or information a system displays on the screen. For example:

```
#include <stdio.h>
```

Italic Italic words in normal text represent *emphasis* and *new terms*.

Italic words or characters in code and commands represent variable values you must supply, such as arguments to commands or path names for your particular system. For example:

```
% cd /users/your_name
```

Note: some command examples may use angle brackets to represent variable values you must supply. This is an older convention that is replaced with *italic* words or characters.

This guide may use the following keying conventions:

No prompt	When a command's format is the same for multiple platforms, a prompt is not used.
%	A percent sign represents the UNIX command shell prompt for a command that does not require root privileges.
#	A number sign represents the UNIX command shell prompt for a command that requires root privileges.
>	The notation > represents the DOS, WindowsNT, Windows95, or Windows98 command prompt.
. . . . . .	Horizontal or vertical ellipses in format and syntax descriptions indicate that material has been eliminated to simplify a discussion.
[ ]	Brackets enclose optional items in format and syntax descriptions.
{ }	Braces enclose a list from which you must choose an item in format and syntax descriptions.
|	A vertical bar separates items in a list of choices enclosed in { } (braces) in format and syntax descriptions.

Introduction

This describes all of the standard programmer's API for CORBA and Orbix. This introduction contains the following topics:

The rest of the *CORBA Programmer's Reference* contains the following modules and appendix:

Interface Repository Quick Reference

The interface repository (IFR) is the component of Orbix that provides persistent storage of IDL definitions. Programs use the following API to query the IFR at runtime to obtain information about IDL definitions:

Table 1: *Interface Repository API*

CORBA Types	CORBA Sequences
ContextIdentifier	AttrDescriptionSeq
Identifier	ContainedSeq
RepositoryId	ContextIdSeq
ScopedName	ExceptionDefSeq
VersionSpec	ExcDescriptionSeq
ValueModifier	EnumMemberSeq
Visibility	InitializerSeq
ValueModifier	InterfaceDefSeq
Visibility	OpDescriptionSeq
	ParDescriptionSeq
	RepositoryIdSeq
	StructMemberSeq
	UnionMemberSeq
	ValueDefSeq
	ValueMemberSeq

CORBA Structures	CORBA Enumerated Types
AttributeDescription	AttributeMode
ConstantDescription	DefinitionKind
ExceptionDescription	OperationMode
Initializer	ParameterMode
InterfaceDescription	PrimitiveKind
ModuleDescription	TCKind
OperationDescription	
ParameterDescription	
StructMember	
TypeDescription	
UnionMember	
ValueDescription	
ValueMember	

Table 1: *Interface Repository API*

CORBA Classes and Interfaces	Typecode Methods in CORBA::ORB
AliasDef	create_abstract_interface_tc()
ArrayDef	create_alias_tc()
AttributeDef	create_array_tc()
ConstantDef	create_enum_tc()
Contained	create_exception_tc()
Container	create_fixed_tc()
EnumDef	create_interface_tc()
ExceptionDef	create_native_tc()
Environment	create_recursive_tc()
FixedDef	create_sequence_tc()
IDLType	create_string_tc()
InterfaceDef	create_struct_tc()
IRObject	create_union_tc()
ModuleDef	create_value_box_tc()
NativeDef	create_value_tc()
OperationDef	create_wstring_tc()
PrimitiveDef	
Repository	
SequenceDef	
StringDef	
StructDef	
TypeCode	
TypedefDef	
UnionDef	
ValueBoxDef	
ValueDef	
ValueMemberDef	
WstringDef	

DII and DSI Quick Reference

The client-side dynamic invocation interface (DII) provides for the dynamic creation and invocation of requests for objects. The server-side counterpart to the DII is the dynamic Skeleton interface (DSI) which dynamically handles object invocations. This dynamic system uses the following data structures, interfaces, and classes:

Table 2: *DII and DSI API*

DII Classes	DSI Classes

`CORBA::Context`
`CORBA::ContextList`
`CORBA::ExceptionList`
`CORBA::Request`
`CORBA::TypeCode`
	`CORBA::ServerRequest`
	`PortableServer::DynamicImplementation`

Key Data Types	**DII-Related Methods**

`CORBA::Any`
`CORBA::Flags`
`CORBA::NamedValue`
`CORBA::NVList`
	`CORBA::Object::_create_request()`
	`CORBA::ORB::create_list()`
	`CORBA::ORB::create_operation_list()`
	`CORBA::ORB::get_default_context()`

Value Type Quick Reference

A value type is the mechanism by which objects can be passed by value in CORBA operations. Value types use the following data structures, methods, and value types from the CORBA module:

Types

`StringValue`
`ValueFactory`
`WStringValue`

Value Types and Classes

`CustomMarshal`

DataInputStream
DataOutputStream
ValueBase
ValueFactory
ValueFactoryBase
ValueDef

Global Functions

add_ref()
remove_ref()

Sequences

AnySeq
BooleanSeq
CharSeq
DoubleSeq
FloatSeq
OctetSeq
ShortSeq
UShortSeq
ULongLongSeq
ULongSeq
WCharSeq

About Standard Functions for all Interfaces

Every IDL interface also has generated helper functions:

_duplicate()

```
inline static CLASS_ptr _duplicate(
    CLASS_ptr p
);
```

This function returns a duplicate object reference and increments the reference count of the object. Use this function to create a copy of an object reference.

Parameters

p	The current object reference to duplicate.

Notes This is a standard function generated for all interfaces.

_narrow()

```
static CLASS_ptr _narrow(
    CORBA::Object_ptr obj
);
```

This function returns a new object reference given an existing reference. Use this function to narrow an object reference.

Parameters

obj	A reference to an object. The function returns a nil object reference if this parameter is a nil object reference.

Notes This is a standard function generated for all interfaces.

When you have IDL interfaces that inherit from each other, you often need to convert a reference of one type to a related type. This is analogous to casting between pointers to classes which inherit from each other classes in C++. For example suppose you have the following interfaces:

```
// IDL
interface Base { ... };
interface Derived : Base { ... };
```

Now suppose you have a reference of type `Base` but it refers to an object which is actually of type `Derived`. Converting the `Base` reference to a `Derived` reference is called *narrowing* because you are converting from a more general type to a more specific, or narrow, type. Conversely converting a `Derived` reference to a `Base` reference is called *widening*. Note that narrowed or widened references still refer to the same object, they are simply different *views* of that object.

Always check the results of `_narrow()` with `CORBA::is_nil()`. The `_narrow()` function checks whether the reference actually refers to an object of the type you are narrowing to. If not, `_narrow()` returns a nil reference.

The _narrow() function does an implicit duplicate, so you are responsible for releasing both the original reference and the new reference returned. The easiest way to do this is by assigning both to _var variables.

The _narrow() function can actually both narrow and widen references. It takes a CORBA::Object_ptr parameter and tests whether the requested interface is compatible with the actual most-derived interface implemented by the object, regardless of the inheritance relationships involved.

Exceptions A standard system exception can be raised in some unusual cases where a remote call occurs to the object being narrowed. However, normally _narrow() is a local function call and it can figure out the conversion based on information in the IDL compiler generated stub code.

See Also _unchecked_narrow()

_nil()

```
inline static CLASS_ptr _nil();
```

Returns a nil object reference to the object.

Notes This is a standard function generated for all interfaces.

_unchecked_narrow()

```
static CLASS_ptr _unchecked_narrow(
    CORBA::Object_ptr obj
);
```

Returns a new object reference to the object given an existing reference. However, unlike _narrow(), this function does not verify that the actual type of the parameter at runtime can be widened to the requested interface's type.

Parameters

obj	A reference to an object.

Notes This is a standard function generated for all interfaces.

See Also _narrow()

About Reference Types _ptr, _var, and _out

Every IDL interface has generated helper pointer types that you use as object references. The object reference pointer type names generated are based on the interface name and include:

*InterfaceName*_ptr	Use the *InterfaceName*_ptr type to reference *InterfaceName* objects in a manner similar to a C++ pointer.
*InterfaceName*_var	Use the *InterfaceName*_var type to reference objects so that the object's memory management is automatic.
*InterfaceName*_out	The *InterfaceName*_out type is used only in method signatures when referring to *InterfaceName* objects as out parameters. This type gives Orbix the ability to implicitly release a previous value of an *InterfaceName*_var when it is passed as an out parameter.

Reference Example

Assume the following interface for this discussion:

```
// IDL
interface InterfaceName {
    InterfaceName op(
        in   InterfaceName arg1,
        out InterfaceName arg2
    );
};
```

The following example shows the C++ pointer helper classes that the IDL compiler generates for the object reference pointer types. (No inline implementation details are shown):

```
class InterfaceName; // forward reference


typedef InterfaceName *InterfaceName_ptr;


class InterfaceName_var : public _var {
```

```
public:
    InterfaceName_var() : ptr_(InterfaceName::_nil()) { }
    InterfaceName_var(InterfaceName_ptr p) : ptr_(p) { }
    InterfaceName_var(const InterfaceName_var &a) :
        ptr_(InterfaceName::_duplicate(InterfaceName_ptr(a){ }
    ~InterfaceName_var() { }
    InterfaceName_var &operator=(InterfaceName_ptr p) { }
    InterfaceName_var &operator=(const InterfaceName_var& a) { }
    InterfaceName_ptr in() const { }
    InterfaceName_ptr& inout() { }
    InterfaceName_ptr& out() { }
    InterfaceName_ptr _retn() { }
    operator const InterfaceName_ptr&() const { }
    operator InterfaceName_ptr&() { }
    InterfaceName_ptr operator->() const { }
protected:
    InterfaceName_ptr ptr_;
    void free() { }
    void reset(InterfaceName_ptr p) { }
private:
    ...
};


class InterfaceName_out {
public:
    InterfaceName_out(InterfaceName_ptr& p) : ptr_(p) { }
    InterfaceName_out(InterfaceName_var& p) : ptr_(p.ptr_) { }
    InterfaceName_out(InterfaceName_out& a) : ptr_(a.ptr_) { }
    InterfaceName_out& operator=(InterfaceName_out& a) { }
    InterfaceName_out& operator=(const InterfaceName_var& a) { }
    InterfaceName_out& operator=(InterfaceName_ptr p) { }
    operator InterfaceName_ptr&() { }
    InterfaceName_ptr& ptr() { }
    InterfaceName_ptr operator->() { }
private:
    ...
};
```

Widening and Narrowing References

As with C++ class pointers you can widen _ptr references by assignment. For example:

```
// C++
// This is legal, but be careful of memory management with _ptr!
Derived_ptr derived_ref = ...;   // Get a Derived reference.
Base_ptr base_ref = derived_ref; // Widening assignment.
```

In general you should use _var references to avoid memory leaks. You cannot widen by direct assignment of _var types, instead you must use _duplicate() explicitly. This is because of C++ problems in implementing all the necessary conversion operators.

```
Derived_var derived_ref = ...;
Base_var base_ref = Base::_duplicate(derived_ref);
```

As in C++ you cannot narrow references by simple assignment or duplication. Note that it is *not* legal to use C++ casting to narrow CORBA object references (even if your compiler supports dynamic casts.) Instead you use the static _narrow() function on a class corresponding to the interface you want to narrow to. For example:

```
// C++
Base_var base_ref = ...; // Get a Base reference somehow.
Derived_var derived_ref = Derived::_narrow(base_ref);
if (CORBA::is_nil(derived_ref))
{
    // base_ref does not refer to an object of type Derived.
}
else
{
    // We can use derived_ref to call Derived operations.
}
```

About Sequences

An IDL sequence maps to a class of the same name. For example, an IDL sequence named *Type*Seq which is made up of a sequence of *Type* IDL data types, has the class *Type*Seq implemented.

```
// IDL
```

```
typedef sequence<Type> TypeSeq;
```

The implemented *Type*Seq class contains the following functions:

```
// C++
class TypeSeq {
public:
    // default constructor
    TypeSeq();
    // initial maximum length constructor
    TypeSeq(ULong max);
    // data constructor
    TypeSeq(
        ULong max,
        ULong length,
        Type *data,
        Boolean release = FALSE
    );
    // copy constructor
    TypeSeq(const TypeSeq&);

     // destructor
    ~TypeSeq();

    // assignment operator
    TypeSeq &operator=(const TypeSeq&);

    ULong maximum() const;
    void length(ULong);
    ULong length() const;

    // subscript operators
    Type &operator[](ULong index);
    const Type &operator[](ULong index) const;

    Boolean release() const;
    void replace(
        ULong max,
        ULong length,
        Type *data,
        Boolean release = FALSE
    );

    // buffer reference
```

```
    Type* get_buffer(Boolean orphan = FALSE);
    // buffer access
    const Type* get_buffer() const;
};
```

Each function is described as follows.

*Type*Seq() A sequence has four possible constructors:

- The default constructor sets the sequence length equal to 0.
- The constructor with the single `max` parameter allows you to set the initial value for the maximum length of the sequence. This allows you to control how much buffer space is initially allocated by the sequence. This constructor also sets the length to 0 and the sequence release flag to TRUE.
- The data constructor (the one with the `*data` parameter) lets you set the length and contents of the sequence. It also allows you to set the initial value for the maximum length. For this constructor, ownership of the buffer is determined by the release parameter.
- The copy constructor creates a new sequence with the same maximum and length as the given sequence parameter, copies each of its current elements (items zero through length-1), and sets the sequence release flag to TRUE.

~*Type*Seq() For the destructor, if the sequence release flag equals TRUE the destructor destroys each of the current elements (items zero through length-1), and destroys the underlying sequence buffer. If the sequence release flag equals FALSE, the calling code is responsible for managing the buffer's storage.

&operator=() The assignment operator (=) deep-copies the sequence, releasing old storage if necessary.

`maximum()`	The `maximum()` function returns the total number of sequence elements that can be stored in the current sequence buffer. This allows you to know how many items you can insert into an unbounded sequence without causing a reallocation.
`length()`	Use the `length()` functions to access and modify the length of the sequence. Increasing the length of a sequence adds new elements at the end. The newly-added elements behave as if they are default-constructed when the sequence length is increased.
`&operator[]()`	The overloaded subscript operators (`[]`) return the item at the given index.
`release()`	The `release()` function returns the state of the sequence release flag. FALSE means the caller owns the storage for the buffer and its elements, while TRUE means that the sequence manages its own storage for the buffer and its elements.
`replace()`	The `replace()` function lets you replace the buffer underlying a sequence. The parameters to `replace()` are identical in type, order, and purpose to those for the data constructor for the sequence.

`get_buffer()` The overloaded `get_buffer()` functions allow direct access to the buffer underlying a sequence. These can be very useful when sending large blocks of data as sequences and the per-element access provided by the overloaded subscript operators is not sufficient.

- The non-constant `get_buffer()` reference function allows read-write access to the underlying buffer. If its orphan argument is FALSE (the default), the sequence returns a pointer to its buffer, allocating one if it has not yet done so. The size of the buffer can be determined using the sequence's `maximum()` function. The number of elements in the buffer can be determined from the sequence's `length()` function. The sequence maintains ownership of the underlying buffer. Elements in the returned buffer may be directly replaced by your code. However, because the sequence maintains the length and size of the buffer, code that calls `get_buffer()` cannot lengthen or shorten the sequence by directly adding elements to or removing elements from the buffer.
- The const `get_buffer()` access function allows read-only access to the sequence buffer. The sequence returns its buffer, allocating one if one has not yet been allocated. No direct modification of the returned buffer is allowed by the calling code.

About Value Boxes

A value box is a value type that is a form of simple containment. It is like an additional namespace that contains only one name. A value box has no inheritance or operations and it contains a single state member. This allows it to be a concrete rather than abstract class.

The C++ mapping for a value box depends on the underlying type. CORBA contains the two string value boxes `StringValue` and `WStringValue`. The mapping as follows:

```
// IDL
```

valuetype *StringTypeValue stringtype*;

The implemented *StringTypeValue* class contains the following functions:

```
class StringTypeValue : public DefaultValueRefCountBase {
public:
    // constructors
    StringTypeValue();
    StringTypeValue(const StringTypeValue& val);
    StringTypeValue(char* str);
    StringTypeValue(const char* str);
    StringTypeValue(const String_var& var);

    // assignment operators
    StringTypeValue& operator=(char* str);
    StringTypeValue& operator=(const char* str);
    StringTypeValue& operator=(const String_var& var);

    // accessor
    const char* _value() const;

    // modifiers
    void _value(char* str);
    void _value(const char* str);
    void _value(const String_var& var);

    // explicit argument passing conversions for underlying string
    const char* _boxed_in() const;
    char*& _boxed_inout();
    char*& _boxed_out();

    // ...other String_var functions such as overloaded

    // subscript operators, etc....
    static StringTypeValue* _downcast(ValueBase* base);
protected:
    ~StringTypeValue();
    ...
};
```

In order to allow boxed strings to be treated as normal strings where appropriate, a boxed string provides most of the same interface as the `String_var` class.

The function of the value box class for strings are described as follows:

`StringTypeValue()` Public constructors include:

- The default constructor initializes the underlying string to an empty string.
- One constructor takes a `char*` argument which is adopted.
- One constructor takes a `const char*` which is copied.
- One constructor takes a `const String_var&` from which the underlying string value is copied. If the `String_var` holds no string, the boxed string value is initialized to the empty string.

`operator=()` There are three public assignment operators. Each returns a reference to the object being assigned to:

- one that takes a parameter of type `char*` which is adopted.
- One that takes a parameter of type `const char*` which is copied.
- One that takes a parameter of type `const String_var&` from which the underlying string value is copied. If the `String_var` holds no string, the boxed string value is set equal to the empty string.

`_value()`	Public accessor and modifier functions for the `StringValue`. • The single accessor function takes no arguments and returns a `const char*`. There are three modifier functions, each taking a single argument. • One takes a `char*` argument which is adopted by the value box class. • One modifier function takes a `const char*` argument which is copied. • One takes a `const String_var&` from which the underlying string value is copied.
`_boxed_in()`	Allows the boxed value to be passed as an in parameter. This is the boxed string counterpart to the `String_var::in()` function.
`_boxed_inout()`	Allows the boxed value to be passed as an inout parameter. This is the boxed string counterpart to the `String_var::inout()` function.
`_boxed_out()`	Allows the boxed value to be passed as an out parameter. This is the boxed string counterpart to the `String_var::out()` function.
`operator[]()`	Note that even though the boxed string provides overloaded subscript operators, the fact that values are normally handled by pointer means that they must be dereferenced before the subscript operators can be used.
`_downcast()`	A downcast function.
`~StringValue()`	The destructor is not generally used.

IT_Config Module

IT_Config Overview

Every ORB is associated with a configuration domain that provides it with configuration information. The configuration mechanism enables Orbix to get its configuration information from virtually any source including files or configuration repositories. The `IT_Config` module contains the API to both get configuration settings and receive notifications when a particular configuration value changes. The module contains the following interfaces:

- `Configuration`
- `Listener`

The `IT_Config` module does not give you a mechanism for changing configurations. Administrators typically setup and manage a configuration domain using various tools described in the *Application Server Platform Administrator's Guide*. However, applications can locally override a configuration, without changing the configuration domain, by passing in configuration variables in the command line. These configuration variables are processed by `CORBA::ORB_init()` where the ORB processes them first before querying the configuration domain.

A single *configuration domain* can hold configuration information for multiple ORBs – each ORB uses its ORB name as a "key" to locate its particular configuration within the domain. Often, an administrator will want to use a default configuration domain for a group of applications, overriding only certain configuration variables for individual applications or ORBs. This might be useful within a hierarchical organization, or where different development groups or applications need slightly different configurations.

A configuration domain can be organized into a hierarchy of nested *configuration scopes* to enable a high degree of flexibility. Each scope within a domain must be uniquely named relative to its containing scope. Scope names consist of any combinations of alphanumeric characters and underscores. Scopes are usually identified by their fully qualified name, which contains the scope name and the names of all parent scopes, separated by a dot (.).

Within each configuration scope, variables are organized into configuration contexts. A *configuration context* is simply a collection of related configuration variables. A context may also contain sub-contexts. You can consider the configuration scope as the root context. Contained in the root context are a number of sub-contexts. For example, there is a plug-ins context and an initial-references context. The initial-references context contains a list of initial-references for the services available to the system. The plug-ins context contains a sub-context for each plug-in, in which it holds its configuration information. This context will have the same name as the plug-in, and will hold information such as the name of the plug-in library and any dependencies the plug-in has, as well as other plug-in-specific settings.

You as a programmer need not worry about this configuration hierarchy set up by your administrator. You simply request configuration values via the `Configuration` interface. See the *Application Server Platform Administrator's Guide* for more on configuration.

IT_Config::ConfigList Sequence

```
// IDL
typedef sequence<string> ConfigList;
```

A list of configuration settings as strings.

Enhancement This is an Orbix enhancement.

See Also `IT_Config::Configuration::get_list()`
`IT_Config::Listener::list_changed()`

IT_Config::TargetScope Enumeration

```
// IDL
enum TargetScope {
    OBJECT_SCOPE,
    ONELEVEL_SCOPE,
    SUBTREE_SCOPE
};
```

A target scope refers to the extent of a configuration hierarchy that a Listener object monitors.

OBJECT_SCOPE	The Listener is only interested in changes to the specific target variable. For example, a Listener with a target variable of initial_references:Naming:reference and a target scope of OBJECT_SCOPE is informed if that variable changes.
ONELEVEL_SCOPE	The Listener is interested in changes to variables contained in the target, a configuration context, but not the target itself. For example, if the target is plugins:iiop, the Listener is informed of any changes to variable in the plugins:iiop configuration context.
SUBTREE_SCOPE	The Listener is interested in changes to the target and any variables or namespaces in the subtree of the target. For example, if the target is initial_references, the Listener is informed of any changes to anything under the initial_references namespace, including the namespace itself.

Enhancement This is an Orbix enhancement.

See Also IT_Config::Configuration::add_listener()

IT_Config::Configuration Interface

This interface provides access to configuration information. You get a reference to a `Configuration` implementation by calling `ORB::resolve_initial_references()` with the string argument `IT_Configuration`.

In a configuration domain, the ORB name acts as the configuration scope in which to start looking for configuration information. The ORB supplies this information when querying the configuration system for a configuration variable. If the variable cannot be found within that scope or the scope does not exist, the system recursively searches the containing scope. For example, if an ORB with an ORB name of `IONA.ProdDev.TestSuite.TestMgr` requests a variable, the system will first look in the `IONA.ProdDev.TestSuite.TestMgr` scope, then `IONA.ProdDev.TestSuite`, and so on, until it finally looks in the root scope. This allows administrators to place default configuration information at the highest level scope, then override this information in descendant scopes to produce a specific, tailored configuration.

Although there are specific operations such as `get_boolean()` and `get_double()` to retrieve certain types of configuration information, the `Configuration` interface is not strictly typed. This means that when a certain type of variable is requested, an effort is made to convert the retrieved value to the requested type. For example, if you call `get_long()`, and the domain has a string such as "1234", an attempt is made to convert the string to a long. In this case, it can successfully return 1234 as a long. If, however, the value for the requested variable were words such as "A String Value", then it cannot be converted to a long and a `TypeMismatch` exception is thrown.

```
// IDL in module IT_Config
interface Configuration {

    exception TypeMismatch {};

    boolean get_string(
        in string  name,
        out string value
    ) raises (TypeMismatch);
```

```
    boolean get_list(
        in string      name,
        out ConfigList value
    ) raises (TypeMismatch);

    boolean get_boolean(
        in string   name,
        out boolean value
    ) raises (TypeMismatch);

    boolean get_long(
        in string name,
        out long  value
    ) raises (TypeMismatch);

    boolean get_double(
        in string  name,
        out double value
    ) raises (TypeMismatch);

    void add_listener(
        in string target,
        in TargetScope target_scope,
        in Listener l
    );

    void remove_listener(
        in Listener l
    );

    // INTERNAL USE ONLY
    //
    void shutdown();
};
```

Configuration::add_listener()

```
// IDL
void add_listener(
    in string target,
```

```
    in TargetScope target_scope,
    in Listener l
);
```

Adds a Listener object so your application can be notified of certain configuration changes.

Parameters

target	The target configuration scope for the Listener.
target_scope	The scope parameter determines the extent of change that the Listener is told about.
l	The Listener object.

Not all types of configuration domains support change notification.

Enhancement This is an Orbix enhancement.

See Also IT_Config::TargetScope
IT_Config::Configuration::remove_listener()

Configuration::get_boolean()

```
// IDL
boolean get_boolean(
    in string   name,
    out boolean value
) raises (TypeMismatch);
```

Returns true if the boolean value is successfully retrieved and false if the variable could not be found.

Parameters

name	Name of the variable to retrieve.
value	The value of the variable returned.

Enhancement This is an Orbix enhancement.

Exceptions

TypeMismatch	The variable exists but is of the wrong type for this operation.

Configuration::get_double()

```
// IDL
boolean get_double(
    in string  name,
    out double value
) raises (TypeMismatch);
```

Returns true if the double value is successfully retrieved and false if the variable could not be found.

Parameters

name	Name of the variable to retrieve.
value	The value of the variable returned.

Enhancement This is an Orbix enhancement.

Exceptions

TypeMismatch	The variable exists but is of the wrong type for this operation.

Configuration::get_list()

```
// IDL
boolean get_list(
    in string      name,
    out ConfigList value
) raises (TypeMismatch);
```

Returns true if the list of configuration settings is successfully retrieved and false if the list could not be found.

Parameters

name	Name of the configuration list to retrieve.
value	The values returned.

Enhancement This is an Orbix enhancement.

Exceptions

TypeMismatch	The variable exists but is of the wrong type for this operation.

Configuration::get_long()

```
// IDL
boolean get_long(
    in string name,
    out long  value
) raises (TypeMismatch);
```

Returns true if the long value is successfully retrieved and false if the variable could not be found.

Parameters

`name`	Name of the variable to retrieve.
`value`	The value of the variable returned.

Enhancement This is an Orbix enhancement.

Exceptions

`TypeMismatch`	The variable exists but is of the wrong type for this operation.

Configuration::get_string()

```
// IDL
boolean get_string(
    in string  name,
    out string value
) raises (TypeMismatch);
```

Returns true if the string value is successfully retrieved and false if the variable could not be found.

Parameters

`name`	Name of the variable to retrieve.
`value`	The value of the variable returned.

Enhancement This is an Orbix enhancement.

Exceptions

`TypeMismatch`	The variable exists but is of the wrong type for this operation.

Configuration::remove_listener()

```
// IDL
void remove_listener(
    in Listener l
);
```

Removes a Listener object.

Enhancement This is an Orbix enhancement.

See Also IT_Config::Configuration::add_listener()

Configuration::shutdown()

```
// IDL
void shutdown();
```

Note: For internal use only

Configuration::TypeMismatch Exception

```
// IDL
exception TypeMismatch {};
```

The type of the configuration variable named in the operation does not match the type required for the operation.

Enhancement This is an Orbix enhancement.

IT_Config::Listener Interface

You can add a `Listener` object to your application that will be notified of configuration changes that occur. Use `add_listener()` and `remove_listener()` of the `Configuration` interface to manage a `Listener` object.

```
// IDL in module IT_Config
interface Listener {
    void string_changed(
        in string name,
        in string new_value,
        in string old_value
    );

    void list_changed(
        in string     name,
        in ConfigList new_value,
        in ConfigList old_value
    );

    void boolean_changed(
        in string  name,
        in boolean new_value,
        in boolean old_value
    );

    void long_changed(
        in string name,
        in long   new_value,
        in long   old_value
    );

    void double_changed(
        in string name,
        in double new_value,
        in double old_value
    );
};
```

Listener::boolean_changed()

```
// IDL
void boolean_changed(
    in string  name,
    in boolean new_value,
    in boolean old_value
);
```

The application is notified if the boolean value changes.

Parameters

`name`	The name of the variable.
`new_value`	The value of the variable after the change occurred. If a variable is deleted this value will be NULL.
`old_value`	The previous value of the variable before the change occurred. If a variable is added this value will be NULL.

Enhancement This is an Orbix enhancement.

Listener::double_changed()

```
// IDL
void double_changed(
    in string name,
    in double new_value,
    in double old_value
);
```

The application is notified if the double value changes.

Parameters

`name`	The name of the variable.
`new_value`	The value of the variable after the change occurred. If a variable is deleted this value will be NULL.
`old_value`	The previous value of the variable before the change occurred. If a variable is added this value will be NULL.

Enhancement This is an Orbix enhancement.

Listener::list_changed()

```
// IDL
void list_changed(
    in string     name,
    in ConfigList new_value,
    in ConfigList old_value
);
```

The application is notified if the configuration list changes.

Parameters

`name`	The name of the variable.
`new_value`	The value of the variable after the change occurred. If a variable is deleted this value will be NULL.
`old_value`	The previous value of the variable before the change occurred. If a variable is added this value will be NULL.

Enhancement This is an Orbix enhancement.

Listener::long_changed()

```
// IDL
void long_changed(
    in string name,
    in long   new_value,
    in long   old_value
);
```

The application is notified if the long value changes.

Parameters

`name`	The name of the variable.
`new_value`	The value of the variable after the change occurred. If a variable is deleted this value will be NULL.
`old_value`	The previous value of the variable before the change occurred. If a variable is added this value will be NULL.

Enhancement This is an Orbix enhancement.

Listener::string_changed()

```
// IDL
void string_changed(
    in string name,
    in string new_value,
    in string old_value
);
```

The application is notified if the string value changes.

Parameters

`name`	The name of the variable.
`new_value`	The value of the variable after the change occurred. If a variable is deleted this value will be NULL.
`old_value`	The previous value of the variable before the change occurred. If a variable is added this value will be NULL.

Enhancement This is an Orbix enhancement.

CORBA Module

CORBA Overview

The CORBA namespace implements the IDL CORBA module. Additional introductory chapters describe the common methods and definitions found in the scope of the CORBA namespace.

- "Common CORBA Methods"
- "Common CORBA Data Types"

All classes or interfaces defined in the CORBA namespace are described in the following alphabetically ordered chapters:

AliasDef	ExceptionDef	Repository
Any	ExceptionList	Request
ArrayDef	FixedDef	SequenceDef
AttributeDef	IDLType	ServerRequest
ConstantDef	InterfaceDef	StringDef
ConstructionPolicy	IRObject	String_var
Contained	ModuleDef	StructDef
Container	NamedValue	TypeCode
Context	NativeDef	TypedefDef
ContextList	NVList	UnionDef
Current	Object	ValueBase
CustomMarshal	OperationDef	ValueBoxDef
DataInputStream	ORB	ValueDef
DataOutputStream	Policy	ValueFactory
DomainManager	PolicyCurrent	ValueMemberDef
EnumDef	PolicyManager	WstringDef
Environment	PrimitiveDef	WString_var

Some standard system exceptions are also defined in the CORBA module. However, these exceptions are described in Appendix A.

Common CORBA Methods

This section contains details of all common CORBA methods. The following alphabetically ordered list contains a link to the details of each method:

- add_ref()
- _duplicate()
- is_nil()
- _nil()
- ORB_init()
- release()
- remove_ref()
- string_alloc()
- string_dup()
- string_free()

CORBA::add_ref()

```
void add_ref(ValueBase* vb);
```

Increments the reference count of the valuetype instance pointed to by the method's argument. This method does nothing if the argument is null.

Parameters

vb	Pointer to the object reference for the valuetype instance.

This method is provided for consistency with the reference counting method for object references. Unlike the ValueBase::_add_ref() member method, add_ref() can be called with null valuetype pointers.

See Also

CORBA::remove_ref()
CORBA::ValueBase::_remove_ref()
CORBA::ValueBase::_add_ref()

CORBA::_duplicate()

```
static Type_ptr _duplicate(Type_ptr p);
```

Increments the reference count of the object reference, p and returns a copy of the object reference. If p is nil, _duplicate() returns a nil object reference.

Parameters

p	Pointer to the object reference.

See Also CORBA::Object::_duplicate()

CORBA::is_nil()

```
Boolean is_nil(Type_ptr p);
```

Returns a true value if the object reference contains the special value for a nil object reference as defined by the ORB. Otherwise the method returns a false value.

Parameters

p Pointer to the object reference.

Object references cannot be compared using `operator==`; therefore, `is_nil()` is the only compliant way an object reference can be checked to see if it is nil. A compliant program cannot attempt to invoke a method through a nil object reference, since a valid C++ implementation of a nil object reference is a null pointer.

Overloaded versions of this method are generated for each IDL interface, and for each pseudo object type. Object reference types include:

```
Context_ptr
Environment_ptr
NamedValue_ptr
NVList_ptr
Object_ptr
ORB_ptr
POA_ptr
Request_ptr
TypeCode_ptr
```

See Also

CORBA::Object
CORBA::release()
CORBA::_nil()

"About Reference Types _ptr, _var, and _out"

CORBA::_nil()

```
static Type_ptr _nil();
```

Returns a nil object reference for the *Type* interface.

See Also

CORBA::Object
CORBA::is_nil()

CORBA::ORB_init()

```
static ORB_ptr ORB_init(
    int& argc,
    char** argv,
    const char* orb_identifier = ""
);
```

Initializes a client or server connection to an ORB.

Parameters

`argc`	Number of arguments in the argument list, `argv`.
`argv`	Pointer to an argument list of environment-specific data for the call. Valid ORB arguments include: • `-ORBdomain` *`value`* Where to get the ORB actual configuration information. • `-ORBid` *`value`* The ORB identifier. • `-ORBname` *`value`* The ORB name. Each ORB argument is a sequence of configuration strings or options in either of the following forms: `-ORB`*`suffix value`* `-ORB`*`suffixvalue`* The *`suffix`* is the name of the ORB option being set, and *`value`* is the value to which the option is set. Spaces between the *`suffix`* and *`value`* are optional. Any string in the argument list that is not in one of these formats is ignored by the `ORB_init()` method.
`orb_identifier`	The string identifier for the ORB initialized. For example, the string "Orbix" identifies the Orbix ORB from IONA Technologies.

When an application requires a CORBA environment, it uses `ORB_init()` to get the ORB pseudo-object reference. This method first initializes an application in the ORB environment and then it returns the ORB pseudo-object reference to the application for use in future ORB calls. Because applications do not initially have an object on which to invoke ORB calls, `ORB_init()` is a bootstrap call into the CORBA environment. Thus, the `ORB_init()` call is part of the CORBA module but not part of the `CORBA::ORB` class.

Applications can be initialized in one or more ORBs. Special ORB identifiers (indicated by either the `orb_identifier` parameter or the `-ORBid` argument) are intended to uniquely identify each ORB used within the same address

space in a multi-ORB application. The ORB identifiers are allocated by the ORB administrator who is responsible for ensuring that the names are unambiguous. Note the following when assigning ORB identifiers in an `ORB_init()` call:

- If the `orb_identifier` parameter has a value, any `-ORBid` arguments in the `argv` are ignored. However, all other ORB arguments in `argv` might be significant during the ORB initialization process.
- If the `orb_identifier` parameter is null, then the ORB identifier is obtained from the `-ORBid` argument of `argv`.
- If the `orb_identifier` is null and there is no `-ORBid` argument in `argv`, the default ORB is returned in the call.

The `argv` arguments are also examined to determine if there are any other ORB arguments (arguments of the form `-ORB`*`suffix`*). These ORB arguments are processed only by the `ORB_init()` method. In fact, before `ORB_init()` returns, it removes from `argv` all ORB arguments. This unique format for start-up arguments means that your servers do not have to be written to handle ORB arguments.

ORB initialization must occur before POA initialization.

Exceptions

`BAD_PARAM`	A string in `argv` that matches the ORB argument pattern `-ORB`*`suffix`* is not recognized by the ORB.

See Also `CORBA::ORB`

CORBA::release()

```
void release(Type_ptr);
```

Indicates that the caller will no longer access the object reference so that associated resources can be deallocated.

Parameters

`Type``_ptr`	Pointer to the object reference to be released.

If the given object reference is nil, `release()` does nothing.

Overloaded versions of this method are generated for each IDL interface, and for each pseudo object type. Object reference types include:

```
Context_ptr
Environment_ptr
NamedValue_ptr
NVList_ptr
Object_ptr
ORB_ptr
POA_ptr
Request_ptr
TypeCode_ptr
```

See Also

CORBA::Object
CORBA::is_nil()

"About Reference Types _ptr, _var, and _out"

CORBA::remove_ref()

```
void remove_ref(ValueBase* vb);
```

Decrements the reference count of the valuetype instance pointed to by the parameter vb. If the parameter value is a null pointer, this method does nothing.

Parameters

vb	Pointer to the object reference for the valuetype instance.

Unlike the _remove_ref() method, remove_ref() can be called with null valuetype pointers.

See Also

CORBA::add_ref()
CORBA::ValueBase::_remove_ref()
CORBA::ValueBase::_add_ref()

CORBA::string_alloc()

```
char *string_alloc(ULong len);
```

Dynamically allocates a string. The method returns a pointer to the start of the character array. It returns a zero pointer if it cannot perform the allocation. A conforming program should use this method to dynamically allocate a string that is passed between a client and a server.

Parameters

`len`	A string of length len + 1 is allocated.

See Also CORBA::string_free()
CORBA::string_dup()

CORBA::string_dup()

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

Duplicates a string. The method returns a duplicate of the input string or it returns a zero pointer if it is unable to perform the duplication. CORBA::string_alloc() can be used to allocate space for the string.

Parameters

`str`	The string to be duplicated.

See Also CORBA::string_alloc()
CORBA::string_free()

CORBA::string_free()

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

Deallocates a string that was previously allocated using CORBA::string_alloc().

Parameters

`str`	The string to be freed.

See Also CORBA::string_alloc()
CORBA::string_dup()

Common CORBA Data Types

This chapter contains details of all common CORBA data types. Table 3 consists of descriptions of the primitive C++ data types such as `Short`, `Long`, and so on. The following alphabetically ordered list contains a link to the details of each data type:

AnySeq
AttrDescriptionSeq
AttributeDescription
AttributeMode
BooleanSeq
CharSeq
ConstantDescription
ContainedSeq
ContextIdentifier
ContextIdSeq
DefinitionKind
DomainManagersList
DoubleSeq
EnumMemberSeq
ExcDescriptionSeq
ExceptionDefSeq
ExceptionDescription
Flags
FloatSeq
Identifier
Initializer
InitializerSeq
InterfaceDefSeq
InterfaceDescription
InvalidPolicies
ModuleDescription
OctetSeq
OpDescriptionSeq
OperationDescription
OperationMode
ORBid
ParameterDescription
ParameterMode
ParDescriptionSeq
PolicyError
PolicyErrorCode
PolicyList
PolicyType
PolicyTypeSeq
PrimitiveKind
RepositoryId
RepositoryIdSeq
ScopedName
ServiceDetail
ServiceDetailType
ServiceInformation
ServiceOption
ServiceType
SetOverrideType
ShortSeq
StringValue
StructMember
StructMemberSeq
TCKind
TypeDescription
ULongLongSeq
ULongSeq
UnionMember
UnionMemberSeq
UShortSeq
ValueDefSeq
ValueDescription
ValueMember
ValueMemberSeq
ValueModifier
VersionSpec
Visibility
WCharSeq
WStringValue

Primitive C++ types are defined as shown in Table 3:

Table 3: *Primitive C++ Data Types*

Primitive C++ Type	C++ Definition
`Boolean`	`typedef unsigned char Boolean;` (Valid values are 1 for true or 0 for false.)
`Boolean_out`	`typedef Boolean& Boolean_out;`
`Char`	`typedef unsigned char Char;`
`Char_out`	`typedef Char& Char_out;`
`Double`	`typedef double Double;`
`Double_out`	`typedef Double& Double_out;`
`Float`	`typedef float Float;`
`Float_out`	`typedef Float& Float_out;`
`Long`	`typedef long Long;`
`Long_out`	`typedef Long& Long_out;`
`LongDouble`	`typedef long double LongDouble;`
`LongDouble_out`	`typedef LongDouble& LongDouble_out;`
`LongLong`	`typedef ... LongLong;`
`LongLong_out`	`typedef LongLong& LongLong_out;`
`Octet`	`typedef unsigned char Octet;`
`Octet_out`	`typedef Octet& Octet_out;`
`Short`	`typedef short Short;`
`Short_out`	`typedef Short& Short_out;`
`ULong`	`typedef unsigned long ULong;`
`ULong_out`	`typedef ULong& ULong_out;`

Table 3: *Primitive C++ Data Types*

Primitive C++ Type	C++ Definition
ULongLong	typedef ... ULongLong;
ULongLong_out	typedef ULongLong& ULongLong_out;
UShort	typedef unsigned short UShort;
UShort_out	typedef UShort& UShort_out;
WChar	typedef wchar_t WChar;
WChar_out	typedef WChar& WChar_out;

CORBA::AnySeq Sequence

```
//IDL
typedef sequence<any> AnySeq;

//C++
class AnySeq {
    ...
};
```

A sequence of Any data values used for marshalling custom value types.

See Also

CORBA::DataOutputStream
CORBA::DataInputStream

"About Sequences"

CORBA::AttrDescriptionSeq Sequence

```
//IDL
typedef sequence <AttributeDescription> AttrDescriptionSeq;

// C++
class AttrDescriptionSeq {
    ...
```

```
};
```

A sequence of AttributeDescription structures in the interface repository.

See Also CORBA::AttributeDescription

"About Sequences"

CORBA::AttributeDescription Structure

```
// IDL
struct AttributeDescription {
    Identifier name;
    RepositoryId id;
    RepositoryId defined_in;
    VersionSpec version;
    TypeCode type;
    AttributeMode mode;
};
```

```
struct AttributeDescription {
    Identifier name;
    RepositoryId id;
    RepositoryId defined_in;
    VersionSpec version;
    TypeCode type;
    AttributeMode mode;
};
```

The description of an interface attribute in the interface repository.

name	The name of the attribute.
id	The identifier of the attribute.
defined_in	The identifier of the interface in which the attribute is defined.
version	The version of the attribute.
type	The data type of the attribute.
mode	The mode of the attribute.

See Also CORBA::AttributeDef

CORBA::AttributeMode Enumeration

```
// IDL
enum AttributeMode {ATTR_NORMAL, ATTR_READONLY};

// C++
enum AttributeMode {ATTR_NORMAL, ATTR_READONLY};
typedef AttributeMode& AttributeMode_out;
```

The mode of an attribute in the interface repository.

ATTR_NORMAL	Mode is read and write.
ATTR_READONLY	Mode is read-only.

See Also CORBA::AttributeDef

CORBA::BooleanSeq Sequence

```
// IDL
typedef sequence<boolean> BooleanSeq;

// C++
class BooleanSeq {
    ...
};
```

A sequence of Boolean values used in marshalling custom value types.

See Also CORBA::DataOutputStream
CORBA::DataInputStream

"About Sequences"

CORBA::CharSeq Sequence

```
// IDL
typedef sequence<char> CharSeq;

// C++
class CharSeq {
    ...
};
```

A sequence of character (char) values used in marshalling custom value types.

See Also

CORBA::DataOutputStream
CORBA::DataInputStream

"About Sequences"

CORBA::CompletionStatus Enumeration

```
// C++
enum CompletionStatus {
COMPLETED_YES,
COMPLETED_NO,
COMPLETED_MAYBE
};
```

CORBA::ConstantDescription Structure

```
// IDL
struct ConstantDescription {
    Identifier name;
    RepositoryId id;
    RepositoryId defined_in;
    VersionSpec version;
    TypeCode type;
    any value;
};
```

```
// C++
struct ConstantDescription {
    Identifier name;
    RepositoryId id;
    RepositoryId defined_in;
    VersionSpec version;
    TypeCode type;
    any value;
};
```

The description of a constant in the interface repository.

name	The name of the constant.

`id`	The identifier of the constant.
`defined_in`	The identifier of the interface in which the constant is defined.
`version`	The version of the constant.
`type`	The data type of the constant.
`value`	The value of the constant.

See Also CORBA::ConstantDef

CORBA::ContainedSeq Sequence

```
// IDL
typedef sequence <Contained> ContainedSeq;

// C++
class ContainedSeq {
    ...
};
```

A sequence of Contained objects in the interface repository.

See Also CORBA::Contained
"About Sequences"

CORBA::ContextIdentifier Type

```
// IDL
typedef Identifier ContextIdentifier;

// C++
typedef Identifer ContextIdentifier;
```

A context identifier used in an IDL operation in the interface repository.

An IDL operation's context expression specifies which elements of the client's context might affect the performance of a request by the object. The runtime system makes the context values in the client's context available to the object implementation when the request is delivered.

See Also CORBA::OperationDef
CORBA::ContextIdSeq

CORBA::ContextIdSeq Sequence

```
// IDL
typedef sequence <ContextIdentifier> ContextIdSeq;

// C++
class ContextIdSeq {
    ...
};
```

A sequence of `ContextIdentifier` values in the interface repository.

See Also

`CORBA::ContextIdentifier`

"About Sequences"

CORBA::DefinitionKind Enumeration

```
// IDL
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, dk_Fixed,
    dk_Value, dk_ValueBox, dk_ValueMember,
    dk_Native
};

// C++
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, dk_Fixed,
    dk_Value, dk_ValueBox, dk_ValueMember,
    dk_Native
};
typedef DefinitionKind& DefinitionKind_out;
```

Identifies the type of an interface repository object.

Each interface repository object has an attribute (CORBA::IRObject::def_kind) of the type DefinitionKind that records the kind of the IFR object. For example, the def_kind attribute of an InterfaceDef object is dk_interface. The enumeration constants dk_none and dk_all have special meanings when searching for an object in a repository.

See Also

CORBA::IRObject::def_kind
CORBA::Contained
CORBA::Container

CORBA::DomainManagersList Sequence

```
// IDL
typedef sequence <DomainManager> DomainManagersList;
```

```
// C++
class DomainManagersList {
    ...
};
```

A sequence of DomainManager objects.

See Also

CORBA::DomainManager

"About Sequences"

CORBA::DoubleSeq Sequence

```
// IDL
typedef sequence<double> DoubleSeq;
```

```
// C++
class DoubleSeq {
    ...
};
```

A sequence of Double values used in marshalling custom value types.

See Also

CORBA::DataOutputStream
CORBA::DataInputStream

"About Sequences"

CORBA::EnumMemberSeq Sequence

```
// IDL
typedef sequence <Identifier> EnumMemberSeq;

// C++
class EnumMemberSeq {
    ...
};
```

A sequence of `Identifier` strings representing the members of an enumeration type in the interface repository.

See Also

`CORBA::Identifier`
`CORBA::ORB::create_enum_tc()`

"About Sequences"

CORBA::ExcDescriptionSeq Sequence

```
// IDL
typedef sequence <ExceptionDescription> ExcDescriptionSeq;

// C++
class ExcDescriptionSeq {
    ...
};
```

A sequence of `ExceptionDescription` structures in the interface repository. This sequence is used only in the `OperationDescription` structure.

See Also

`CORBA::ExceptionDescription`
`CORBA::OperationDescription`

"About Sequences"

CORBA::ExceptionDefSeq Sequence

```
// IDL
typedef sequence <ExceptionDef> ExceptionDefSeq;

// C++
class ExceptionDefSeq {
    ...
};
```

A sequence of ExceptionDef objects in the interface repository.

See Also CORBA::ExceptionDef

"About Sequences"

CORBA::ExceptionDescription

```
// C++
struct ExceptionDescription {
    Identifier name;
    RepositoryId id;
    RepositoryId defined_in;
    VersionSpec version;
    TypeCode type;
};
```

The description of an exception in the interface repository.

`name`	The name of the exception.
`id`	The identifier of the exception.
`defined_in`	The identifier of the interface in which the exception is defined.
`version`	The version of the exception.
`type`	The data type of the exception.

See Also CORBA::ExcDescriptionSeq

CORBA::ExceptionType Enumeration

```
// IDL
enum exception_type {
NO_EXCEPTION,
USER_EXCEPTION,
SYSTEM_EXCEPTION};
```

CORBA::Flags Type

```
// IDL
typedef unsigned long Flags;
typedef string Identifier;
const Flags ARG_IN = 1;
const Flags ARG_OUT = 2;
const Flags ARG_INOUT = 3;
const Flags CTX_RESTRICT_SCOPE = 15;

//C++
typedef ULong Flags;
```

A flag value is a bitmask `long` used to identify one or more modes.

Most flag values identify the argument passing mode for arguments. The common argument passing flag values include:

`ARG_IN`	Indicates that the associated value is an input-only argument.
`ARG_INOUT`	Flag value indicating that the associated value is an input or output argument.
`ARG_OUT`	Flag value indicating that the associated value is an output-only argument.

Other flag values have specific meanings for request and list methods.

`NVList` methods that add a `NamedValue` to an `NVList` have a `flags` parameter used to specify features of an argument. These additional flag values include:

`IN_COPY_VALUE`	Causes a copy of the argument value to be made and used instead of the argument.
`DEPENDENT_LIST`	If a list structure is added as an item such as in a sublist, this flag indicates that the sublist should be freed when the parent list is freed.

The Object::_create_request() method has a request flags parameter used to specify how storage is to be allocated for output parameters. The additional flag value is:

OUT_LIST_MEMORY	Indicates that any out argument memory is associated with the argument list of the requested IDL operation.

See Also

CORBA::NVList
CORBA::NamedValue
CORBA::Object::_create_request()
CORBA::Context::get_values()

CORBA::FloatSeq Sequence

```
// IDL
typedef sequence<float> FloatSeq;

// C++
class FloatSeq {
    ...
};
```

A sequence of Float values used in marshalling custom value types.

See Also

CORBA::DataOutputStream
CORBA::DataInputStream

"About Sequences"

CORBA::Identifier Type

```
// C++
typedef char* Identifier;
```

A simple name that identifies modules, interfaces, constants, typedefs, exceptions, attributes, and operations in the interface repository. An identifier is not necessarily unique within the entire interface repository; it is unique only within a particular Repository, ModuleDef, InterfaceDef, or OperationDef.

CORBA::Initializer Structure

```
// IDL
struct Initializer {
    StructMemberSeq members;
    Identifier name;
};

// C++
struct Initializer {
    StructMemberSeq members;
    Identifier name;
};
```

An initializer structure for a sequence in the interface repository.

members	The sequence of structure members.
name	The name of the initializer structure.

See Also CORBA::InitializerSeq

CORBA::InitializerSeq Sequence

```
// C++
class InitializerSeq {
    ...
};
```

A sequence of Initializer structures in the interface repository.

See Also CORBA::ValueDef

"About Sequences"

CORBA::InterfaceDefSeq Sequence

```
// C++
class InterfaceDefSeq {
    ...
};
```

A sequence of interface definitions in the interface repository.

See Also

CORBA::InterfaceDef
CORBA::Container::create_interface()
CORBA::Container::create_value()

"About Sequences"

CORBA::InterfaceDescription Structure

```
// IDL
struct InterfaceDescription {
    Identifier name;
    RepositoryId id;
    RepositoryId defined_in;
    VersionSpec version;
    RepositoryIdSeq base_interfaces;
    boolean is_abstract;
};
```

```
// C++
struct InterfaceDescription {
    Identifier name;
    RepositoryId id;
    RepositoryId defined_in;
    VersionSpec version;
    RepositoryIdSeq base_interfaces;
    boolean is_abstract;
};
```

A description of an interface in the interface repository. This structure is returned by the inherited `describe()` method in the `InterfaceDef` interface. The structure members consist of the following:

`name`	The name of the interface.
`id`	The identifier of the interface.
`defined_in`	The identifier of where the interface is defined.
`version`	The version of the interface.
`base_interfaces`	The sequence of base interfaces from which this interface is derived.
`is_abstract`	A true value if the interface is an abstract one, a false value otherwise.

See Also CORBA::InterfaceDef::describe()

CORBA::InvalidPolicies Exception

```
// IDL
exception InvalidPolicies {
    sequence <unsigned short> indices;
};
```

This exception is thrown by operations that are passed a bad policy. The indicated policies, although valid in some circumstances, are not valid in conjunction with other policies requested or already overridden at this scope.

CORBA::ModuleDescription Structure

```
// IDL
struct ModuleDescription {
    Identifier name;
    RepositoryId id;
    RepositoryId defined_in;
    VersionSpec version;
};
struct ModuleDescription {
    Identifier name;
    RepositoryId id;
    RepositoryId defined_in;
    VersionSpec version;
};
```

The description of an IDL module in the interface repository. The structure members consist of the following:

`name`	The name of the module.
`id`	The identifier of the module.
`defined_in`	The identifier of where the module is defined.
`version`	The version of the module.

See Also CORBA::ModuleDef

CORBA::OctetSeq Sequence

```
// C++
class OctetSeq {
    ...
};
```

A sequence of Octet values used in marshalling custom value types.

See Also CORBA::DataOutputStream
CORBA::DataInputStream

"About Sequences"

CORBA::OpDescriptionSeq Sequence

```
// C++
class OpDescriptionSeq {
    ...
};
```

A sequence of OperationDescription structures in the interface repository that describe each IDL operation of an interface or value type.

See Also CORBA::OperationDescription
CORBA::InterfaceDef::FullInterfaceDescription
CORBA::ValueDef::FullValueDescription

"About Sequences"

CORBA::OperationDescription Structure

```
// IDL
struct OperationDescription {
    Identifier name;
    RepositoryId id;
    RepositoryId defined_in;
    VersionSpec version;
    TypeCode result;
    OperationMode mode;
```

```
    ContextIdSeq contexts;
    ParDescriptionSeq parameters;
    ExcDescriptionSeq exceptions;
};

struct OperationDescription {
    Identifier name;
    RepositoryId id;
    RepositoryId defined_in;
    VersionSpec version;
    TypeCode result;
    OperationMode mode;
    ContextIdSeq contexts;
    ParDescriptionSeq parameters;
    ExcDescriptionSeq exceptions;
};
```

This structure describes an IDL operation in the interface repository. The structure members consist of the following:

`name`	The name of the IDL operation.
`id`	The identifier of the IDL operation.
`defined_in`	The identifier of where the IDL operation is defined.
`version`	The version of the IDL operation.
`result`	The `TypeCode` of the result returned by the defined IDL operation.
`mode`	Specifies whether the IDL operation's mode is normal (`OP_NORMAL`) or one-way (`OP_ONEWAY`).
`contexts`	The sequence of context identifiers specified in the context clause of the IDL operation.
`parameters`	The sequence of structures that give details of each parameter of the IDL operation.
`exceptions`	The sequence of structures containing details of exceptions specified in the `raises` clause of the IDL operation.

See Also `CORBA::OpDescriptionSeq`

CORBA::OperationMode Enumeration

```
enum OperationMode {OP_NORMAL, OP_ONEWAY};
typedef OperationMode& OperationMode_out;
```

The mode of an IDL operation in the interface repository. An operation's mode indicates its invocation semantics.

`OP_NORMAL`	The IDL operation's invocation mode is normal.
`OP_ONEWAY`	The IDL operation's invocation mode is oneway which means the operation is invoked only once with no guarantee that the call is delivered.

CORBA::ORBid Type

```
// IDL
typedef string ORBid;
// C++
typedef char* ORBid;
```

The name that identifies an ORB. `ORBid` strings uniquely identify each ORB used within the same address space in a multi-ORB application. `ORBid` strings (except the empty string) are not managed by the OMG but are allocated by ORB administrators who must ensure that the names are unambiguous.

See Also CORBA::ORB_init()

CORBA::ParameterDescription Structure

```
// IDL
struct ParameterDescription {
    Identifier name;
    TypeCode type;
    IDLType type_def;
    ParameterMode mode;
};
```

```
struct ParameterDescription {
    Identifier name;
    TypeCode type;
    IDLType type_def;
```

```
    ParameterMode mode;
};
```

This structure describes an IDL operation's parameter in the interface repository. The structure members consist of the following:

name	The name of the parameter.
type	The TypeCode of the parameter.
type_def	Identifies the definition of the type for the parameter.
mode	Specifies whether the parameter is an in input, output, or input and output parameter.

See Also CORBA::ParDescriptionSeq

CORBA::ParameterMode Enumeration

```
enum ParameterMode {PARAM_IN, PARAM_OUT, PARAM_INOUT};
typedef ParameterMode& ParameterMode_out;
```

The mode of an IDL operation's parameter in the interface repository.

PARAM_IN	The parameter is passed as input only.
PARAM_OUT	The parameter is passed as output only.
PARAM_INOUT	The parameter is passed as both input and output.

CORBA::ParDescriptionSeq Sequence

```
// C++
class ParDescriptionSeq {
    ...
};
```

A sequence of ParameterDescription structures in the interface repository.

See Also CORBA::ParameterDescription
CORBA::OperationDef
CORBA::OperationDescription
CORBA::InterfaceDef
CORBA::ValueDef

"About Sequences"

CORBA::PolicyError Exception

```
// IDL
exception PolicyError {
    PolicyErrorCode reason;
};
```

The `PolicyError` exception is thrown to indicate problems with parameter values passed to `ORB::create_policy()`. Possible reasons are described in the `PolicyErrorCode`.

See Also

`CORBA::ORB::create_policy()`
`CORBA::PolicyErrorCode`

CORBA::PolicyErrorCode Type

```
typedef short PolicyErrorCode;

// C++
```

typedef `Short` PolicyErrorCode;

A value representing an error when creating a new `Policy`. The following constants are defined to represent the reasons a request to create a `Policy` might be invalid:

Table 4: *PolicyErrorCode Constants*

Constant	Explanation
`BAD_POLICY`	The requested `Policy` is not understood by the ORB.
`UNSUPPORTED_POLICY`	The requested `Policy` is understood to be valid by the ORB, but is not currently supported.

Table 4: *PolicyErrorCode Constants*

Constant	Explanation
BAD_POLICY_TYPE	The type of the value requested for the Policy is not valid for that PolicyType.
BAD_POLICY_VALUE	The value requested for the Policy is of a valid type but is not within the valid range for that type.
UNSUPPORTED_POLICY_VALUE	The value requested for the Policy is of a valid type and within the valid range for that type, but this valid value is not currently supported.

See Also CORBA::ORB::create_policy()

CORBA::PolicyList Sequence

```
// C++
class PolicyList {
    ...
};
```

A list of Policy objects. Policies affect an ORB's behavior.

See Also
CORBA::Policy
CORBA::Object::set_policy_overrides()
PortableServer::POA::POA_create_POA()

"About Sequences"

CORBA::PolicyType Type

```
// C++
typedef ULong PolicyType;
```

Defines the type of Policy object.

The CORBA module defines the following constant PolicyType:

```
// IDL
```

```
const PolicyType SecConstruction = 11;
// C++
static const PolicyType SecConstruction = 11;
```

Other valid constant values for a PolicyType are described with the definition of the corresponding Policy object. There are standard OMG values and IONA-specific values.

See Also

CORBA::Policy
CORBA::PolicyTypeSeq
CORBA::ORB::create_policy()
CORBA::Object::_get_policy()
CORBA::DomainManager::get_domain_policy()

CORBA::PolicyTypeSeq Sequence

```
// IDL
typedef sequence<PolicyType> PolicyTypeSeq;

// C++
class PolicyTypeSeq {
    ...
};
```

A sequence of PolicyType data types.

See Also

CORBA::Object::get_policy_overrides()
CORBA::PolicyManager::get_policy_overrides()

CORBA::PrimitiveKind Enumeration

```
// IDL
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, pk_value_base
};
typedef PrimitiveKind& PrimitiveKind_out;
```

Indicates the kind of primitive type a PrimitiveDef object represents in the interface repository.

Most kinds are self explanatory with the exception of the following:

- There are no `PrimitiveDef` objects with the kind `pk_null`.
- The kind `pk_string` represents an unbounded string.
- The kind `pk_objref` represents the IDL type `Object`.
- The kind `pk_value_base` represents the IDL type `ValueBase`.

See Also

`CORBA::PrimitiveDef`
`CORBA::Repository`

CORBA::RepositoryId Type

```
// C++
```

typedef char* RepositoryId;

A string that uniquely identifies, in the interface repository, an IDL module, interface, constant, typedef, exception, attribute, value type, value member, value box, native type, or operation.

The format of `RepositoryId` types is a short format name followed by a colon followed by characters, as follows:

`format_name``:`*`string`*

The most common format encountered is the OMG IDL format. For example:

```
IDL:Pre/B/C:5.3
```

This format contains three components separated by colons:

`IDL`	The first component is the format name, `IDL`.
`Pre/B/C`	The second component is a list of identifiers separated by '/' characters that uniquely identify a repository item and its scope. These identifiers can contain other characters including underscores (_), hyphens (-), and dots (.).
`5.3`	The third component contains major and minor version numbers separated by a dot (.).

See Also

`CORBA::Repository::lookup_id()`

CORBA::RepositoryIdSeq Sequence

```
// C++
class RepositoryIdSeq {
    ...
};
```

A sequence of RepositoryId strings in the interface repository.

See Also

CORBA::RepositoryId

"About Sequences"

CORBA::ScopedName Type

```
// C++
typedef char* ScopedName;
```

A string that specifies an IDL item's name relative to a scope in the interface repository. A ScopedName correspond to an OMG IDL scoped name.

Examples

A ScopedName that begins with "::" is an *absolute scoped name*; one that uniquely identifies an item within a repository. For example:

```
::Account::makeWithdrawal
```

A ScopedName that does not begin with "::" is a relative scoped name; one that identifies an item relative to some other item. For example:

```
makeWithdrawal
```

This example would be within the absolute scoped name of ::Account.

See Also

CORBA::Contained::absolute_name
CORBA::Container::lookup()

CORBA::ServiceDetail Structure

```
// IDL
struct ServiceDetail {
    ServiceDetailType service_detail_type;
    sequence <Octet> service_detail;
};
```

Detailed information about a single service or facility available to an ORB. Structure members consist of:

`service_detail_type`

`service_detail`

See Also CORBA::ServiceInformation

CORBA::ServiceDetailType Type

```
// C++
typedef ULong ServiceDetailType;
```

The type of service.

See Also CORBA::ServiceDetail

CORBA::ServiceInformation Structure

```
//IDL
struct ServiceInformation {
    sequence <ServiceOption> service_options;
    sequence <ServiceDetail> service_details;
};
```

Information about CORBA facilities and services that are supported by an ORB. Structure members consist of:

`service_options`

`service_details`

See Also CORBA::ORB::get_service_information()

CORBA::ServiceOption Type

```
// C++
typedef ULong ServiceOption;
```

An option for a service.

See Also CORBA::ServiceInformation

CORBA::ServiceType Type

```
typedef UShort ServiceType;
```

Used as a parameter in get_service_information() to obtain information about CORBA facilities and services that are supported by an ORB. A possible value consists of:

```
Security = 1
```

CORBA::SetOverrideType Enumeration

```
// IDL
enum SetOverrideType {SET_OVERRIDE, ADD_OVERRIDE};
```

The type of override to use in the set_policy_overrides() method when setting new policies for an object reference. Possible types consist of:

SET_OVERRIDE	Indicates that new policies are to be associated with an object reference.
ADD_OVERRIDE	Indicates that new policies are to be added to the existing set of policies and overrides for an object reference.

See Also CORBA::Object::_set_policy_overrides()

CORBA::ShortSeq Sequence

```
// C++
class ShortSeq {
    ...
};
```

A sequence of Short values used in marshalling custom value types.

See Also CORBA::DataOutputStream
CORBA::DataInputStream

"About Sequences"

CORBA::StringValue Value Box

```
// C++
class StringValue : public DefaultValueRefCountBase {
public:
    // constructors
    StringValue();
    StringValue(const StringValue& val);
    StringValue(char* str);
    StringValue(const char* str);
    StringValue(const String_var& var);
    // assignment operators
    StringValue& operator=(char* str);
    StringValue& operator=(const char* str);
    StringValue& operator=(const String_var& var);
    // accessor
    const char* _value() const;
    // modifiers
    void _value(char* str);
    void _value(const char* str);
    void _value(const String_var& var);
    // explicit argument passing conversions for underlying string
    const char* _boxed_in() const;
    char*& _boxed_inout();
    char*& _boxed_out();
    // ...other String_var methods such as overloaded
    // subscript operators, etc....
    static StringValue* _downcast(ValueBase* base);
protected:
    ~StringValue();
    ...
};
```

`StringValue` is a value box class that provides a reference-counted version of a string.

See Also

"About Value Boxes"

CORBA::StructMember Structure

```
// C++
struct StructMember {
    Identifier name;
    TypeCode type;
    IDLType type_def;
};
```

This describes an IDL structure member in the interface repository. The structure members consist of the following:

`name`	The name of the member.
`type`	The `TypeCode` for the member.
`type_def`	Identifies the definition of the type for the member.

See Also

`CORBA::StructMemberSeq`

CORBA::StructMemberSeq Sequence

```
// C++
class StructMemberSeq {
    ...
};
```

A sequence of `StructMember` objects in the interface repository.

See Also

`CORBA::StructMember`
`CORBA::ORB::create_struct_tc()`
`CORBA::ORB::create_exception_tc()`
`CORBA::Container::create_struct()`
`CORBA::Container::create_exception()`
`CORBA::StructDef::members`
`CORBA::ExceptionDef::members`
`CORBA::Initializer`

"About Sequences"

CORBA::TCKind Enumeration

```
// IDL
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_longdouble,
    tk_wchar, tk_wstring, tk_fixed,
    tk_value, tk_value_box,
    tk_native,
    tk_abstract_interface
};
```

A `TCKind` value indicates the kind of data type for a `TypeCode`. A `TypeCode` is a value that represent an invocation argument type or attribute type, such as that found in the interface repository or with a dynamic any type.

See Also

`CORBA::TypeCode::kind()`
`DynamicAny::DynStruct::current_member_kind()`
`DynamicAny::DynUnion::discriminator_kind()`
`DynamicAny::DynUnion::member_kind()`
`DynamicAny::DynValue::current_member_kind()`

CORBA::TypeDescription Structure

```
// IDL
struct TypeDescription {
    Identifier name;
    RepositoryId id;
    RepositoryId defined_in;
    VersionSpec version;
    TypeCode type;
};
```

```
// C++
struct TypeDescription {
    Identifier name;
    RepositoryId id;
    RepositoryId defined_in;
```

```
    VersionSpec version;
    TypeCode type;
};
```

This structure describes an IDL data type in the interface repository. The structure members consist of the following:

`name`	The name of the data type.
`id`	The identifier for the data type.
`defined_in`	The identifier of where the data type is defined.
`version`	The version of the data type.
`type`	The `TypeCode` of the data type.

CORBA::ULongLongSeq Sequence

```
// C++
class ULongLongSeq {
    ...
};
```

A sequence of `ULongLong` values used in marshalling custom value types.

See Also

`CORBA::DataOutputStream`
`CORBA::DataInputStream`

"About Sequences"

CORBA::ULongSeq Sequence

```
// C++
class ULongSeq {
    ...
};
```

A sequence of `ULong` values used in marshalling custom value types.

See Also

`CORBA::DataOutputStream`
`CORBA::DataInputStream`

"About Sequences"

CORBA::UnionMember Structure

```
// IDL
struct UnionMember {
    Identifier name;
    any label;
    TypeCode type;
    IDLType type_def;
};
```

```
// C++
struct UnionMember {
    Identifier name;
    any label;
    TypeCode type;
    IDLType type_def;
};
```

This structure describes an IDL union member in the interface repository. The structure members consist of the following:

`name`	The name of the union member.
`label`	The label of the union member.
`type`	The `TypeCode` of the union member.
`type_def`	The IDL data type of the union member.

See Also `CORBA::UnionMemberSeq`

CORBA::UnionMemberSeq Sequence

```
// C++
class UnionMemberSeq {
    ...
};
```

A sequence of `UnionMember` structures in the interface repository.

See Also
`CORBA::UnionMember`
`CORBA::ORB::create_union_tc()`
`CORBA::Container::create_union()`
`CORBA::UnionDef::members`

"About Sequences"

CORBA::UShortSeq Sequence

```
// C++
class UShortSeq {
    ...
};
```

A sequence of UShort values used in marshalling custom value types.

See Also

CORBA::DataOutputStream
CORBA::DataInputStream

"About Sequences"

CORBA::ValueDefSeq Sequence

```
// C++
class ValueDefSeq {
    ...
};
```

A sequence of ValueDef objects in the interface repository.

See Also

CORBA::ValueDef
CORBA::Container::create_value()

"About Sequences"

CORBA::ValueDescription Structure

```
// IDL
struct ValueDescription {
    Identifier name;
    RepositoryId id;
    boolean is_abstract;
    boolean is_custom;
    RepositoryId defined_in;
    VersionSpec version;
    RepositoryIdSeq supported_interfaces;
    RepositoryIdSeq abstract_base_values;
    boolean is_truncatable;
    RepositoryId base_value;
};
```

```
struct ValueDescription {
    Identifier name;
    RepositoryId id;
    Boolean is_abstract;
    Boolean is_custom;
    RepositoryId defined_in;
    VersionSpec version;
    RepositoryIdSeq supported_interfaces;
    RepositoryIdSeq abstract_base_values;
    Boolean is_truncatable;
    RepositoryId base_value;
};
```

The description of an IDL value type in the interface repository. Value types enable the passing of objects by value rather than just passing by reference. The structure members consist of the following:

`name`	The name of the value type.
`id`	The identifier of the value type.
`is_abstract`	True of the value type is abstract. False if the value type is not abstract.
`is_custom`	True of the value type is custom. False if the value type is not custom.
`defined_in`	The identifier of where the value type is defined.
`version`	The version of the value type.
`supported_interfaces`	
`abstract_base_values`	
`is_truncatable`	
`base_value`	

See Also `CORBA::ValueDef::describe()`

CORBA::ValueMember Structure

```
// IDL
struct ValueMember {
    Identifier name;
```

```
    RepositoryId id;
    RepositoryId defined_in;
    VersionSpec version;
    TypeCode type;
    IDLType type_def;
    Visibility access;
};
```

```
// C++
struct ValueMember {
    Identifier name;
    RepositoryId id;
    RepositoryId defined_in;
    VersionSpec version;
    TypeCode type;
    IDLType type_def;
    Visibility access;
};
```

This structure describes an IDL value type member in the interface repository. The structure members consist of the following:

`name`	The name of the value type member.
`id`	The identifier of the value type member.
`defined_in`	The identifier of where the value type member is defined.
`version`	The version of the value type member.
`type`	The `TypeCode` of the value type member.
`type_def`	The type definition of the value type member.
`access`	The accessibility of the value type member (public or private).

See Also `CORBA::ValueMemberSeq`

CORBA::ValueMemberSeq Sequence

```
// C++
class ValueMemberSeq {
    ...
};
```

A sequence of `ValueMember` structures in the interface repository.

See Also

`CORBA::ValueMember`
`CORBA::ORB::create_value_tc()`

"About Sequences"

CORBA::ValueModifier Type

```
typedef Short ValueModifier;
```

A modifier for an IDL value type in the interface repository. Possible values consist of:

`VM_NONE`	The IDL value type has no modifiers.
`VM_CUSTOM`	The IDL value type has the `custom` modifier. This specifies that the value type uses custom marshalling.
`VM_ABSTRACT`	The IDL value type has the `abstract` modifier. Value types that are abstract can not be instantiated. Essentially they are a bundle of IDL operation signatures with a purely local implementation.
`VM_TRUNCATABLE`	The IDL value type has the `truncatable` modifier. A value with a state that derives from another value with a state can be specified as truncatable. A truncatable type means the object can be truncated to the base type.

See Also

`CORBA::ORB::create_value_tc()`
`CORBA::TypeCode::type_modifier()`

CORBA::VersionSpec Type

```
// C++
typedef char* VersionSpec;
```

A string that describes a version of an IDL item in the interface repository. Version information can be associated with many IDL data types including modules, constants, types, exceptions, attributes, and operations.

See Also

`CORBA::Contained::version`

CORBA::Contained::move()
CORBA::Container

CORBA::Visibility Type

```
typedef Short Visibility;
```

Indicates the visibility of a state member of an IDL value type in the interface repository. Possible values consist of:

```
PRIVATE_MEMBER
PUBLIC_MEMBER
```

IDL value types can have state members that are either public or private. Private members are not visible to clients but are only visible to implementation code and the marshalling routines.

See Also

CORBA::ValueMember
CORBA::ValueMemberDef::access
CORBA::ValueDef::create_value_member()
CORBA::TypeCode::member_visibility()

CORBA::WCharSeq Sequence

```
// C++
class WCharSeq {
    ...
};
```

A sequence of WChar values used in marshalling custom value types.

See Also

CORBA::DataOutputStream
CORBA::DataInputStream

"About Sequences"

CORBA::WStringValue Value Box

```
// C++
class WStringValue : public DefaultValueRefCountBase {
public:
    // constructors
```

```
    WStringValue();
    WStringValue(const WStringValue& val);
    WStringValue(char* str);
    WStringValue(const char* str);
    WStringValue(const String_var& var);
    // assignment operators
    WStringValue& operator=(char* str);
    WStringValue& operator=(const char* str);
    WStringValue& operator=(const String_var& var);
    // accessor
    const char* _value() const;
    // modifiers
    void _value(char* str);
    void _value(const char* str);
    void _value(const String_var& var);
    // explicit argument passing conversions for underlying string
    const char* _boxed_in() const;
    char*& _boxed_inout();
    char*& _boxed_out();
    // ...other String_var methods such as overloaded
    // subscript operators, etc....
    static WStringValue* _downcast(ValueBase* base);
protected:
    ~WStringValue();
    ...
};
```

`WStringValue` is a value box class that provides a reference-counted version of a wide string.

See Also

"About Value Boxes"

CORBA::AliasDef Interface

The AliasDef interface describes an IDL typedef that aliases another definition in the interface repository. It is used to represent an IDL typedef.

```
// IDL in module CORBA.
interface AliasDef : TypedefDef {
    attribute IDLType original_type_def;
};
```

The following items are described for this interface:

- The describe() IDL operation
- The original_type_def attribute

See Also

CORBA::Contained
CORBA::Container::create_alias()

AliasDef::describe()

```
// IDL
Description describe();
```

Inherited from Contained (which is inherited by TypedefDef), the describe() operation returns a structure of type Contained::Description. The DefinitionKind for the kind member is dk_Alias. The value member is an any whose TypeCode is _tc_AliasDescription and whose value is a structure of type TypeDescription.

See Also

CORBA::TypedefDef::describe()

AliasDef::original_type_def Attribute

```
// IDL
attribute IDLType original_type_def;
```

Identifies the type being aliased. Modifying the `original_type_def` attribute will automatically update the `type` attribute (the `type` attribute is inherited from `TypedefDef` which in turn inherits it from `IDLType`). Both attributes contain the same information.

See Also `CORBA::IDLType::type`

CORBA::Any Class

The class Any implements the IDL basic type any, which allows the specification of values that can express an arbitrary IDL type. This allows a program to handle values whose types are not known at compile time. The IDL type any is most often used in code that uses the interface repository or the dynamic invocation interface (DII) or with CORBA services in general.

Consider the following interface:

```
// IDL
interface Example {
    void op(in any value);
};
```

A client can construct an any to contain an arbitrary type of value and then pass this in a call to op(). A process receiving an any must determine what type of value it stores and then extract the value (using the TypeCode). Refer to the *CORBA Programmer's Guide* for more details.

Methods and structures are as follows:

```
Any()                       operator=()
~Any()                      replace()
from_boolean structure      to_boolean structure
from_char structure         to_char structure
from_fixed structure        to_fixed structure
from_octet structure        to_object structure
from_string structure       to_octet structure
from_wchar structure        to_string structure
from_wstring structure      to_wchar structure
it_get_streamable()         to_wstring structure
it_set_streamable()         type()
it_take_streamable()
```

```
// C++
class IT_ART_API Any : public ITCxxMemBase
{
  public:
    Any();
```

```
Any(
    const Any& any
);
Any(
    TypeCode_ptr tc,
    void*        value,
    Boolean      release = 0
);
Any(
    IT_Streamable*            strm,
    IT_Streamable::MemPolicy policy
);
~Any();

Any& operator=(
    const Any&
);

//
// type-unsafe operations
//
void replace(
    TypeCode_ptr tc,
    void*        value,
    Boolean      release = 0
);

TypeCode_ptr type() const;

void type(
    TypeCode_ptr new_type
);

const void* value() const;

struct from_boolean {
    from_boolean(
        Boolean b
    );
    Boolean m_val;
};

struct from_octet {
```

```
    from_octet(
        Octet octet
    );
    Octet m_val;
};

struct from_char {
    from_char(
        Char c
    );
    Char m_val;
};

struct from_wchar {
    from_wchar(
        WChar c
    );
    WChar m_val;
};

struct from_string {
    from_string(
        char* s,
        ULong b,
        Boolean nocopy = 0
    );
    from_string(
        const char* s,
        ULong b
    );
    char* m_val;
    ULong m_bound;
    Boolean m_nocopy;
};

struct from_wstring {
    from_wstring(
        WChar* s,
        ULong b,
        Boolean nocopy = 0
    );
    from_wstring(
        const WChar* s,
```

```
        ULong b
    );
    WChar* m_val;
    ULong m_bound;
    Boolean m_nocopy;
};

struct from_fixed {
    from_fixed(
        const Fixed& f,
        UShort digits,
        Short scale
    );
    const Fixed& m_val;
    UShort m_digits;
    Short m_scale;
};

struct to_boolean {
    to_boolean(
        Boolean& b
    );
    Boolean& m_ref;
};

struct to_char {
    to_char(
        Char& c
    );
    Char& m_ref;
};

struct to_wchar {
    to_wchar(
        WChar& c
    );
    WChar& m_ref;
};

struct to_octet {
    to_octet(
        Octet& o
    );
```

```
    Octet& m_ref;
};

struct to_object {
    to_object(
        Object_ptr& obj
    );
    Object_ptr& m_ref;
};

struct to_string {
    to_string(
        char*& s,
        ULong  b
    );
    char*& m_ref;
    ULong  m_bound;
};

struct to_wstring {
    to_wstring(
        WChar*& s,
        ULong  b
    );
    WChar*& m_ref;
    ULong  m_bound;
};

struct to_fixed {
    to_fixed(
        Fixed& f,
        UShort digits,
        Short scale
    );
    Fixed& m_ref;
    UShort m_digits;
    Short m_scale;
};

IT_Streamable* it_get_streamable(
    Boolean make_copy = 0
) const;
```

```
    Boolean it_take_streamable(
        IT_Streamable* &strm
    );

    void it_set_streamable(
        IT_Streamable*            strm,
        IT_Streamable::MemPolicy policy
    );

  private:
   ...
}
```

Any::Any() Constructors

```
Any();
```

The default constructor creates an `Any` with a `TypeCode` of type `tk_null` and with a zero value.

```
Any(
    const Any& any
);
```

This copy constructor duplicates the `TypeCode_ptr` of any and copies the value.

```
Any(
    TypeCode_ptr tc,
    void*        value,
    Boolean      release = 0
);
```

Constructs an `Any` with a specific `TypeCode` and value. This constructor is needed for cases where it is not possible to use the default constructor and `operator<<=()`. For example, since all strings are mapped to `char*`, it is not possible to create an `Any` with a specific `TypeCode` for a bounded string.

This constructor is not type-safe; you must ensure consistency between the `TypeCode` and the actual type of the argument `value`.

```
Any(
    IT_Streamable*            strm,
    IT_Streamable::MemPolicy policy
);
```

Constructs an Any from a stream.

Parameters

`type`	A reference to a `CORBA::TypeCode`. The constructor duplicates this object reference.
`val`	The value pointer. A conforming program should make no assumptions about the lifetime of the value passed in this parameter once it has been passed to this constructor with `release=1`.
`release`	A boolean variable to decide ownership of the storage pointed to by `value`. If set to 1, the Any object assumes ownership of the storage. If the release parameter is set to 0 (the default), the calling program is responsible for managing the memory pointed to by `value`.

`IT_Streamable*`

`IT_Streamable::MemPolicy`

Examples The easiest and the type-safe way to construct an Any is to use the default constructor and then use `operator<<=()` to insert a value into the Any. For example:

```
// C++
CORBA::Short s = 10;
CORBA::Any a;
a <<= s;
```

See Also `CORBA::Any::operator<<=()`

Any::~Any() Destructor

```
~Any();
```

Destructor for an Any. Depending on the value of the Boolean release parameter to the complex constructor, it frees the value contained in the Any based on the TypeCode of the Any. It then frees the TypeCode.

See Also `CORBA::Any::Any()`

Any::from_*type* Structure

```
struct from_boolean {
    from_boolean(
        Boolean b
    );
    Boolean m_val;
};

struct from_char {
    from_char(
        Char c
    );
    Char m_val;
};

struct from_fixed {
    from_fixed(
        const Fixed& f,
        UShort digits,
        Short scale
    );
    const Fixed& m_val;
    UShort m_digits;
    Short m_scale;
};

struct from_octet {
    from_octet(
        Octet octet
    );
    Octet m_val;
};

struct from_string {
    from_string(
        char* s,
        ULong b,
        Boolean nocopy = 0
    );
    from_string(
        const char* s,
        ULong b
    );
    char* m_val;
```

```
    ULong m_bound;
    Boolean m_nocopy;
};

struct from_wchar {
    from_wchar(
        WChar c
    );
    WChar m_val;
};

struct from_wstring {
    from_wstring(
        WChar* s,
        ULong b,
        Boolean nocopy = 0
    );
    from_wstring(
        const WChar* s,
        ULong b
    );
    WChar* m_val;
    ULong m_bound;
    Boolean m_nocopy;
};
```

Inserts the specific IDL type into the any. These helper structures are nested in the any class interface to distinguish these IDL data types from each other. Because these IDL types are not required to map to distinct C++ types, another means of distinguishing them from each other is necessary so that they can be used with the type-safe any interface.

See Also `CORBA::Any::to_`*`type`*

Any::it_get_streamable()

```
IT_Streamable* it_get_streamable(
    Boolean make_copy = 0
) const;
```

Enhancement IONA-specific enhancement.

Any::it_set_streamable()

```
void it_set_streamable(
    IT_Streamable*            strm,
    IT_Streamable::MemPolicy policy
);
```

Enhancement IONA-specific enhancement.

Any::it_take_streamable()

```
Boolean it_take_streamable(
    IT_Streamable* &strm
);
```

Enhancement IONA-specific enhancement.

Any::operator=()

```
Any& operator=(
    const Any& a
);
```

The assignment operator releases its TypeCode and frees the value if necessary.

Parameters

`a` The value to duplicate. The method duplicates the TypeCode of a and deep copies the parameter's value.

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

This method is needed for cases where it is not possible to use `operator<<=()` to insert into an existing Any. For example, because all strings are mapped to `char*`, it is not possible to create an Any with a specific TypeCode for a bounded string.

Parameters

`tc`	A reference to a `CORBA::TypeCode`. The method duplicates this object reference.
`value`	The value pointer. A conforming program should make no assumptions about the lifetime of the value passed in this parameter if it has been passed to `Any::replace()` with `release=1`.
`release`	A boolean variable to decide ownership of the storage pointed to by `value`. If set to `1`, the `Any` object assumes ownership of the storage. If the release parameter is set to 0 (the default), the calling program is responsible for managing the memory pointed to by `value`.

This function is not type-safe; you must ensure consistency between the `TypeCode` and the actual type of the argument `value`.

Any::to_*type* Structure

```
struct to_boolean {
    to_boolean(
        Boolean& b
    );
    Boolean& m_ref;
};

struct to_char {
    to_char(
        Char& c
    );
    Char& m_ref;
};

struct to_fixed {
    to_fixed(
        Fixed& f,
        UShort digits,
        Short scale
    );
    Fixed& m_ref;
```

```
    UShort m_digits;
    Short m_scale;
};

struct to_object {
    to_object(
        Object_ptr& obj
    );
    Object_ptr& m_ref;
};

struct to_octet {
    to_octet(
        Octet& o
    );
    Octet& m_ref;
};

struct to_string {
    to_string(
        char*& s,
        ULong  b
    );
    char*& m_ref;
    ULong  m_bound;
};

struct to_wchar {
    to_wchar(
        WChar& c
    );
    WChar& m_ref;
};

struct to_wstring {
    to_wstring(
        WChar*& s,
        ULong  b
    );
    WChar*& m_ref;
    ULong  m_bound;
};
```

Extracts the specific IDL type from the any. These helper structures are nested in the any class interface to distinguish these IDL data types from each other. Because these IDL types are not required to map to distinct C++ types, another means of distinguishing them from each other is necessary so that they can be used with the type-safe any interface.

See Also `CORBA::Any::from_`*type*

Any::type()

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

Returns the `TypeCode` of the `Object` encapsulated within the `Any`.

```
void type(TypeCode_ptr t);
```

Sets the `TypeCode` of the `Object` encapsulated within the `Any`.

Parameters

`t`	The `TypeCode` of the object.

CORBA::ArrayDef Interface

The ArrayDef interface represents a one-dimensional array in an interface repository. A multi-dimensional array is represented by an ArrayDef with an element type that is another array definition. The final element type represents the type of element contained in the array. An instance of interface ArrayDef can be created using create_array().

```
// IDL in module CORBA.
interface ArrayDef : IDLType {
    attribute unsigned long length;
    readonly attribute TypeCode element_type;
    attribute IDLType element_type_def;
};
```

See Also

CORBA::IDLType
CORBA::ArrayDef::element_type_def
CORBA::Repository::create_array()

ArrayDef::element_type Attribute

```
// IDL
readonly attribute TypeCode element_type;
```

Identifies the type of the element contained in the array. This contains the same information as in the element_type_def attribute.

See Also

CORBA::ArrayDef::element_type_def

ArrayDef::element_type_def Attribute

```
// IDL
attribute IDLType element_type_def;
```

Describes the type of the element contained within the array. This contains the same information as in the attribute element_type attribute.

The type of elements contained in the array can be changed by changing this attribute. Changing this attribute also changes the `element_type` attribute.

See Also `CORBA::ArrayDef::element_type`

ArrayDef::length Attribute

```
// IDL
attribute unsigned long length;
```

Returns the number of elements in the array.

Specifies the number of elements in the array.

CORBA::AttributeDef Interface

The `AttributeDef` interface describes an attribute of an interface in the interface repository.

```
// IDL in module CORBA.
interface AttributeDef : Contained {
    readonly attribute TypeCode type;
    attribute IDLType type_def;
    attribute AttributeMode mode;
};
```

The inherited `describe()` method is also described.

See Also

`CORBA::Contained`
`CORBA::InterfaceDef::create_attribute()`

AttributeDef::describe()

```
// IDL
Description describe();
```

Inherited from `Contained`, the `describe()` method returns a structure of type `Contained::Description`. The `DefinitionKind` for the `kind` member of this structure is `dk_Attribute`. The value member is an any whose `TypeCode` is `_tc_AttributeDescription`. The value is a structure of type `AttributeDescription`.

See Also

`CORBA::Contained::describe()`

AttributeDef::mode Attribute

```
// IDL
attribute AttributeMode mode;

// C++
virtual AttributeMode mode() = 0;
```

Returns the mode of the attribute.

```
// C++
virtual void mode(
    AttributeMode _itvar_mode
) = 0;
```

Specifies whether the attribute is read and write (ATTR_NORMAL) or read-only (ATTR_READONLY).

AttributeDef::type Attribute

```
// IDL
readonly attribute TypeCode type;

// C++
virtual TypeCode_ptr type() = 0;
```

Returns the type of this attribute. The same information is contained in the type_def attribute.

See Also

CORBA::TypeCode
CORBA::AttributeDef::type_def

AttributeDef::type_def Attribute

```
// IDL
attribute IDLType type_def;

// C++
virtual IDLType_ptr type_def() = 0;
```

Returns the type of this attribute.

```
// C++
virtual void type_def(
    IDLType_ptr  _itvar_type_def
) = 0;
```

Describes the type for this attribute. The same information is contained in the type attribute. Changing the type_def attribute automatically changes the type attribute.

See Also

CORBA::IDLType
CORBA::AttributeDef::type

CORBA::ConstantDef Interface

Interface ConstantDef describes an IDL constant in the interface repository. The name of the constant is inherited from Contained.

```
// IDL
// in module CORBA.
interface ConstantDef : Contained {
    readonly attribute TypeCode type;
    attribute IDLType type_def;
    attribute any value;
};
```

The inherited operation describe() is also described.

See Also

CORBA::Contained
CORBA::Container::create_constant()

ConstantDef::describe()

```
// IDL
Description describe();
```

Inherited from Contained, describe() returns a structure of type Contained::Description.

The kind member is dk_Constant.

The value member is an any whose TypeCode is _tc_ConstantDescription and whose value is a structure of type ConstantDescription.

See Also

CORBA::Contained::describe()

ConstantDef::type Attribute

```
// IDL
readonly attribute TypeCode type;
```

Identifies the type of this constant. The type must be a TypeCode for one of the simple types (such as long, short, float, char, string, double, boolean, unsigned long, and unsigned short). The same information is contained in the type_def attribute.

See Also CORBA::ConstantDef::type_def

ConstantDef::type_def Attribute

```
// IDL
attribute IDLType type_def;
```

Returns the type of this constant.

Identifies the type of the constant. The same information is contained in the type attribute.

The type of a constant can be changed by changing its type_def attribute. This also changes its type attribute.

See Also CORBA::ConstantDef::type

ConstantDef::value Attribute

```
// IDL
attribute any value;
```

Returns the value of this attribute.

Contains the value for this constant. When changing the value attribute, the TypeCode of the any must be the same as the type attribute.

See Also CORBA::TypeCode

CORBA::ConstructionPolicy Interface

When new object references are created, the ConstructionPolicy object allows the caller to specify that the instance should be automatically assigned membership in a newly created policy domain. When a policy domain is created, it also has a DomainManager object associated with it. The ConstructionPolicy object provides a single operation that makes the DomainManager object.

```
// IDL in CORBA Module
interface ConstructionPolicy:  Policy {
    void make_domain_manager(
        in CORBA::InterfaceDef object_type,
        in boolean constr_policy
    );
};
```

ConstructionPolicy::make_domain_manager()

```
// IDL
void make_domain_manager(
    in CORBA::InterfaceDef object_type,
    in boolean constr_policy
);
```

This operation sets the construction policy that is to be in effect in the policy domain for which this ConstructionPolicy object is associated.

Parameters

`object_type`	The type of the objects for which domain managers will be created. If this is nil, the policy applies to all objects in the policy domain.
`constr_policy`	A value of true indicates to the ORB that new object references of the specified `object_type` are to be associated with their own separate policy domains (and associated domain manager). Once such a construction policy is set, it can be reversed by invoking `make_domain_manager()` again with the value of false. A value of false indicates the construction policy is set to associate the newly created object with the policy domain of the creator or a default policy domain.

You can obtain a reference to the newly created domain manager by calling `_get_domain_managers()` on the newly created object reference.

See Also

`CORBA::DomainManager`
`CORBA::Object::_get_domain_managers()`

CORBA::Contained Interface

Interface `Contained` is an abstract interface that describes interface repository objects that can be contained in a module, interface, or repository. It is a base interface for the following interfaces:

- ModuleDef
- InterfaceDef
- ConstantDef
- TypedefDef
- ExceptionDef
- AttributeDef
- OperationDef
- StructDef
- EnumDef
- UnionDef
- AliasDef
- ValueDef

The complete interface is shown here:

```
// IDL
// In module CORBA.
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();
```

```
    // write interface
    void move(
        in Container new_container,
        in Identifier new_name,
        in VersionSpec new_version
    );
};CORBA::Container
```

See Also CORBA::IRObject

Contained::absolute_name Attribute

```
//IDL
readonly attribute ScopedName absolute_name;
```

Gives the absolute scoped name of an object.

See Also CORBA::ScopedName

Contained::containing_repository Attribute

```
// IDL
readonly attribute Repository containing_repository;
```

Gives the Repository within which the object is contained.

Contained::defined_in Attribute

```
// IDL
attribute Container defined_in;
```

Specifies the Container for the interface repository object in which the object is contained.

An IFR object is said to be contained by the IFR object in which it is defined. For example, an InterfaceDef object is contained by the ModuleDef in which it is defined.

A second notion of contained applies to objects of type AttributeDef or OperationDef. These objects may also be said to be contained in an InterfaceDef object if they are inherited into that interface. Note that inheritance of operations and attributes across the boundaries of different modules is also allowed.

See Also CORBA::Container::contents()

Contained::describe()

```
// IDL
Description describe();
```

Returns a structure of type Contained::Description.

The kind field of the Description structure contains the same value as the def_kind attribute that Contained inherits from IRObject.

See Also CORBA::Container::describe_contents()
CORBA::DefinitionKind

Contained::Description Structure

```
// IDL
struct Description {
    DefinitionKind kind;
    any value;
};
```

This is a generic form of description which is used as a wrapper for another structure stored in the value field.

Depending on the type of the Contained object, the value field will contain a corresponding description structure:

- ConstantDescription
- ExceptionDescription
- AttributeDescription
- OperationDescription
- ModuleDescription
- InterfaceDescription
- TypeDescription

The last of these, TypeDescription is used for objects of type StructDef, UnionDef, EnumDef, and AliasDef (it is associated with interface TypedefDef from which these four listed interfaces inherit).

Contained::id Attribute

```
// IDL
attribute RepositoryId id;
```

A RepositoryId provides an alternative method of naming an object which is independent of the ScopedName.

In order to be CORBA compliant the naming conventions specified for CORBA RepositoryIds should be followed. Changing the id attribute changes the global identity of the contained object. It is an error to change the id to a value that currently exists in the contained object's Repository.

Contained::move()

```
// IDL
void move(
    in Container new_container,
    in Identifier new_name,
    in VersionSpec new_version
);
```

Removes this object from its container, and adds it to the container specified by new_container. The new container must:

- Be in the same repository.
- Be capable of containing an object of this type.
- Not contain an object of the same name (unless multiple versions are supported).

The name attribute of the object being moved is changed to that specified by the new_name parameter. The version attribute is changed to that specified by the new_version parameter.

See Also CORBA::Container

Contained::name Attribute

```
// IDL
attribute Identifier name;
```

Return or set the name of the object within its scope. For example, in the following definition:

```
// IDL
interface Example {
    void op();
};
```

the names are `Example` and `op`. A `name` must be unique within its scope but is not necessarily unique within an interface repository. The `name` attribute can be changed but it is an error to change it to a value that is currently in use within the object's `Container`.

See Also `CORBA::Contained::id`

Contained::version Attribute

```
// IDL
attribute VersionSpec version;
```

Return or set the version number for this object. Each interface object is identified by a version which distinguishes it from other versioned objects of the same name.

CORBA::Container Interface

Interface `Container` describes objects that can contain other objects in the interface repository. A `Container` can contain any number of objects derived from the `Contained` interface. Such objects include:

- `AttributeDef`
- `ConstantDef`
- `ExceptionDef`
- `InterfaceDef`
- `ModuleDef`
- `OperationDef`
- `TypedefDef`
- `ValueDef`
- `ValueMemberDef`

The interface is shown here:

```
//IDL
// In CORBA Module
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<Description> 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 StructMemberSeq members
);

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
    in boolean is_abstract
);

ValueDef create_value(
    in RepositoryId id,
    in Identifier name,
    in VersionSpec version,
    in boolean is_custom,
    in boolean is_abstract,
    in ValueDef base_value,
    in boolean is_truncatable,
    in ValueDefSeq abstract_base_values,
    in InterfaceDef supported_interface,
    in InitializerSeq initializers
);

ValueBoxDef create_value_box(
    in RepositoryId id,
    in Identifier name,
    in VersionSpec version,
    in IDLType original_type_def
);

ExceptionDef create_exception(
    in RepositoryId id,
    in Identifier name,
```

```
        in VersionSpec version,
        in StructMemberSeq members
    );

    NativeDef create_native(
        in RepositoryId id,
        in Identifier name,
        in VersionSpec version,
    );
}; // End Interface Container
```

See Also CORBA::IRObject

Container::contents()

```
// IDL
ContainedSeq contents(
    in DefinitionKind limit_type,
    in boolean exclude_inherited
);
```

Returns a sequence of Contained objects that are directly contained in (defined in or inherited into) the target object. This operation can be used to navigate through the hierarchy of definitions—starting, for example, at a Repository.

Parameters

limit_type	If set to dk_all, all of the contained interface repository objects are returned. If set to the DefinitionKind for a specific interface type, it returns only interfaces of that type. For example, if set to, dk_Operation, then it returns contained operations only.
exclude_inherited	Applies only to interfaces. If true, no inherited objects are returned. If false, objects are returned even if they are inherited.

See Also CORBA::Container::describe_contents()
CORBA::DefinitionKind

Container::create_alias()

```
// IDL
AliasDef create_alias(
    in RepositoryId id,
    in Identifier name,
    in VersionSpec version,
    in IDLType original_type
);
```

Creates a new AliasDef object within the target Container. The defined_in attribute is set to the target Container. The containing_repository attribute is set to the Repository in which the new AliasDef object is defined.

Parameters

id	The repository ID for the new AliasDef object. An exception is raised if an interface repository object with the same ID already exists within the object's repository.
name	The name for the new AliasDef object. It is an error to specify a name that already exists within the object's Container when multiple versions are not supported.
version	A version for the new AliasDef.
original_type	The original type that is being aliased.

Exceptions

BAD_PARAM, minor code 2	An object with the specified id already exists in the repository.
BAD_PARAM, minor code 3	The specified name already exists within this Container and multiple versions are not supported.
BAD_PARAM, minor code 4	The created object is not allowed by the Container. Certain interfaces derived from Container may restrict the types of definitions that they may contain.

See Also

CORBA::AliasDef

Container::create_constant()

```
// IDL
ConstantDef create_constant(
```

```
    in RepositoryId id,
    in Identifier name,
    in VersionSpec version,
    in IDLType type,
    in any value
);
```

Creates a ConstantDef object within the target Container. The defined_in attribute is set to the target Container. The containing_repository attribute is set to the Repository in which the new ConstantDef object is defined.

Parameters

id	The repository ID of the new ConstantDef object. It is an error to specify an ID that already exists within the object's repository.
name	The name of the new ConstantDef object. It is an error to specify a name that already exists within the object's Container when multiple versions are not supported.
version	The version number of the new ConstantDef object.
type	The type of the defined constant. This must be one of the simple types (long, short, ulong, ushort, float, double, char, string, boolean).
value	The value of the defined constant.

Exceptions

BAD_PARAM, minor code 2	An object with the specified id already exists in the repository.
BAD_PARAM, minor code 3	The specified name already exists within this Container and multiple versions are not supported.
BAD_PARAM, minor code 4	The created object is not allowed by the Container. Certain interfaces derived from Container may restrict the types of definitions that they may contain.

See Also CORBA::ConstantDef

Container::create_enum()

```
// IDL
EnumDef create_enum(
```

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

Creates a new EnumDef object within the target Container. The defined_in attribute is set to Container. The containing_repository attribute is set to the Repository in which the new EnumDef object is defined.

Parameters

id	The repository ID of the new EnumDef object. It is an error to specify an ID that already exists within the Repository.
name	The name of the EnumDef object. It is an error to specify a name that already exists within the object's Container when multiple versions are not supported.
version	The version number of the new EnumDef object.
members	A sequence of structures that describes the members of the new EnumDef object.

Exceptions

BAD_PARAM, minor code 2	An object with the specified id already exists in the repository.
BAD_PARAM, minor code 3	The specified name already exists within this Container and multiple versions are not supported.
BAD_PARAM, minor code 4	The created object is not allowed by the Container. Certain interfaces derived from Container may restrict the types of definitions that they may contain.

See Also CORBA::EnumDef

Container::create_exception()

```
// IDL
ExceptionDef create_exception(
    in RepositoryId id,
    in Identifier name,
    in VersionSpec version,
```

```
    in StructMemberSeq members
);
```

Creates a new ExceptionDef object within the target Container. The defined_in attribute is set to Container. The containing_repository attribute is set to the Repository in which new ExceptionDef object is defined. The type attribute of the StructMember structures is ignored and should be set to _tc_void.

Parameters

id	The repository ID of the new ExceptionDef object. It is an error to specify an ID that already exists within the object's repository.
name	The name of the new ExceptionDef object. It is an error to specify a name that already exists within the object's Container when multiple versions are not supported.
version	A version number for the new ExceptionDef object.
members	A sequence of StructMember structures that describes the members of the new ExceptionDef object.

Exceptions

BAD_PARAM, minor code 2	An object with the specified id already exists in the repository.
BAD_PARAM, minor code 3	The specified name already exists within this Container and multiple versions are not supported.
BAD_PARAM, minor code 4	The created object is not allowed by the Container. Certain interfaces derived from Container may restrict the types of definitions that they may contain.

See Also CORBA::ExceptionDef

Container::create_interface()

```
// IDL
InterfaceDef create_interface(
    in RepositoryId id,
    in Identifier name,
    in VersionSpec version,
    in InterfaceDefSeq base_interfaces
```

```
    in boolean is_abstract
);
```

Creates a new empty InterfaceDef object within the target Container. The defined_in attribute is set to Container. The containing_repository attribute is set to the Repository in which the new InterfaceDef object is defined.

Parameters

id	The repository ID of the new InterfaceDef object. It is an error to specify an ID that already exists within the object's repository.
name	The name of the new InterfaceDef object. It is an error to specify a name that already exists within the object's Container when multiple versions are not supported.
version	A version for the new InterfaceDef object.
base_interfaces	A sequence of InterfaceDef objects from which the new interface inherits.
is_abstract	If true the interface is abstract.

Exceptions

BAD_PARAM, minor code 2	An object with the specified id already exists in the repository.
BAD_PARAM, minor code 3	The specified name already exists within this Container and multiple versions are not supported.
BAD_PARAM, minor code 4	The created object is not allowed by the Container. Certain interfaces derived from Container may restrict the types of definitions that they may contain.

See Also CORBA::InterfaceDef

Container::create_module()

```
// IDL
ModuleDef create_module (
    in RepositoryId id,
    in Identifier name,
    in VersionSpec version
```

```
);
```

Creates an empty ModuleDef object within the target Container. The defined_in attribute is set to Container. The containing_repository attribute is set to the repository in which the newly created ModuleDef object is defined.

Parameters

id	The repository ID of the new ModuleDef object. It is an error to specify an ID that already exists within the object's repository.
name	The name of the new ModuleDef object. It is an error to specify a name that already exists within the object's Container when multiple versions are not supported.
version	A version for the ModuleDef object to be created.

Exceptions

BAD_PARAM, minor code 2	An object with the specified id already exists in the repository.
BAD_PARAM, minor code 3	The specified name already exists within this Container and multiple versions are not supported.
BAD_PARAM, minor code 4	The created object is not allowed by the Container. Certain interfaces derived from Container may restrict the types of definitions that they may contain.

Container::create_native()

```
// IDL
NativeDef create_native(
    in RepositoryId id,
    in Identifier name,
    in VersionSpec version,
);
```

Creates a NativeDef object within the target Container. The defined_in attribute is set to Container. The containing_repository attribute is set to the repository in which the newly created NativeDef object is defined.

Parameters

`id`	The repository ID of the new `NativeDef` object. It is an error to specify an ID that already exists within the object's repository.
`name`	The name of the new `NativeDef` object. It is an error to specify a name that already exists within the object's `Container` when multiple versions are not supported.
`version`	A version for the `NativeDef` object to be created.

Exceptions

`BAD_PARAM`, minor code 2	An object with the specified `id` already exists in the repository.
`BAD_PARAM`, minor code 3	The specified `name` already exists within this `Container` and multiple versions are not supported.
`BAD_PARAM`, minor code 4	The created object is not allowed by the `Container`. Certain interfaces derived from `Container` may restrict the types of definitions that they may contain.

Container::create_struct()

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

Creates a new `StructDef` object within the target `Container`. The `defined_in` attribute is set to `Container`. The `containing_repository` attribute is set to the repository in which the new `StructDef` object is defined. The `type` attribute of the `StructMember` structures is ignored and should be set to `_tc_void`.

Parameters

`id`	The repository ID of the new `StructDef` object. It is an error to specify an ID that already exists within the object's repository.
`name`	The name of the new `StructDef` object. It is an error to specify a name that already exists within the object's `Container` when multiple versions are not supported.
`version`	A version for the new `StructDef` object.
`members`	A sequence of `StructMember` structures that describes the members of the new `StructDef` object.

Exceptions

`BAD_PARAM`, minor code 2	An object with the specified `id` already exists in the repository.
`BAD_PARAM`, minor code 3	The specified `name` already exists within this `Container` and multiple versions are not supported.
`BAD_PARAM`, minor code 4	The created object is not allowed by the `Container`. Certain interfaces derived from `Container` may restrict the types of definitions that they may contain.

See Also `CORBA::StructDef`

Container::create_union()

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

Creates a new `UnionDef` object within the target `Container`. The `defined_in` attribute is set to the target `Container`. The `containing_repository` attribute is set to the repository in which the new `UnionDef` object is defined. The `type` attribute of the `UnionMember` structures is ignored and should be set to `_tc_void`.

Parameters

`id`	The repository ID of the new `UnionDef` object. It is an error to specify an ID that already exists within the object's repository.
`name`	The name of the new `UnionDef` object. It is an error to specify a name that already exists within the object's `Container` when multiple versions are not supported.
`version`	A version for the new `UnionDef` object.
`discriminator_type`	The type of the union discriminator.
`members`	A sequence of `UnionMember` structures that describes the members of the new `UnionDef` object.

Exceptions

`BAD_PARAM`, minor code 2	An object with the specified `id` already exists in the repository.
`BAD_PARAM`, minor code 3	The specified `name` already exists within this `Container` and multiple versions are not supported.
`BAD_PARAM`, minor code 4	The created object is not allowed by the `Container`. Certain interfaces derived from `Container` may restrict the types of definitions that they may contain.

See Also `CORBA::UnionDef`

Container::create_value()

```
// IDL
ValueDef create_value(
    in RepositoryId id,
    in Identifier name,
    in VersionSpec version,
    in boolean is_custom,
    in boolean is_abstract,
    in ValueDef base_value,
    in boolean is_truncatable,
    in ValueDefSeq abstract_base_values,
```

```
    in InterfaceDef supported_interfaces,
    in InitializerSeq initializers
);
```

Creates a new empty ValueDef object within the target Container. The defined_in attribute is set to Container. The containing_repository attribute is set to the repository in which the new ValueDef object is defined.

Parameters

id	The repository ID of the new ValueDef object. It is an error to specify an ID that already exists within the object's repository.
name	The name of the new ValueDef object. It is an error to specify a name that already exists within the object's Container when multiple versions are not supported.
version	A version for the new ValueDef object.
is_custom	If true the value type is custom.
is_abstract	If true the value type is abstract.
base_value	The base value for this value type.
is_truncatable	if true the value type is truncatable.
abstract_base_values	A sequence of ValueDef structures that describes the base values of the new ValueDef object.
supported_interfaces	The interface the value type supports.
initializers	A sequence of initializers for the new ValueDef object.

Exceptions

BAD_PARAM, minor code 2	An object with the specified id already exists in the repository.
BAD_PARAM, minor code 3	The specified name already exists within this Container and multiple versions are not supported.
BAD_PARAM, minor code 4	The created object is not allowed by the Container. Certain interfaces derived from Container may restrict the types of definitions that they may contain.

Container::create_value_box()

```
// IDL
ValueBoxDef create_value_box(
    in RepositoryId id,
    in Identifier name,
    in VersionSpec version,
    in IDLType original_type_def
);
```

Creates a new empty ValueBoxDef object within the target Container. The defined_in attribute is set to Container. The containing_repository attribute is set to the repository in which the new ValueBoxDef object is defined.

Parameters

id	The repository ID of the new ValueBoxDef object. It is an error to specify an ID that already exists within the object's repository.
name	The name of the new ValueBoxDef object. It is an error to specify a name that already exists within the object's Container when multiple versions are not supported.
version	A version for the new ValueBoxDef object.
original_type_def	The IDL data type of the value box.

Exceptions

BAD_PARAM, minor code 2	An object with the specified id already exists in the repository.
BAD_PARAM, minor code 3	The specified name already exists within this Container and multiple versions are not supported.
BAD_PARAM, minor code 4	The created object is not allowed by the Container. Certain interfaces derived from Container may restrict the types of definitions that they may contain.

Container::describe_contents()

```
// IDL
DescriptionSeq describe_contents(
```

```
    in DefinitionKind limit_type,
    in boolean exclude_inherited,
    in long max_returned_objs
);
```

Returns a sequence of structures of type Container::Description. describe_contents() is a combination of operations Contained::describe() and Container::contents().

Parameters

limit_type	If this is set to dk_all, then all of the contained interface repository objects are returned. If set to the DefinitionKind for a particular interface repository kind, it returns only objects of that kind. For example, if set to dk_Operation, then it returns contained operations only.
exclude_inherited	Applies only to interfaces. If true, no inherited objects are returned. If false, objects are returned even if they are inherited.
max_returned_objs	The number of objects that can be returned in the call. Setting a value of -1 means return all contained objects.

See Also

CORBA::Container::contents()
CORBA::Contained::describe()

Container::Description Structure

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

This structure gives the object reference of a contained object, together with its kind and value.

Each structure has the following members:

`contained_object`	The object reference, of type `Contained`, of the contained top level object. The `describe()` function can be called on an object reference, of type `Contained`, to get further information on a top level object in the repository.
`kind`	The kind of the object being described.

`value`	An any type that may contain one of the following structures:
	
	`ModuleDescription`
	`ConstantDescription`
	`TypeDescription`
	`ExceptionDescription`
	`AttributeDescription`
	`ParameterDescription`
	`OperationDescription`
	`InterfaceDescription`

See Also

`CORBA::Container::describe_contents()`
`CORBA::Contained::describe()`

Container::DescriptionSeq Sequence

```
// IDL
typedef sequence<Description> DescriptionSeq;
```

A sequence of `Container::Description` structures in the interface repository.

See Also

`CORBA::Container::Description`

"About Sequences"

Container::lookup()

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

Locates an object name within the target container. The objects can be directly or indirectly defined in or inherited into the target container.

Parameters

`search_name`	The `name` of the object to search for relative to the target container. If a relative name is given, the object is looked up relative to the target container. If `search_name` is an absolute scoped name (prefixed by '`::`'), the object is located relative to the containing `Repository`.

See Also

`CORBA::Container::lookup_name()`
`CORBA::ScopedName`

Container::lookup_name()

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

Locates an object or objects by name within the target container and returns a sequence of contained objects. The named objects can be directly or indirectly defined in or inherited into the target container. (More than one object, having the same simple name can exist within a nested scope structure.)

Parameters

`search_name`	The simple name of the object to search for.
`levels_to_search`	Defines whether the search is confined to the current object or should include all interface repository objects contained by the object. If set to `-1`, the current object and all contained interface repository objects are searched. If set to `1`, only the current object is searched.

`limit_type`	If this is set to `dk_all`, then all of the contained interface repository objects are returned. If set to the `DefinitionKind` for a particular interface repository kind, it returns only objects of that kind. For example, if set to `dk_Operation`, then it returns contained operations only.
`exclude_inherited`	Applies only to interfaces. If true, no inherited objects are returned. If false, objects are returned even if they are inherited.

See Also `CORBA::DefinitionKind`

CORBA::Context Class

Class `CORBA::Context` implements the OMG pseudo-interface `Context`. A context is intended to represent information about the client that is inconvenient to pass via parameters. An IDL operation can specify that it is to be provided with the client's mapping for particular identifiers (properties). It does this by listing these identifiers following the operation declaration in a context clause.

An IDL operation that specifies a context clause is mapped to a C++ member method that takes an extra input parameter of type `Context_ptr`, just before the `Environment` parameter. A client can optionally maintain one or more CORBA `Context` objects, that provide a mapping from identifiers (string names) to string values. A `Context` object contains a list of properties; each property consists of a name and a string value associated with that name and can be passed to a method that takes a `Context` parameter.

You can arrange `Context` objects in a hierarchy by specifying parent-child relationships among them. Then, a child passed to an operation also includes the identifiers of its parent(s). The called method can decide whether to use just the context actually passed, or the hierarchy above it.

The `Context` class is as follows:

```
// IDL
pseudo interface Context {
    readonly attribute Identifier context_name;
    readonly attribute Context parent;
    Context create_child(in Identifier child_ctx_name);
    void set_one_value(in Identifier propname, in any propvalue);
    void set_values(in NVList values);
    void delete_values(in Identifier propname);
    NVList get_values(in Identifier start_scope,
        in Flags op_flags,
        in Identifier pattern);
    };

class Context {
public:
```

```
    const char *context_name() const;
    Context_ptr parent() const;
    void create_child(
        const char *,
        Context_out
    );
    void set_one_value(
        const char *,
        const Any &
    );
    void set_values(
        NVList_ptr
    );
    void delete_values(
        const char *
    );
    void get_values(
        const char*,
        Flags,
        const char*,
        NVList_out
    );
};
```

Context::context_name()

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

Returns the name of the `Context` object. Ownership of the returned value is maintained by the `Context` and must not be freed by the caller.

See Also `CORBA::Context::create_child()`

Context::create_child()

```
void create_child(
    const char *ctx_name,
    Context_out child_ctx
);
```

Creates a child context of the current context. When a child context is passed as a parameter to an operation, any searches (using CORBA::Context::get_values()) look in parent contexts if necessary to find matching property names.

Parameters

`ctx_name`	The name of the child context. Context object names follow the rules for IDL identifiers.
`child_ctx`	The newly created context.

See Also CORBA::Context::get_values()

Context::delete_values()

```
void delete_values(
    const char *prop_name
);
```

Deletes the specified property value(s) from the context. The search scope is limited to the `Context` object on which the invocation is made.

Parameters

`prop_name`	The property name to be deleted. If `prop_name` has a trailing asterisk (*), all matching properties are deleted.

Exceptions An exception is raised if no matching property is found.

Context::get_values()

```
void get_values(
    const char* start_scope,
    Flags op_flags,
    const char* prop_name,
    NVList_out values
);
```

Retrieves the specified context property values.

Parameters

`start_scope`	The context in which the search for the values requested should be started. The name of a direct or indirect parent context may be specified to this parameter. If 0 is passed in, the search begins in the context which is the target of the call.
`op_flags`	By default, searching of identifiers propagates upwards to parent contexts; if the value `CORBA::CTX_RESTRICT_SCOPE` is specified, then searching is limited to the specified search scope or context object.
`prop_name`	If `prop_name` has a trailing asterisk (*), all matching properties and their values are returned.
`values`	An `NVList` to contain the returned property values.

Context::parent()

```
Context_ptr parent() const;
```

Returns the parent of the `Context` object. Ownership of the return value is maintained by the `Context` and must not be freed by the caller.

See Also `CORBA::Context::create_child()`

Context::set_one_value()

```
void set_one_value(
    const char * prop_name,
    const Any &value
);
```

Adds a property name and value to the `Context`. Although the value member is of type `Any`, the type of the `Any` must be a string.

Parameters

`prop_name`	The name of the property to add.
`value`	The value of the property to add.

See Also `CORBA::Context::set_values()`

Context::set_values()

```
void set_values(
    NVList_ptr values
);
```

Sets one or more property values in the `Context`. The previous value of a property, if any, is discarded.

Parameters

`values`	An `NVList` containing the `property_name:values` to add or change. In the `NVList`, the flags field must be set to zero, and the `TypeCode` associated with an attribute value must be `CORBA:: _tc_string`.

See Also `CORBA::Context::set_one_value()`

CORBA::ContextList Class

A `ContextList` allows an application to provide a list of `Context` strings that must be supplied when a dynamic invocation `Request` is invoked.

The `Context` is where the actual values are obtained by the ORB. The `ContextList` supplies only the context strings whose values are to be looked up and sent with the request invocation. The serverless `ContextList` object allows the application to specify context information in a way that avoids potentially expensive interface repository lookups for the information by the ORB during a request.

```
// IDL
pseudo interface ContextList {
    readonly attribute unsigned long count;
    void add(in string ctx);
    string item(in unsigned long index) raises (CORBA::Bounds);
    void remove(in unsigned long index) raises (CORBA::Bounds);
};

// C++
class ContextList {
public:
    ULong count();
    void add(
        const char* ctxt
    );
    void add_consume(
        char* ctxt
    );
    const char* item(
        ULong index
    );
    void remove(
        ULong index
    );
};
```

See Also

`CORBA::Object::_create_request()`
`CORBA::Request::contexts`

CORBA::ORB::create_context_list()

ContextList::add()

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

Adds a context string to the context list.

Parameters

ctxt	A string representing context information.

See Also CORBA::ContextList::add_consume()

ContextList::add_consume()

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

Adds a context string to the context list. The memory of the ctxt parameter is managed by the method. The caller cannot access the memory of ctxt after it has been passed in because this method could copy and free the original immediately.

Parameters

ctxt	A string representing context information.

See Also CORBA::ContextList::add()

ContextList::count()

```
ULong count();
```

Returns the number of context strings in the context list.

ContextList::item()

```
const char* item(
    ULong index
);
```

Returns the context item at the indexed location of the list. This return value must not be released by the caller because ownership of the return value is maintained by the `ContextList`.

Parameters

`index`	The indexed location of the desired context item.

ContextList::remove()

```
void remove(
    ULong index
);
```

Removes from the context list the context item at the indexed location.

CORBA::Current Interface

The `Current` interface is the base interface for providing information about the current thread of execution. Each ORB or CORBA service that needs its own context derives an interface from `Current` to provide information that is associated with the thread of execution in which the ORB or CORBA service is running. Interfaces that derives from `Current` include:

`PortableServer::Current`

Your application can obtain an instance of the appropriate `Current` interface by invoking `resolve_initial_references()`.

Operations on interfaces derived from `Current` access the state associated with the thread in which they are invoked, not the state associated with the thread from which the `Current` was obtained.

The IDL interface follows:

```
//IDL
module CORBA {
// interface for the Current object
    interface Current {
    };
...
};
```

See Also

`PortableServer::Current`

`CORBA::ORB::resolve_initial_references()`

CORBA::CustomMarshal Value Type

Custom value types can override the default marshaling/unmarshaling mechanism and provide their own way to encode/decode their state. If an application's value type is marked as custom, you use custom marshaling to facilitate integration of such mechanisms as existing class libraries and other legacy systems. Custom marshaling is not to be used as the standard marshaling mechanism.

`CustomMarshal` is an abstract value type that is meant to be implemented by the application programmer and used by the ORB. For example, if an application's value type needs to use custom marshaling, the IDL declares it explicitly as follows:

```
// Application-specific IDL
custom valuetype type {
    // optional state definition
    ...
};
```

When implementing a custom value type such as this, you must provide a concrete implementation of the `CustomMarshal` operations so that the ORB is able to marshal and unmarshal the value type. Each custom marshaled value type needs its own implementation.

You can use the skeletons generated by the IDL compiler as the basis for your implementation. These operations provide the streams for marshaling. Your implemented `CustomMarshal` code encapsulates the application code that can marshal and unmarshal instances of the value type over a stream using the CDR encoding. It is the responsibility of your implementation to marshal the value type's state of all of its base types (if it has any).

The implementation requirements of the streaming mechanism require that the implementations must be local because local memory addresses such as those for the marshal buffers have to be manipulated by the ORB.

Semantically, `CustomMarshal` is treated as a custom value type's implicit base class, although the custom value type does not actually inherit it in IDL. While nothing prevents you from writing IDL that inherits from

CustomMarshal, doing so will not in itself make the type custom, nor will it cause the ORB to treat it as a custom value type. You must implement these CustomMarshal operations.

Implement the following IDL operations for a custom value type:

```
// IDL in module CORBA
abstract valuetype CustomMarshal {
    void marshal(
        in DataOutputStream os
    );
    void unmarshal(
        in DataInputStream is
    );
};
```

CustomMarshal::marshal()

The operation you implement so that the ORB can marshal a custom value type.

Parameters

os	A handle to the output stream the ORB uses to marshal the custom value type.

Use the operations of the DataOutputStream in your implementation to write the custom value type's data to the stream as appropriate.

See Also CORBA::DataOutputStream

CustomMarshal::unmarshal()

The operation you implement so that the ORB can unmarshal a custom value type.

Parameters

is	A handle to the input stream the ORB uses to unmarshal the custom value type.

Use the operations of the DataInputStream in your implementation to read the custom value type's data from the stream as appropriate.

See Also CORBA::DataInputStream

CORBA::DataInputStream Value Type

The `DataInputStream` value type is a stream used by `unmarshal()` for unmarshaling an application's custom value type. You use the `DataInputStream` operations in your implementation of `unmarshal()` to read specific types of data from the stream, as defined in the custom value type. The stream takes care of breaking the data into chunks if necessary. The IDL code is as follows:

```
// IDL in module CORBA
abstract valuetype DataInputStream {
    any read_any();
    boolean read_boolean();
    char read_char();
    wchar read_wchar();
    octet read_octet();
    short read_short();
    unsigned short read_ushort();
    long read_long();
    unsigned long read_ulong();
    unsigned long long read_ulonglong();
    float read_float();
    double read_double();
    long double read_longdouble();
    string read_string();
    wstring read_wstring();
    Object read_Object();
    AbstractBase read_Abstract();
    ValueBase read_Value();
    TypeCode read_TypeCode();

    void read_any_array(
        inout AnySeq seq,
        in unsigned long offset,
        in unsigned long length
    );
    void read_boolean_array(
        inout BooleanSeq seq,
        in unsigned long offset,
```

```
    in unsigned long length
);
void read_char_array(
    inout CharSeq seq,
    in unsigned long offset,
    in unsigned long length
);
void read_wchar_array(
    inout WCharSeq seq,
    in unsigned long offset,
    in unsigned long length
);
void read_octet_array(
    inout OctetSeq seq,
    in unsigned long offset,
    in unsigned long length
);
void read_short_array(
    inout ShortSeq seq,
    in unsigned long offset,
    in unsigned long length
);
void read_ushort_array(
    inout UShortSeq seq,
    in unsigned long offset,
    in unsigned long length
);
void read_long_array(
    inout LongSeq seq,
    in unsigned long offset,
    in unsigned long length
);
void read_ulong_array(
    inout ULongSeq seq,
    in unsigned long offset,
    in unsigned long length
);
void read_ulonglong_array(
    inout ULongLongSeq seq,
    in unsigned long offset,
    in unsigned long length
);
void read_longlong_array(
```

```
        inout LongLongSeq seq,
        in unsigned long offset,
        in unsigned long length
    );
    void read_float_array(
        inout FloatSeq seq,
        in unsigned long offset,
        in unsigned long length
    );
    void read_double_array(
        inout DoubleSeq seq,
        in unsigned long offset,
        in unsigned long length
    );
};
```

Exceptions

MARSHAL	An inconsistency is detected for any operations.

See Also CORBA::CustomMarshal
CORBA::DataOutputStream

DataInputStreamread_Abstract()

```
// IDL
AbstractBase read_Abstract();
```

Returns an abstract data type from the stream.

DataInputStream::read_any()

```
// IDL
any read_any();
```

Returns an any data type from the stream.

DataInputStream::read_any_array()

```
// IDL
void read_any_array(
```

```
    inout AnySeq seq,
    in unsigned long offset,
    in unsigned long length
);
```

Reads an array of any data from the stream.

Parameters

`seq`	The sequence into which the data is placed.
`offset`	The starting index from which to read from the sequence.
`length`	The number of items to read from the array.

DataInputStream::read_boolean()

```
// IDL
boolean read_boolean();
```

Returns a `boolean` data type from the stream.

DataInputStream::read_boolean_array()

```
// IDL
void read_boolean_array(
    inout BooleanSeq seq,
    in unsigned long offset,
    in unsigned long length
);
```

Reads an array of `boolean` data from the stream.

Parameters

`seq`	The sequence into which the data is placed.
`offset`	The starting index from which to read from the sequence.
`length`	The number of items to read from the array.

DataInputStream::read_char()

```
// IDL
char read_char();
```

Returns a char data type from the stream.

DataInputStream::read_char_array()

```
// IDL
void read_char_array(
    inout CharSeq seq,
    in unsigned long offset,
    in unsigned long length
);
```

Reads an array of char data from the stream.

Parameters

seq	The sequence into which the data is placed.
offset	The starting index from which to read from the sequence.
length	The number of items to read from the array.

DataInputStream::read_double()

```
// IDL
double read_double();
```

Returns a double data type from the stream.

DataInputStream::read_double_array()

```
// IDL
void read_double_array(
    inout DoubleSeq seq,
    in unsigned long offset,
    in unsigned long length
```

```
);
```

Reads an array of `double` data from the stream.

Parameters

`seq`	The sequence into which the data is placed.
`offset`	The starting index from which to read from the sequence.
`length`	The number of items to read from the array.

DataInputStream::read_float()

```
// IDL
float read_float();
```

Returns a `float` data type from the stream.

DataInputStream::read_float_array()

```
// IDL
void read_float_array(
    inout FloatSeq seq,
    in unsigned long offset,
    in unsigned long length
);
```

Reads an array of `float` data from the stream.

Parameters

`seq`	The sequence into which the data is placed.
`offset`	The starting index from which to read from the sequence.
`length`	The number of items to read from the array.

DataInputStream::read_long()

```
// IDL
long read_long();
```

Returns a `long` data type from the stream.

DataInputStream::read_long_array()

```
// IDL
void read_long_array(
    inout LongSeq seq,
    in unsigned long offset,
    in unsigned long length
);
```

Reads an array of `long` data from the stream.

Parameters

`seq`	The sequence into which the data is placed.
`offset`	The starting index from which to read from the sequence.
`length`	The number of items to read from the array.

DataInputStream::read_longdouble()

```
// IDL
long double read_longdouble();
```

Unsupported.

DataInputStream::read_longlong_array()

```
// IDL
void read_longlong_array(
    inout LongLongSeq seq,
    in unsigned long offset,
```

```
    in unsigned long length
);
```

Reads an array of `long long` data from the stream.

Parameters

`seq`	The sequence into which the data is placed.
`offset`	The starting index from which to read from the sequence.
`length`	The number of items to read from the array.

DataInputStream::read_Object()

```
// IDL
Object read_Object();
```

Returns an `Object` (object reference) data type from the stream.

DataInputStream::read_octet()

```
// IDL
octet read_octet();
```

Returns an `octet` data type from the stream.

DataInputStream::read_octet_array()

```
// IDL
void read_octet_array(
    inout OctetSeq seq,
    in unsigned long offset,
    in unsigned long length
);
```

Reads an array of octet data from the stream.

Parameters

seq	The sequence into which the data is placed.
offset	The starting index from which to read from the sequence.
length	The number of items to read from the array.

DataInputStream::read_short()

```
// IDL
short read_short();
```

Returns a short data type from the stream.

DataInputStream::read_short_array()

```
// IDL
void read_short_array(
    inout ShortSeq seq,
    in unsigned long offset,
    in unsigned long length
);
```

Reads an array of short data from the stream.

Parameters

seq	The sequence into which the data is placed.
offset	The starting index from which to read from the sequence.
length	The number of items to read from the array.

DataInputStream::read_string()

```
// IDL
string read_string();
```

Returns a string data type from the stream.

DataInputStream::read_TypeCode()

```
// IDL
TypeCode read_TypeCode();
```

Returns a TypeCode data type from the stream.

DataInputStream::read_ulong()

```
// IDL
unsigned long read_ulong();
```

Returns an unsigned long data type from the stream.

DataInputStream::read_ulong_array()

```
// IDL
void read_ulong_array(
    inout ULongSeq seq,
    in unsigned long offset,
    in unsigned long length
);
```

Reads an array of unsigned long data from the stream.

Parameters

seq	The sequence into which the data is placed.
offset	The starting index from which to read from the sequence.
length	The number of items to read from the array.

DataInputStream::read_ulonglong()

```
// IDL
unsigned long long read_ulonglong();
```

Returns an unsigned long long data type from the stream.

DataInputStream::read_ulonglong_array()

```
// IDL
void read_ulonglong_array(
    inout ULongLongSeq seq,
    in unsigned long offset,
    in unsigned long length
);
```

Reads an array of unsigned long long data from the stream.

Parameters

seq	The sequence into which the data is placed.
offset	The starting index from which to read from the sequence.
length	The number of items to read from the array.

DataInputStream::read_ushort()

```
// IDL
unsigned short read_ushort();
```

Returns an unsigned short data type from the stream.

DataInputStream::read_ushort_array()

```
// IDL
void read_ushort_array(
    inout UShortSeq seq,
    in unsigned long offset,
    in unsigned long length
);
```

Reads an array of unsigned short data from the stream.

Parameters

`seq`	The sequence into which the data is placed.
`offset`	The starting index from which to read from the sequence.
`length`	The number of items to read from the array.

DataInputStream::read_Value()

```
// IDL
ValueBase read_Value();
```

Returns a value type from the stream.

DataInputStream::read_wchar()

```
// IDL
wchar read_wchar();
```

Returns a `wchar` data type from the stream.

DataInputStream::read_wchar_array()

```
// IDL
void read_wchar_array(
    inout WCharSeq seq,
    in unsigned long offset,
    in unsigned long length
);
```

Reads an array of `wchar` data from the stream.

Parameters

`seq`	The sequence into which the data is placed.
`offset`	The starting index from which to read from the sequence.
`length`	The number of items to read from the array.

DataInputStream::read_wstring()

```
// IDL
wstring read_wstring();
```

Returns a `wstring` data type from the stream.

CORBA::DataOutputStream Value Type

The `DataOutputStream` value type is a stream used by `marshal()` for marshaling an application's custom value type. You use the `DataOutputStream` operations in your implementation of `marshal()` to write specific types of data to the stream, as defined in the custom value type. The stream takes care of breaking the data into chunks if necessary. The IDL code is as follows:

```
//IDL in module CORBA
abstract valuetype DataOutputStream {
    void write_any( in any value );
    void write_boolean( in boolean value );
    void write_char( in char value );
    void write_wchar( in wchar value );
    void write_octet( in octet value );
    void write_short( in short value );
    void write_ushort( in unsigned short value );
    void write_long( in long value );
    void write_ulong( in unsigned long value );
    void write_longlong( in long long value );
    void write_ulonglong( in unsigned long long value );
    void write_float( in float value );
    void write_double( in double value );
    void write_longdouble( in long double value );
    void write_string( in string value );
    void write_wstring( in wstring value );
    void write_Object( in Object value );
    void write_Abstract( in AbstractBase value );
    void write_Value( in ValueBase value );
    void write_TypeCode( in TypeCode value );
    void write_any_array(
        in AnySeq seq,
        in unsigned long offset,
        in unsigned long length );
    void write_boolean_array(
```

```
        in BooleanSeq seq,
        in unsigned long offset,
        in unsigned long length );
    void write_char_array(
        in CharSeq seq,
        in unsigned long offset,
        in unsigned long length );
    void write_wchar_array(
        in WcharSeq seq,
        in unsigned long offset,
        in unsigned long length );
    void write_octet_array(
        in OctetSeq seq,
        in unsigned long offset,
        in unsigned long length );
    void write_short_array(
        in ShortSeq seq,
        in unsigned long offset,
        in unsigned long length );
    void write_ushort_array(
        in UShortSeq seq,
        in unsigned long offset,
        in unsigned long length );
    void write_long_array(
        in LongSeq seq,
        in unsigned long offset,
        in unsigned long length );
    void write_ulong_array(
        in ULongSeq seq,
        in unsigned long offset,
        in unsigned long length );
    void write_ulonglong_array(
        in ULongLongSeq seq,
        in unsigned long offset,
        in unsigned long length );
    void write_longlong_array(
        in LongLongSeq seq,
        in unsigned long offset,
        in unsigned long length );
    void write_float_array(
        in FloatSeq seq,
        in unsigned long offset,
        in unsigned long length );
```

```
    void write_double_array(
        in DoubleSeq seq,
        in unsigned long offset,
        in unsigned long length );
};
```

Exceptions

MARSHAL	An inconsistency is detected for any operations.

See Also

CORBA::CustomMarshal
CORBA::DataInputStream

DataOutputStream::write_Abstract()

```
// IDL
void write_Abstract(
    in AbstractBase value
);
```

Writes an abstract data type to the stream.

Parameters

value	The value written to the stream.

DataOutputStream::write_any()

```
// IDL
void write_any(
    in any value
);
```

Writes an any data type to the stream.

Parameters

value	The value written to the stream.

DataOutputStream::write_any_array()

```
// IDL
void write_any_array(
    in AnySeq seq,
    in unsigned long offset,
    in unsigned long length
);
```

Writes an array of any data to the stream.

Parameters

seq	The sequence of data to write to the stream.
offset	The offset in seq from which to start writing data.
length	The number of data items to write.

DataOutputStream::write_boolean()

```
// IDL
void write_boolean(
    in boolean value
);
```

Writes a boolean data type to the stream.

Parameters

value	The value written to the stream.

DataOutputStream::write_boolean_array()

```
// IDL
void write_boolean_array(
    in BooleanSeq seq,
    in unsigned long offset,
    in unsigned long length
);
```

Writes an array of boolean data to the stream.

Parameters

`seq`	The sequence of data to write to the stream.
`offset`	The offset in `seq` from which to start writing data.
`length`	The number of data items to write.

DataOutputStream::write_char()

```
// IDL
void write_char(
    in char value
);
```

Writes a `char` data type to the stream.

Parameters

`value`	The value written to the stream.

DataOutputStream::write_char_array()

```
// IDL
void write_char_array(
    in CharSeq seq,
    in unsigned long offset,
    in unsigned long length
);
```

Writes an array of `char` data to the stream.

Parameters

`seq`	The sequence of data to write to the stream.
`offset`	The offset in `seq` from which to start writing data.
`length`	The number of data items to write.

DataOutputStream::write_double()

```
// IDL
void write_double(
    in double value
);
```

Writes a `double` data type to the stream.

Parameters

`value`	The value written to the stream.

DataOutputStream::write_double_array()

```
// IDL
void write_double_array(
    in DoubleSeq seq,
    in unsigned long offset,
    in unsigned long length
);
```

Writes an array of `double` data to the stream.

Parameters

`seq`	The sequence of data to write to the stream.
`offset`	The offset in `seq` from which to start writing data.
`length`	The number of data items to write.

DataOutputStream::write_float()

```
// IDL
void write_float(
    in float value
);
```

Writes a `float` data type to the stream.

Parameters

`value`	The value written to the stream.

DataOutputStream::write_float_array()

```
// IDL
void write_float_array(
    in FloatSeq seq,
    in unsigned long offset,
    in unsigned long length
);
```

Writes an array of `float` data to the stream.

Parameters

`seq`	The sequence of data to write to the stream.
`offset`	The offset in `seq` from which to start writing data.
`length`	The number of data items to write.

DataOutputStream::write_long()

```
// IDL
void write_long(
    in long value
);
```

Writes a `long` data type to the stream.

Parameters

`value`	The value written to the stream.

DataOutputStream::write_long_array()

```
// IDL
void write_long_array(
    in LongSeq seq,
    in unsigned long offset,
    in unsigned long length
);
```

Writes an array of `long` data to the stream.

Parameters

`seq`	The sequence of data to write to the stream.
`offset`	The offset in `seq` from which to start writing data.
`length`	The number of data items to write.

DataOutputStream::write_longdouble()

```
// IDL
void write_longdouble(
    in long double value
);
```

Writes a `long double` data type to the stream.

Parameters

`value`	The value written to the stream.

DataOutputStream::write_longlong()

```
// IDL
void write_longlong(
    in long long value
);
```

Writes a `long long` data type to the stream.

Parameters

`value`	The value written to the stream.

DataOutputStream::write_longlong_array()

```
// IDL
void write_longlong_array(
    in LongLongSeq seq,
    in unsigned long offset,
    in unsigned long length
```

```
);
```

Writes an array of `long long` data to the stream.

Parameters

`seq`	The sequence of data to write to the stream.
`offset`	The offset in `seq` from which to start writing data.
`length`	The number of data items to write.

DataOutputStream::write_Object()

```
// IDL
void write_Object(
    in Object value
);
```

Writes an `Object` data type (object reference) to the stream.

Parameters

`value`	The value written to the stream.

DataOutputStream::write_octet()

```
// IDL
void write_octet(
    in octet value
);
```

Writes an `octet` data type to the stream.

Parameters

`value`	The value written to the stream.

DataOutputStream::write_octet_array()

```
// IDL
void write_octet_array(
```

```
    in OctetSeq seq,
    in unsigned long offset,
    in unsigned long length
);
```

Writes an array of octet data to the stream.

Parameters

seq	The sequence of data to write to the stream.
offset	The offset in seq from which to start writing data.
length	The number of data items to write.

DataOutputStream::write_short()

```
// IDL
void write_short(
    in short value
);
```

Writes a short data type to the stream.

Parameters

value	The value written to the stream.

DataOutputStream::write_short_array()

```
// IDL
void write_short_array(
    in ShortSeq seq,
    in unsigned long offset,
    in unsigned long length
);
```

Writes an array of short data to the stream.

Parameters

seq	The sequence of data to write to the stream.

`offset`	The offset in `seq` from which to start writing data.
`length`	The number of data items to write.

DataOutputStream::write_string()

```
// IDL
void write_string(
    in string value
);
```

Writes a `string` data type to the stream.

Parameters

`value`	The value written to the stream.

DataOutputStream::write_TypeCode()

```
// IDL
void write_TypeCode(
    in TypeCode value
);
```

Writes a TypeCode data type to the stream.

Parameters

`value`	The value written to the stream.

DataOutputStream::write_ulong()

```
// IDL
void write_ulong(
    in unsigned long value
);
```

Writes an `unsigned long` data type to the stream.

Parameters

`value`	The value written to the stream.

DataOutputStream::write_ulong_array()

```
// IDL
void write_ulong_array(
    in ULongSeq seq,
    in unsigned long offset,
    in unsigned long length
);
```

Writes an array of `unsigned long` data to the stream.

Parameters

`seq`	The sequence of data to write to the stream.
`offset`	The offset in `seq` from which to start writing data.
`length`	The number of data items to write.

DataOutputStream::write_ulonglong()

```
// IDL
void write_ulonglong(
    in unsigned long long value
);
```

Writes an `unsigned long long` data type to the stream.

Parameters

`value`	The value written to the stream.

DataOutputStream::write_ulonglong_array()

```
// IDL
void write_ulonglong_array(
    in ULongLongSeq seq,
```

```
    in unsigned long offset,
    in unsigned long length
);
```

Writes an array of `unsigned long long` data to the stream.

Parameters

`seq`	The sequence of data to write to the stream.
`offset`	The offset in `seq` from which to start writing data.
`length`	The number of data items to write.

DataOutputStream::write_ushort()

```
// IDL
void write_ushort(
    in unsigned short value
);
```

Writes an `unsigned short` data type to the stream.

Parameters

`value`	The value written to the stream.

DataOutputStream::write_ushort_array()

```
// IDL
void write_ushort_array(
    in UShortSeq seq,
    in unsigned long offset,
    in unsigned long length
);
```

Writes an array of `unsigned short` data to the stream.

Parameters

`seq`	The sequence of data to write to the stream.

`offset`	The offset in seq from which to start writing data.
`length`	The number of data items to write.

DataOutputStream::write_Value()

```
// IDL
void write_Value(
    in ValueBase value
);
```

Writes a value type to the stream.

Parameters

`value`	The value written to the stream.

DataOutputStream::write_wchar()

```
// IDL
void write_wchar(
    in wchar value
);
```

Writes a `wchar` data type to the stream.

Parameters

`value`	The value written to the stream.

DataOutputStream::write_wchar_array()

```
// IDL
void write_wchar_array(
    in WCharSeq seq,
    in unsigned long offset,
    in unsigned long length
);
```

Writes an array of `wchar` data to the stream.

Parameters

`seq`	The sequence of data to write to the stream.
`offset`	The offset in `seq` from which to start writing data.
`length`	The number of data items to write.

DataOutputStream::write_wstring()

```
// IDL
void write_wstring(
    in wstring value
);
```

Writes a `wstring` data type to the stream.

Parameters

`value`	The value written to the stream.

CORBA::DomainManager Interface

The DomainManager interface provides an operation to find the Policy objects associated with a policy domain. Each policy domain includes one policy domain manager object (DomainManager). The DomainManager has associated with it the policy objects for that domain and it records the membership of the domain.

```
// IDL in CORBA Module
interface DomainManager {
    Policy get_domain_policy(
        in PolicyType policy_type
    );
};
```

A *policy domain* is a set of objects with an associated set of policies. These objects are the *policy domain members*. The policies represent the rules and criteria that constrain activities of the objects of the policy domain. Policy domains provide a higher granularity for policy management than an individual object instance provides.

When a new object reference is created, the ORB implicitly associates the object reference (and hence the object that it is associated with) with one or more policy domains, thus defining all the policies to which the object is subject. If an object is simultaneously a member of more than one policy domain, it is governed by all policies of all of its domains.

Each DomainManager has a ConstructionPolicy object associated with it which has the make_domain_manager() operation. This operation controls whether a new DomainManager is created or an existing one is used when the new object reference is created.

The DomainManager does not include operations to manage domain membership, structure of domains, or to manage which policies are associated with domains. However, because a DomainManager is a CORBA object, it has access to the CORBA::Object interface, which is available to all CORBA objects. The Object interface includes the following related operations:

_get_domain_managers() allows your applications to retrieve the domain managers and hence the security and other policies applicable to individual objects that are members of the policy domain.

You can also obtain an object's policy using _get_policy().

DomainManager::get_domain_policy()

```
Policy get_domain_policy (
    in PolicyType policy_type
);
```

Returns a reference to the policy object of the specified policy type for objects in this policy domain.

Parameters

policy_type	The type of policy for objects in the domain which the application wants to administer.

There may be several policies associated with a domain, with a policy object for each. There is at most one policy of each type associated with a policy domain. The policy objects are thus shared between objects in the domain, rather than being associated with individual objects. Consequently, if an object needs to have an individual policy, then it must be a singleton member of a policy domain.

Exceptions

INV_POLICY	The value of policy type is not valid either because the specified type is not supported by this ORB or because a policy object of that type is not associated with this object.

See Also

CORBA::Policy
CORBA::Object::_get_domain_managers()
CORBA::ConstructionPolicy::make_domain_manager()
CORBA::Object::_get_policy()

CORBA::EnumDef Interface

Interface `EnumDef` describes an IDL enumeration definition in the interface repository.

```
// IDL in module CORBA.
interface EnumDef : TypedefDef {
    attribute EnumMemberSeq members;
};
```

The inherited operation `describe()` is also described.

EnumDef::describe()

```
// IDL
Description describe();
```

Inherited from `Contained` (which `TypedefDef` inherits), `describe()` returns a structure of type `Contained::Description`. The `DefinitionKind` for the description's `kind` member is `dk_Enum`. The value member is an any whose `TypeCode` is `_tc_TypeDescription` and whose value is a structure of type `TypeDescription`. The `type` field of the `struct` gives the `TypeCode` of the defined enumeration.

See Also `CORBA::TypedefDef::describe()`

EnumDef::members Attribute

```
// IDL
attribute EnumMemberSeq members;
```

Returns or changes the enumeration's list of identifiers (its set of enumerated constants).

See Also `CORBA::Identifier`

CORBA::Environment Class

The Environment class provides a way to handle exceptions in situations where true exception-handling mechanisms are unavailable or undesirable.

For example, in the DII you can use the Environment class to pass information between a client and a server where the C++ host compiler does not support C++ exception handling.

```
// IDL
pseudo interface Environment {
    attribute exception exception;
    void clear();
};

// C++
class Environment {
public:
    void exception(Exception* e);
    Exception *exception() const;
    void clear();

    _duplicate(Environment_ptr obj);
    _nil();
};
```

See Also CORBA::ORB::create_environment()

Environment::clear()

```
//C++
void clear();
```

Deletes the Exception, if any, contained in the Environment. This is equivalent to passing zero to exception(). It is not an error to call clear() on an Environment that holds no exception.

See Also CORBA::Environment::exception()

Environment::_duplicate()

```
// C++
static Environment_ptr _duplicate(
    Environment_ptr obj
);
```

Returns a reference to `obj` and increments the reference count of `obj`.

See Also CORBA::release()

Environment::exception()

Extracts the exception contained in the `Environment` object.

```
// C++
Exception* exception() const;
```

Returns the exception, if any, raised by a preceding remote request. The returned pointer refers to memory owned by the `Environment` and must not be freed by the caller. Once the `Environment` is destroyed, the pointer is no longer valid.

```
// C++
void exception(
    Exception* e
);
```

Assigns the `Exception` denoted by the parameter `e` into the `Environment`.

Parameters

e	The `Exception` assigned to the `Environment` The `Environment` does not copy the parameter but it assumes ownership of it. The `Exception` must be dynamically allocated.

Examples Following is an example of usage:

```
// C++
CORBA::Environment env;
A_var obj = ...
obj->op(env);
if(CORBA::Exception* ex = env.exception()) {
```

```
    ...
}
```

You can make a number of remote requests using the same `Environment` variable. Each attempt at a request immediately aborts if the `Exception` referenced by the `Environment` is not 0, and thus any failure causes subsequent requests not to be attempted, until the exception pointer is reset to 0. Any failed call may also generate one or more null proxies, so that any attempts to use these proxies prior to the end of a `TRY` macro (for non-exception handling compilers) are null operations.

The `Environment` retains ownership of the `Exception` returned. Thus, once the `Environment` is destroyed, or its `Exception` cleared, the reference is no longer valid.

See Also `CORBA::Environment::clear()`

Environment::_nil()

```
// C++
static Environment_ptr _nil();
```

Returns a nil object reference for an `Environment` object.

See Also `CORBA::is_nil()`

CORBA::Exception Class

Details of this class can be found in the CORBA specification. The C++ Language Mapping document provides the following explanation of the `CORBA::Exception` class:

```
// C++
class Exception
{
     public:
     virtual ~Exception();
     virtual void _raise() const = 0;
     virtual const char * _name() const;
     virtual const char * _rep_id() const;
};
```

The `Exception` base class is abstract and may not be instantiated except as part of an instance of a derived class. It supplies one pure virtual function to the exception hierarchy: the `_raise()` function. This function can be used to tell an exception instance to throw itself so that a catch clause can catch it by a more derived type.

Each class derived from `Exception` implements `_raise()` as follows:

```
// C++
void SomeDerivedException::_raise() const
{
     throw *this;
}
```

For environments that do not support exception handling, please refer to Section 1.42.2, "Without Exception Handling," on page 1-169 of the CORBA specification for information about the `_raise()` function.

The `_name()` function returns the unqualified (unscoped) name of the exception. The `_rep_id()` function returns the repository ID of the exception.

CORBA::ExceptionDef Interface

Interface ExceptionDef describes an IDL exception in the interface repository. It inherits from interface Contained and Container.

```
// IDL in module CORBA.
interface ExceptionDef : Contained, Container {
    readonly attribute TypeCode type;
    attribute StructMemberSeq members;
};
```

The inherited operation describe() is also described.

See Also

CORBA::Contained
CORBA::Container

ExceptionDef::describe()

```
// IDL
Description describe();
```

Inherited from Contained, describe() returns a structure of type Contained::Description.

The DefinitionKind for the kind member of this structure is dk_Exception. The value member is an any whose TypeCode is _tc_ExceptionDescription and whose value is a structure of type ExceptionDescription.

The type field of the ExceptionDescription structure gives the TypeCode of the defined exception.

See Also

CORBA::Contained::describe()
CORBA::TypeCode

ExceptionDef::members Attribute

```
// IDL
attribute StructMemberSeq members;
```

In a sequence of StructMember structures, the members attribute describes the exception's members.

The members attribute can be modified to change the structure's members. Only the name and type_def fields of each StructMember should be set. The type field should be set to _tc_void, and it will be set automatically to the TypeCode of the type_def field.

See Also

CORBA::StructDef
CORBA::ExceptionDef::type

ExceptionDef::type Attribute

```
// IDL
readonly attribute TypeCode type;
```

The type of the exception (from which the definition of the exception can be understood). The TypeCode kind for an exception is tk_except.

See Also

CORBA::TypeCode
CORBA::ExceptionDef::members

CORBA::ExceptionList Class

An ExceptionList object allows an application to provide a list of TypeCodes for all application-specific (user-defined) exceptions that may result when a dynamic invocation Request is invoked. This server-less ExceptionList object allows the ORB to avoid potentially expensive interface repository lookups for the exception information during a request.

```
// PIDL
pseudo interface ExceptionList {
    readonly attribute unsigned long count;
    void add(in TypeCode exc);
    TypeCode item(in unsigned long index) raises(Bounds);
    void remove(in unsigned long index) raises(Bounds);
};
// C++
class ExceptionList {
public:
    ULong count();
    void add(TypeCode_ptr tc);
    void add_consume(TypeCode_ptr tc);
    TypeCode_ptr item(ULong index);
    void remove(ULong index);
};
```

See Also

CORBA::Object::_create_request()
CORBA::Request::exceptions
CORBA::ORB::create_exception_list()

ExceptionList::add()

```
// C++
void add(
    TypeCode_ptr tc
);
```

Adds a TypeCode to the exception list.

Parameters

`tc`	A `TypeCode` representing exception information.

See Also `CORBA::ExceptionList::add_consume()`

ExceptionList::add_consume()

```
// C++
void add_consume(
    TypeCode_ptr tc
);
```

Adds an item to the exception list. The memory of the `tc` parameter is managed by the function. The caller cannot access the memory of `tc` after it has been passed in because this function could copy and free the original immediately.

Parameters

`tc`	A `TypeCode` representing exception information.

See Also `CORBA::ExceptionList::add()`

ExceptionList::count()

```
// C++
ULong count();
```

Returns the number of items in the exception list.

ExceptionList::item()

```
// C++
TypeCode_ptr item(
    ULong index
);
```

Returns the exception item at the indexed location of the list. This return value must not be released by the caller because ownership of the return value is maintained by the `ExceptionList`.

Parameters

`index`	The indexed location of the desired item.

ExceptionList::remove()

```
// C++
void remove(
    ULong index
);
```

Removes from the exception list the item at the indexed location.

Parameters

`index`	The indexed location of the desired item.

CORBA::FixedDef Interface

The FixedDef interface describes an IDL fixed-point type in the interface repository. A fixed-point decimal literal consists of an integer part, a decimal point, a fraction part, and a `d` or `D`.

```
// IDL in module CORBA.
interface FixedDef : IDLType {
    attribute unsigned short digits;
    attribute short scale;
```

};The inherited `IDLType` attribute is a tk_fixed `TypeCode`, which describes a fixed-point decimal number.

See Also `CORBA::Repository::create_fixed()`

FixedDef::digits Attribute

```
// IDL
attribute unsigned short digits;
```

The `digits` attribute specifies the total number of decimal digits in the fixed-point number, and must be in the range of 1 to 31, inclusive.

FixedDef::scale Attribute

```
// IDL
attribute short scale;
```

The `scale` attribute specifies the position of the decimal point.

CORBA::IDLType Interface

The abstract base interface `IDLType` describes interface repository objects that represent IDL types. These types include interfaces, type definitions, structures, unions, enumerations, and others. Thus, the `IDLType` is a base interface for the following interfaces:

- `ArrayDef`
- `AliasDef`
- `EnumDef`
- `FixedDef`
- `InterfaceDef`
- `NativeDef`
- `PrimitiveDef`
- `SequenceDef`
- `StringDef`
- `StructDef`
- `TypedefDef`
- `UnionDef`
- `ValueBoxDef`
- `ValueDef`
- `WstringDef`

The `IDLType` provides access to the `TypeCode` describing the type, and is used in defining other interfaces wherever definitions of IDL types must be referenced.

```
// IDL in module CORBA.
interface IDLType : IRObject {
    readonly attribute TypeCode type;
};
```

See Also

`CORBA::IRObject`
`CORBA::TypeCode`
`CORBA::TypedefDef`

IDLType::type Attribute

```
//IDL
readonly attribute TypeCode type;
```

Encodes the type information of an interface repository object. Most type information can also be extracted using operations and attributes defined for derived types of the IDLType.

See Also CORBA::TypeCode

CORBA::InterfaceDef Interface

`InterfaceDef` describes an IDL interface definition in the interface repository. It may contain lists of constants, typedefs, exceptions, operations, and attributes. it inherits from the interfaces `Container`, `Contained`, and `IDLType`.

Calling `_get_interface()` on a reference to an object (*`interface`*`_ptr` or *`interface`*`_var`) returns a reference to the `InterfaceDef` object that defines the CORBA object's interface.

```
// IDL in module CORBA.
interface InterfaceDef : Container, Contained, IDLType {
    // read/write interface
    attribute InterfaceDefSeq base_interfaces;

    // read interface
    boolean is_a(
        in RepositoryId interface_id
    );

    struct FullInterfaceDescription {
        Identifier name;
        RepositoryId 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
        );
    }; // End interface InterfaceDef
```

The inherited operation describe() is also described.

See Also

CORBA::Contained
CORBA::Container
CORBA::Object::_get_interface()

InterfaceDef::base_interfaces Attribute

```
// IDL
attribute InterfaceDefSeq base_interfaces;
```

The base_interfaces attribute lists in a sequence of InterfaceDef objects the interfaces from which this interface inherits.

The inheritance specification of an InterfaceDef object can be changed by changing its base_interfaces attribute.

Exceptions

BAD_PARAM, minor code 5	The name of any definition contained in the interface conflicts with the name of a definition in any of the base interfaces.

See Also

CORBA::Object::_get_interface()

InterfaceDef::create_attribute()

```
// IDL
AttributeDef create_attribute(
    in RepositoryId id,
    in Identifier name,
    in VersionSpec version,
    in IDLType type,
    in AttributeMode mode
);
```

Creates a new AttributeDef within the target InterfaceDef. The defined_in attribute of the new AttributeDef is set to the target InterfaceDef.

Parameters

id	The identifier of the new attribute. It is an error to specify an id that already exists within the target object's repository.
name	The name of the attribute. It is an error to specify a name that already exists within this InterfaceDef.
version	A version for this attribute.
type	The IDLType for this attribute.
mode	Specifies whether the attribute is read only (ATTR_READONLY) or read/write (ATTR_NORMAL).

Exceptions

BAD_PARAM, minor code 2	An object with the specified id already exists in the repository.
BAD_PARAM, minor code 3	An object with the same name already exists in this InterfaceDef.

See Also CORBA::AttributeDef

InterfaceDef::create_operation()

```
// IDL
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
);
```

Creates a new OperationDef within the target InterfaceDef. The defined_in attribute of the new OperationDef is set to the target InterfaceDef.

Parameters

id	The identifier of the new attribute. It is an error to specify an id that already exists within the target object's repository.
name	The name of the attribute. It is an error to specify a name that already exists within this InterfaceDef.
version	A version number for this operation.
result	The return type for this operation.
mode	Specifies whether this operation is normal (OP_NORMAL) or oneway (OP_ONEWAY).
params	A sequence of ParameterDescription structures that describes the parameters to this operation.
exceptions	A sequence of ExceptionDef objects that describes the exceptions this operation can raise.
contexts	A sequence of context identifiers for this operation.

See Also CORBA::OperationDef
CORBA::ExceptionDef

InterfaceDef::describe()

```
// IDL
Description describe();
```

Inherited from Contained, describe() returns a structure of type Contained::Description. The DefinitionKind for the kind member is dk_Interface. The value member is an any whose TypeCode is _tc_InterfaceDescription and whose value is a structure of type InterfaceDescription.

See Also CORBA::Contained::describe()

InterfaceDef::describe_interface()

```
// IDL
FullInterfaceDescription describe_interface();
```

Returns a description of the interface, including its operations, attributes, and base interfaces in a `FullInterfaceDescription`.

Details of exceptions and contexts can be determined via the returned sequence of `OperationDescription` structures.

See Also

`CORBA::OperationDef::describe()`
`CORBA::AttributeDef::describe()`

InterfaceDef::FullInterfaceDescription Structure

```
// IDL
struct FullInterfaceDescription {
    Identifier name;
    RepositoryId id;
    RepositoryId defined_in;
    VersionSpec version;
    OpDescriptionSeq operations;
    AttrDescriptionSeq attributes;
    RepositoryIdSeq base_interfaces;
    TypeCode type;
};
```

Describes an interface including its operations and attributes.

`name`	The name of the interface.
`id`	An identifier of the interface.
`defined_in`	The identifier where the interface is defined.
`version`	The version of the interface.
`operations`	A sequence of interface operations.
`attributes`	A sequence of interface attributes.
`base_interfaces`	A sequence of base interfaces from which this interface is derived.
`type`	The type of the interface.

See Also CORBA::InterfaceDef::describe_interface()

InterfaceDef::is_a()

```
// IDL
boolean is_a(
    in RepositoryId interface_id
);
```

Returns TRUE if the interface is either identical to or inherits (directly or indirectly) from the interface represented by `interface_id`. Otherwise the operation returns FALSE.

Parameters

`interface_id` The repository ID of another `InterfaceDef` object.

CORBA::AbstractInterfaceDef Interface

`AbstractInterfaceDef` describes an abstract IDL interface in the interface repository. It inherits from the `InterfaceDef` interface.

```
// IDL
interface AbstractInterfaceDef : InterfaceDef
{
};
```

CORBA::IRObject Interface

The interface `IRObject` is the base interface from which all interface repository interfaces are derived.

```
// IDL in module CORBA.

interface IRObject {
    readonly attribute DefinitionKind def_kind;
    void destroy();
};
```

IRObject::def_kind Attribute

```
// IDL
readonly attribute DefinitionKind def_kind;
```

Identifies the kind of an IFR object. For example, an `OperationDef` object, describing an IDL operation, has the kind `dk_Operation`.

See Also `CORBA::DefinitionKind`

IRObject::destroy()

```
// IDL
void destroy();
```

Deletes an IFR object. This also deletes any objects contained within the target object.

Exceptions `BAD_INV_ORDER` with a minor value of:

2	`destroy()` is invoked on a `Repository` or on a `PrimitiveDef` object.
1	An attempt is made to destroy an object that would leave the repository in an incoherent state.

CORBA::ModuleDef Interface

The interface `ModuleDef` describes an IDL module in the interface repository. It inherits from the interfaces `Container` and `Contained`.

```
// IDL in module CORBA.
interface ModuleDef : Container, Contained { };
```

The inherited operation `describe()` is also described.

ModuleDef::describe()

```
// IDL
Description describe();
```

Inherited from `Contained`, `describe()` returns a structure of type `Contained::Description`.

The `kind` member is `dk_Module`. The value member is an `any` whose `TypeCode` is `_tc_ModuleDescription` and whose value is a structure of type `ModuleDescription`.

See Also `CORBA::Contained::describe()`

CORBA::NamedValue Class

A NamedValue object describes an argument to a request or a return value, especially in the DII, and is used as an element of an NVList object. A NamedValue object maintains an any value, parameter-passing mode flags, and an (optional) name.

```
// IDL
pseudo interface NamedValue {
    readonly attribute Identifier name;
    readonly attribute any value;
    readonly attribute Flags flags;
};
// C++
class NamedValue {
public:
    const char *name() const;
    Any *value() const;
    Flags flags() const;

    static NamedValue_ptr _duplicate(NamedValue_ptr nv);
    static NamedValue_ptr _nil();
};
```

See Also

CORBA::NVList
CORBA::ORB::create_named_value()
CORBA::Request::result()
CORBA::Object::_create_request()

NamedValue::_duplicate()

```
static NamedValue_ptr _duplicate(NamedValue_ptr nv);
```

Returns a new reference to the NamedValue object input and increments its reference count.

Parameters

nv	The NamedValue object reference to be duplicated.

See Also CORBA::release()

NamedValue::flags()

```
Flags flags() const;
```

Returns the flags associated with the NamedValue. Flags identify the parameter passing mode for arguments of an NVList.

See Also CORBA::Flags

NamedValue::name()

```
const char *name() const;
```

Returns a pointer to the optional name associated with the NamedValue. This is the name of a parameter or argument of a request. The return value is a pointer to the internal memory of the NamedValue object and must not be freed by the caller.

NamedValue:: _nil()

```
static NamedValue_ptr _nil();
```

Returns a nil object reference for a NamedValue.

See Also CORBA::is_nil()

NamedValue::value()

```
Any *value() const;
```

Returns a pointer to Any value contained in the NamedValue.

The return value is a pointer to the internal memory of the NamedValue object and must not be freed by the caller. However, the value in a NamedValue may be manipulated via standard operations on any values.

CORBA::NativeDef Interface

The interface `NativeDef` describes an IDL native type in the interface repository. It inherits from the interface `TypedefDef`. The inherited type attribute is a `tk_native` `TypeCode` that describes the native type.

```
// IDL in module CORBA
interface NativeDef : TypedefDef {};
```

See Also `CORBA::Container::create_native()`

CORBA::NVList Class

An NVList is a pseudo-object used for constructing parameter lists. It is a list of NamedValue elements where each NamedValue describes an argument to a request.

The NamedValue and NVList types are used mostly in the DII in the request operations to describe arguments and return values. They are also used in the context object routines to pass lists of property names and values. The NVList is also used in the DSI operation ServerRequest::arguments().

The NVList class is partially opaque and may only be created by using ORB::create_list(). The NVList class is as follows:

```
// IDL
pseudo interface NVList {
    readonly attribute unsigned long count;
    NamedValue add(in Flags flags);
    NamedValue add_item(in Identifier item_name, in Flags flags);
    NamedValue add_value( in Identifier item_name,
        in any val, in Flags flags );
    NamedValue item(in unsigned long index) raises(Bounds);
    void remove(in unsigned long index) raises(Bounds);
};

// C++
class NVList {
public:
    ULong count() const;
    NamedValue_ptr add(Flags);
    NamedValue_ptr add_item(const char*, Flags);
    NamedValue_ptr add_value(const char*, const Any&, Flags);
    NamedValue_ptr add_item_consume(char*, Flags);
    NamedValue_ptr add_value_consume(char*, Any*, Flags);
    NamedValue_ptr item(ULong);
    void remove(ULong);

    static NVList_ptr _duplicate(NVList_ptr nv);
    static NVList_ptr _nil();
};
```

See Also

CORBA::NamedValue
CORBA::ORB:create_list()
CORBA::Object::_create_request()

NVList::count()

```
ULong count() const;
```

Returns the number of elements in the list.

NVList::add()

```
NamedValue_ptr add(
    Flags flags
);
```

Creates an unnamed value, initializes only the flags, and adds it to the list. The new NamedValue is returned.

Parameters

flags	Possible values include: ARG_IN ARG_OUT ARG_INOUT IN_COPY_VALUE DEPENDENT_LIST

The reference count of the returned NamedValue pseudo object is not incremented. Therefore, the caller should not release the returned reference when no longer needed, nor assign it to a *type*_var variable.

See Also

CORBA::NVList::add_item()
CORBA::NVList::add_value()
CORBA::NVList::add_item_consume()
CORBA::NVList::add_value_consume()

NVList::add_item()

```
NamedValue_ptr add_item(
    const char* item_name,
    Flags flags
);
```

Creates and returns a NamedValue with name and flags initialized, and adds it to the list.

Parameters

item_name	Name of item.
flags	Possible values include: ARG_IN ARG_OUT ARG_INOUT IN_COPY_VALUE DEPENDENT_LIST

The reference count of the returned NamedValue pseudo object is not incremented. Therefore, the caller should not release the returned reference when no longer needed, nor assign it to a *type*_var variable.

See Also

CORBA::NVList::add()
CORBA::NVList::add_value()
CORBA::NVList::add_item_consume()
CORBA::NVList::add_value_consume()

NVList::add_item_consume()

```
NamedValue_ptr add_item_consume(
    char* item_name,
    Flags flags
);
```

Creates and returns a NamedValue with name and flags initialised, and adds it to the list. The NVList takes over memory management responsibilities for the item_name parameter.

Parameters

`item_name`	Name of item. This parameter is consumed by the `NVList`. The caller may not access this data after it has been passed to this function.

`flags`	Possible values include:
	`ARG_IN`
	`ARG_OUT`
	`ARG_INOUT`
	`IN_COPY_VALUE`
	`DEPENDENT_LIST`

The reference count of the returned `NamedValue` pseudo object is not incremented. Therefore, the caller should not release the returned reference when no longer needed, nor assign it to a *type*_`var` variable.

See Also

`CORBA::NVList::add()`
`CORBA::NVList::add_item()`
`CORBA::NVList::add_value()`
`CORBA::NVList::add_value_consume()`

NVList::add_value()

```
NamedValue_ptr add_value(
    const char* item_name,
    const Any& value,
    Flags flags
);
```

Creates and returns a `NamedValue` with name, value, and flags initialized and adds it to the list.

Parameters

`item_name`	Name of item.

`value`	Value of item.

`flags`	Possible values include:
	`ARG_IN`
	`ARG_OUT`
	`ARG_INOUT`
	`IN_COPY_VALUE`
	`DEPENDENT_LIST`

The reference count of the returned `NamedValue` pseudo object is not incremented. Therefore, the caller should not release the returned reference when no longer needed, nor assign it to a *type*_var variable.

See Also

`CORBA::NVList::add()`
`CORBA::NVList::add_item()`
`CORBA::NVList::add_item_consume()`
`CORBA::NVList::add_value_consume()`

NVList::add_value_consume()

```
NamedValue_ptr add_value_consume(
    char* item_name,
    Any* value,
    Flags flags
);
```

Creates and returns a `NamedValue` with name, value, and flags initialised, and adds it to the list. The `NVList` takes over memory management responsibilities for both the name and value parameters.

Parameters

`item_name`	Name of item. This parameter is consumed by the `NVList`. The caller may not access this data after it has been passed to this function.

value	Value of item. This parameter is consumed by the NVList. The caller may not access this data after it has been passed to this function.
flags	Possible values include: ARG_IN ARG_OUT ARG_INOUT IN_COPY_VALUE DEPENDENT_LIST

The caller should use NamedValue::value() to modify the value attribute of the underlying NamedValue, if needed.

The reference count of the returned NamedValue pseudo object is not incremented. Therefore, the caller should not release the returned reference when no longer needed, nor assign it to a *type*_var variable.

See Also

CORBA::NamedValue::value()
CORBA::NVList::add()
CORBA::NVList::add_item()
CORBA::NVList::add_item_consume()
CORBA::NVList::add_value()

NVList::count()

```
ULong count() const;
```

Returns the number of NamedValue elements in the NVList.

NVList::_duplicate()

```
static NVList_ptr _duplicate(
    NVList_ptr nv
);
```

Returns a new reference to the NVList and increments the reference count of the nv object.

Parameters

nv	The NamedValue for which to get a duplicate reference.

See Also CORBA::release()

NVList::item()

```
NamedValue_ptr item(
    ULong index
);
```

Returns the NamedValue list item at the given index. The first item is at index 0. This method can be used to access existing elements in the list.

Parameters

index	Index of item.

Exceptions

Bounds	The index is out of range.

NVList::_nil()

```
static NVList_ptr _nil();
```

Returns a nil object reference for an NVList object.

See Also CORBA::is_nil()

NVList::remove()

```
void remove(
    ULong index
);
```

Removes the item at the given index. The first item is at index 0. The method calls CORBA::release() on the item.

Parameters

index	Index of item

Exceptions

`Bounds`	The index is out of range.

See Also `CORBA::release()`

CORBA::Object Class

The `Object` class is the base class for all normal CORBA objects. This class has some common methods that operate on any CORBA object. These operations are implemented directly by the ORB, not passed on to your object's implementation.

On the client side, the methods of this class are called on a proxy (unless collocation is set). On the server side, they are called on the real object.

Table 5 shows the methods provided by the `CORBA::Object` class:

Table 5: *Methods of the Object Class*

Manage Object References	Create Requests for the DII

`_duplicate()`
`_hash()`
`_is_a()`
`_is_equivalent()`
`_nil()`
`_non_existent()`
`_release()`
	`_create_request()`
	`_request()`

	Access Information in the IFR
	`_get_interface()`
Manage Policies and Domains	**Orbix Enhancements**

`_get_client_policy()`
`_get_domain_managers()`
`_get_policy()`
`_get_policy_overrides()`
`_set_policy_overrides()`
`_validate_connection()`
	`_it_get_orb()`
	`_it_proxy_for()`
	`_it_marshal()`
	`_it_get_type_id()`

The CORBA namespace provides the `is_nil()` and `release()` operations that are defined in the `Object` interface's IDL. All other IDL operations for the `Object` interface map to C++ functions with leading underscores.

```
// IDL
interface Object {
    boolean is_nil();
```

```
    Object duplicate();
    void release();
    ImplementationDef get_implementation();
    InterfaceDef get_interface();
    boolean is_a(in string logical_type_id);
    boolean non_existent();
    boolean is_equivalent(in Object other_object);
    unsigned long hash(in unsigned long maximum);
    void create_request(
        in Context ctx,
        in Identifier operation,
        in NVList arg_list,
        in NamedValue result,
        out Request request,
        in Flags req_flags
    );
    void create_request2(
        in Context ctx,
        in Identifier operation,
        in NVList arg_list,
        in NamedValue result,
        in ExceptionList exclist,
        in ContextList ctxtlist,
        out Request request,
        in Flags req_flags
    );
    Policy_ptr get_policy(in PolicyType policy_type);
    DomainManagerList get_domain_managers();
    Object set_policy_overrides(
        in PolicyList policies,
        in SetOverrideType set_or_add
    );

// IDL Additions from CORBA Messaging
    Policy get_policy(
        in PolicyType type
    );
    Policy get_client_policy(
        in PolicyType type
    );
    Object set_policy_overrides(
        in PolicyList policies,
        in SetOverrideType set_add
```

```
    )
        raises (InvalidPolicies);
    PolicyList get_policy_overrides(
        in PolicyTypeSeq types
    );
    boolean validate_connection(
        out PolicyList inconsistent_policies
    );
};
class Object {
public:
    static Object_ptr _duplicate(Object_ptr obj);
    static Object_ptr _nil();
    InterfaceDef_ptr _get_interface();
    Boolean _is_a(const char* logical_type_id);
    Boolean _non_existent();
    Boolean _is_equivalent(Object_ptr other_object);
    ULong _hash(ULong maximum);
    void _create_request(
        Context_ptr ctx,
        const char     *operation,
        NVList_ptr     arg_list,
        NamedValue_ptr result,
        Request_out    request,
        Flags          req_flags
    );
    void _create_request(
        Context_ptr       ctx,
        const char        *operation,
        NVList_ptr        arg_list,
        NamedValue_ptr    result,
        ExceptionList_ptr ,
        ContextList_ptr   ,
        Request_out       request,
        Flags             req_flags
    );
    Request_ptr _request(const char* operation);
    Policy_ptr _get_policy(PolicyType policy_type);
    DomainManagerList* _get_domain_managers();
    Object_ptr _set_policy_overrides(
        const PolicyList &policies,
        SetOverrideType set_add
    );
```

```
    virtual Policy_ptr _get_client_policy(
        PolicyType type
    ) = 0;
    virtual PolicyList * _get_policy_overrides(
        const PolicyTypeSeq & types
    ) = 0;
    virtual Boolean _validate_connection(
        PolicyList &inconsistent_policies
    ) = 0;

    //
    // Non-CORBA pseudo-operations.
    //

    virtual ORB_ptr _it_get_orb() = 0;

    virtual Object_ptr _it_proxy_for() = 0;

    virtual void _it_marshal(
        IT_OutStream_ptr os,
        ORB_ptr          orb
    ) = 0;

    virtual char* _it_get_type_id() = 0;
};
```

Object::_create_request()

```
void _create_request(
    Context_ptr ctx,
    const char *operation,
    NVList_ptr arg_list,
    NamedValue_ptr result,
    Request_out request,
    Flags req_flags
);

void _create_request(
    Context_ptr ctx,
    const char *operation,
    NVList_ptr arg_list,
```

```
    NamedValue_ptr result,
    ExceptionList_ptr exceptions,
    ContextList_ptr contexts,
    Request_out request,
    Flags req_flags
);
```

These construct a CORBA::Request object. These methods are part of the DII and create an ORB request on an object by constructing one of the object's operations.

See _request() for a simpler alternative way to create a Request.

Parameters

ctx	Context object, if any, to be sent in the request. If the ctx argument to _create_request() is a nil Context object reference, then you can add the Context later by calling the Request::ctx() function on the Request object.
operation	The name of the request operation. The operation name is the same operation identifier that is specified in the IDL definition for this operation.
arg_list	The parameters, for the operation, each of type NamedValue. If this value is zero, you can add the arguments later by calling the Request::arguments() function. You can also add each argument one at a time by calling the appropriate helper function such as add_in_arg() on the Request object.
result	The result of the operation invocation is placed in this argument after the invocation completes. Use ORB::create_named_value() to create the NamedValue object to be used as this return value parameter.
request	Contains the newly created Request.
req_flags	If you specify flag values they are ignored because argument insertion or extraction is handled using the Any type.

exceptions	A reference to a list of TypeCodes for all application-specific (user-defined) exceptions that may result when the Request is invoked.
contexts	A reference to a list of context strings for the operation.

The only implicit object reference operations allowed with the _create_request() call include:

_non_existent()
_is_a()
_get_interface()

Exceptions

BAD_PARAM	The name of an implicit operation that is not allowed is passed to _create_request()—for example, _is_equivalent is passed to _create_request() as the operation parameter.

See Also

CORBA::Object::_request()
CORBA::Request
CORBA::Request::arguments()
CORBA::Request::ctx()
CORBA::NVList
CORBA::NamedValue

Object::_duplicate()

```
static Object_ptr _duplicate(
    Object_ptr obj
);
```

Returns a new reference to obj and increments the reference count of the object. Because object references are opaque and ORB-dependent, it is not possible for your application to allocate storage for them. Therefore, if more than one copy of an object reference is needed, use this method to create a duplicate.

Parameters

obj	Pointer to the object to duplicate.

See Also

CORBA::release()

Object::_get_client_policy()

```
virtual Policy_ptr _get_client_policy(
    PolicyType type
) = 0;
```

Returns the effective overriding policy for the object reference. The effective override is obtained by first checking for an override of the given PolicyType at the Object scope, then at the Current scope, and finally at the ORB scope. If no override is present for the requested PolicyType, the system-dependent default value for that PolicyType is used.

Portable applications should set the desired defaults at the ORB scope since default policy values are not specified.

Parameters

type	The type of policy desired.

See Also

CORBA::Object::_get_policy()
CORBA::Object::_set_policy_overrides()
CORBA::Object::_get_policy_overrides()

Object::_get_domain_managers()

```
DomainManagersList* _get_domain_managers();
```

Returns the list of immediately enclosing domain managers of this object. At least one domain manager is always returned in the list since by default each object is associated with at least one domain manager at creation.

The _get_domain_managers() method allows applications such as administration services to retrieve the domain managers and hence the security and other policies applicable to individual objects that are members of the domain.

See Also

CORBA::DomainManager

Object::_get_interface()

```
InterfaceDef_ptr _get_interface();
```

Returns a reference to an object in the interface repository that describes this object's interface.

See Also CORBA::InterfaceDef

Object::_get_policy()

```
Policy_ptr _get_policy(
    PolicyType policy_type
);
```

Returns a reference to the Policy object of the type specified by the policy_type parameter.

Parameters

policy_type	The type of policy to get.

_get_policy() returns the effective policy which is the one that would be used if a request were made. Note that the effective policy may change from invocation to invocation due to transparent rebinding. Invoking _non_existent() on an object reference prior to _get_policy() ensures the accuracy of the returned effective policy.

Quality of Service (see "Quality of Service Framework") is managed on a per-object reference basis with _get_policy(), _set_policy_overrides(), _get_policy_overrides(), and _get_client_policy().

Exceptions

INV_POLICY	The value of policy_type is not valid either because the specified type is not supported by this ORB or because a policy object of that type is not associated with this object.

See Also

CORBA::Object::_non_existent()
CORBA::Object::_set_policy_overrides()
CORBA::Object::_get_policy_overrides()
CORBA::Object::_get_client_policy()
CORBA::Object::_validate_connection()

Object::_get_policy_overrides()

```
virtual PolicyList * _get_policy_overrides(
    const PolicyTypeSeq & types
) = 0;
```

Returns the list of policy overrides of the specified policy types set at the Object scope. If the specified sequence is empty, all policy overrides at this scope will be returned. If none of the requested policy types are overridden at the Object scope, an empty sequence is returned.

Parameters

types	A sequence of policy types for which the overrides are desired.

See Also

CORBA::Object::_get_policy()
CORBA::Object::_set_policy_overrides()
CORBA::Object::_get_client_policy()

Object::_hash()

```
ULong _hash(
    ULong maximum
);
```

Returns a hashed value for the object reference in the range 0...maximum.

Parameters

maximum	The maximum value that is to be returned from the hash method.

Use _hash() to quickly guarantee that objects references refer to different objects. For example, if _hash() returns the same hash number for two object references, the objects might or might not be the same, however, if the method returns different numbers for object references, these object references are guaranteed to be for different objects.

In order to efficiently manage large numbers of object references, some applications need to support a notion of object reference identity. Object references are associated with internal identifiers that you can access indirectly by using _hash(). The value of this internal identifier does not change during the lifetime of the object reference.

You can use _hash() and _is_equivalent() to support efficient maintenance and search of tables keyed by object references. _hash() allows you to partition the space of object references into sub-spaces of potentially equivalent object references. For example, setting maximum to 7 partitions the object reference space into a maximum of 8 sub-spaces (0 - 7).

See Also CORBA::Object::_is_equivalent()

Object::_is_a()

```
Boolean _is_a(
    const char* logical_type_id
);
```

Returns 1 (true) if the target object is either an instance of the type specified in logical_type_id or of a derived type of the type in logical_type_id. If the target object is neither, it returns 0 (false).

Parameters

logical_type_id	The fully scoped name of the IDL interface. This is a string denoting a shared type identifier (RepositoryId). Use an underscore ('_') rather than a scope operator (**::**) to delimit the scope.

The ORB maintains type-safety for object references over the scope of an ORB, but you can use this method to help maintaining type-safety when working in environments that do not have compile time type checking to explicitly maintain type safety.

Exceptions If _is_a() cannot make a reliable determination of type compatibility due to failure, it raises an exception in the calling application code. This enables the application to distinguish among the true, false, and indeterminate cases.

See Also CORBA::Object::_non_existent()

Object::_is_equivalent()

```
Boolean _is_equivalent(
    Object_ptr other_object
);
```

Returns 1 (true) if the object references definitely refer to the same object. A return value of 0 (false) does not necessarily mean that the object references are not equivalent, only that the ORB cannot confirm that they reference the same object. Two objects are equivalent if they have the same object reference, or they both refer to the same object.

Parameters

`other_object` An object reference of other object.

A typical application use of `_is_equivalent()` is to match object references in a hash table. Bridges could use the method to shorten the lengths of chains of proxy object references. Externalization services could use it to flatten graphs that represent cyclical relationships between objects.

See Also `CORBA::Object::_is_a()`
`CORBA::Object::_hash()`

Object::_it_get_orb()

```
virtual ORB_ptr _it_get_orb() = 0;
```

Returns the ORB.

Enhancement This is an Orbix enhancement.

Object::_it_get_type_id()

```
virtual char* _it_get_type_id() = 0;
```

Returns the repository ID string contained within the Interoperable Object Reference (IOR). If the IOR contains no type ID the return value is an empty string. This function follows the standard C++ mapping rules for string return values, which means the caller of this function must take responsibility for the returned string and ensure that it is freed via `CORBA::string_free()` when they are finished with it.

Enhancement This is an Orbix enhancement.

Object::_it_marshal()

```
virtual void _it_marshal(
    IT_OutStream_ptr os,
    ORB_ptr          orb
) = 0;
```

Enhancement This is an Orbix enhancement.

Object::_it_proxy_for()

```
virtual Object_ptr _it_proxy_for() = 0;
```

Returns a proxy for this object.

Enhancement This is an Orbix enhancement.

Object::_nil()

```
static Object_ptr _nil();
```

Returns a nil object reference.

See Also CORBA::is_nil()

Object::_non_existent()

```
Boolean _non_existent();
```

Returns 1 (true) if the object does not exist or returns 0 (false) otherwise.

Normally you might invoke this method on a proxy to determine whether the real object still exists. This method may be used to test whether an object has been destroyed because the method does not raise an exception if the object does not exist.

Applications that maintain state that includes object references, (such as bridges, event channels, and base relationship services) might use this method to sift through object tables for objects that no longer exist, deleting them as they go, as a form of garbage collection.

Object::_request()

```
Request_ptr _request(
    const char* operation
);
```

Returns a reference to a constructed Request on the target object. This is the simpler form of _create_request().

Parameters

operation	The name of the operation.

You can add arguments and contexts after construction using Request::arguments() and Request::ctx().

See Also

CORBA::Object::_create_request()
CORBA::Request::arguments()
CORBA::Request::ctx()

Object::_set_policy_overrides()

```
Object_ptr _set_policy_overrides(
    const PolicyList& policies,
    SetOverrideType   set_add
);
```

Returns a new object reference with the overriding policies associated with it.

Parameters

policies	A sequence of Policy object references that are to be associated with the new copy of the object reference returned.
set_add	Indicates whether the policies are in addition to (ADD_OVERRIDE) or as replacement of (SET_OVERRIDE) any existing overrides already associated with the object reference.

Exceptions

`NO_PERMISSION`	An attempt is made to override any policy that cannot be overridden. Only certain policies that pertain to the invocation of an operation at the client end can be overridden using this operation.

See Also

`CORBA::Object::_get_policy()`
`CORBA::Object::_get_policy_overrides()`
`CORBA::Object::_get_client_policy()`

Object::_validate_connection()

```
virtual Boolean _validate_connection(
    PolicyList &inconsistent_policies
) = 0;
```

Returns true if the current effective policies for the object will allow an invocation to be made. Returns false if the current effective policies would cause an invocation to raise the system exception `INV_POLICY`.

Parameters

`inconsistent_policies`	If the current effective policies are incompatible, This parameter contains those policies causing the incompatibility. This returned list of policies is not guaranteed to be exhaustive.

If the object reference is not yet bound, a binding will occur as part of this operation. If the object reference is already bound, but current policy overrides have changed or for any other reason the binding is no longer valid, a rebind will be attempted regardless of the setting of any `RebindPolicy` override. This method is the only way to force such a rebind when implicit rebinds are disallowed by the current effective `RebindPolicy`.

Exceptions The appropriate system exception is raised if the binding fails due to some reason unrelated to policy overrides.

CORBA::OperationDef Interface

Interface `OperationDef` describes an IDL operation that is defined in an IDL interface stored in the interface repository.

One way you can use the `OperationDef` is to construct an `NVList` for a specific operation for use in the Dynamic Invocation Interface. For details see `ORB::create_operation_list()`.

```
// IDL in module CORBA.
interface OperationDef : Contained {
    readonly attribute TypeCode result;
    attribute IDLType result_def;
    attribute ParDescriptionSeq params;
    attribute OperationMode mode;
    attribute ContextIdSeq contexts;
    attribute ExceptionDefSeq exceptions;
};
```

The inherited operation `describe()` is also described.

See Also

`CORBA::Contained`
`CORBA::ORB::create_operation_list()`
`CORBA::ExceptionDef`

OperationDef::contexts Attribute

```
// IDL
attribute ContextIdSeq contexts;
```

The list of context identifiers specified in the context clause of the operation.

OperationDef::exceptions Attribute

```
// IDL
attribute ExceptionDefSeq exceptions;
```

The list of exceptions that the operation can raise.

See Also CORBA::ExceptionDef

OperationDef::describe()

```
// IDL
Description describe();
```

Inherited from Contained, describe() returns a structure of type Contained::Description.

The DefinitionKind for the kind member of this structure is dk_Operation. The value member is an any whose TypeCode is _tc_OperationDescription and whose value is a structure of type OperationDescription.

See Also CORBA::Contained::describe()
CORBA::ExceptionDef

OperationDef::mode Attribute

```
// IDL
attribute OperationMode mode;
```

Specifies whether the operation is normal (OP_NORMAL) or oneway (OP_ONEWAY). The mode attribute can only be set to OP_ONEWAY if the result is _tc_void and all parameters have a mode of PARAM_IN.

OperationDef::params Attribute

```
// IDL
attribute ParDescriptionSeq params;
```

Specifies the parameters for this operation. It is a sequence of structures of type ParameterDescription.

The name member of the ParameterDescription structure provides the name for the parameter. The type member identifies the TypeCode for the parameter. The type_def member identifies the definition of the type for the parameter. The mode specifies whether the parameter is an in (PARAM_IN), an out (PARAM_OUT) or an inout (PARAM_INOUT) parameter. The order of the ParameterDescriptions is significant.

See Also

CORBA::TypeCode
CORBA::IDLType

OperationDef::result Attribute

```
// IDL
readonly attribute TypeCode result;
```

The return type of this operation. The attribute `result_def` contains the same information.

See Also

CORBA::TypeCode
CORBA::OperationDef::result_def

OperationDef::result_def Attribute

```
// IDL
attribute IDLType result_def;
```

Describes the return type for this operation. The attribute `result` contains the same information.

Setting the `result_def` attribute also updates the `result` attribute.

See Also

CORBA::IDLType
CORBA::OperationDef::result

CORBA::ORB Class

The ORB class provides a set of methods and data types that control the ORB from both the client and the server. See Table 6:

Table 6: *Methods and Types of the ORB Class*

Object Reference Manipulation	ORB Operation and Threads
_duplicate() list_initial_services() _nil() ObjectId type ObjectIdList sequence object_to_string() resolve_initial_references() string_to_object()	destroy() perform_work() run() shutdown() work_pending()
	ORB Policies and Services
	create_policy() get_service_information()
Dynamic Invocation Interface (DII)	**TypeCode Creation Methods**
create_environment() create_exception_list() create_list() create_named_value() create_operation_list() get_next_response() poll_next_response() RequestSeq sequence send_multiple_requests_deferred() send_multiple_requests_oneway()	create_abstract_interface_tc() create_alias_tc() create_array_tc() create_enum_tc() create_exception_tc() create_fixed_tc() create_interface_tc() create_native_tc() create_recursive_tc() create_sequence_tc() create_string_tc() create_struct_tc() create_union_tc() create_value_box_tc() create_value_tc() create_wstring_tc()
Value Type Factory Methods	
lookup_value_factory() register_value_factory() unregister_value_factory()	

You initialize the ORB using ORB_init().

The ORB class is defined as follows:

```
// IDL
pseudo interface ORB {
    typedef sequence<Request> RequestSeq;
    string object_to_string(in Object obj);
    Object string_to_object(in string str);
    void create_list(in long count, out NVList new_list);
    void create_operation_list(
        in OperationDef oper,
        out NVList
        new_list
    );
    void create_named_value(out NamedValue nmval);
    void create_exception_list(out ExceptionList exclist);
    void create_context_list(out ContextList ctxtlist);
    void get_default_context(out Context ctx);
    void create_environment(out Environment new_env);
    void send_multiple_requests_oneway(in RequestSeq req);
    void send_multiple_requests_deferred(in RequestSeq req);
    boolean poll_next_response();
    void get_next_response(out Request req);
    Boolean work_pending();
    void perform_work();
    void shutdown(in Boolean wait_for_completion);
    void run();
    void destroy();
    Boolean get_service_information (
        in ServiceType service_type,
        out ServiceInformation service_information
    );
    typedef string ObjectId;
    typedef sequence<ObjectId> ObjectIdList;
    Object resolve_initial_references(
        in ObjectId id
    ) raises(InvalidName);
    ObjectIdList list_initial_services();
    Policy create_policy(in PolicyType type, in any val)
```

```
    raises(PolicyError);
};

// C++
class ORB {
public:
    class RequestSeq {...};
    char *object_to_string(Object_var);
    Object_var string_to_object(const char *);
    void create_list(Long, NVList_out);
    void create_operation_list(OperationDef_ptr, NVList_out);
    void create_named_value(NamedValue_out);
    void create_exception_list(ExceptionList_out);
    void create_context_list(ContextList_out);
    void get_default_context(Context_out);
    void create_environment(Environment_out);
    void send_multiple_requests_oneway(const RequestSeq &);
    void send_multiple_requests_deferred(const RequestSeq &);
    Boolean poll_next_response();
    void get_next_response(Request_out);
    Boolean work_pending();
    void perform_work();
    void shutdown(Boolean wait_for_completion);
    void run();
    Boolean get_service_information(
        ServiceType svc_type,
        ServiceInformation_out svc_info
    );
    void destroy();
    typedef char* ObjectId;
    class ObjectIdList { ... };
    Object_ptr resolve_initial_references(const char* id);
    ObjectIdList* list_initial_services();
    Policy_ptr create_policy(PolicyType type, const Any& val);

    static ORB_ptr _duplicate(ORB_ptr orb);
    static ORB_ptr _nil();

    virtual TypeCode_ptr
    create_struct_tc(
        const char*  id,
        const char*  name,
        const StructMemberSeq &  members
```

```
) = 0;

virtual TypeCode_ptr
create_union_tc(
    const char*  id,
    const char*  name,
    TypeCode_ptr  discriminator_type,
    const UnionMemberSeq &  members
) = 0;

virtual TypeCode_ptr
create_enum_tc(
    const char*  id,
    const char*  name,
    const EnumMemberSeq &  members
) = 0;

virtual TypeCode_ptr
create_alias_tc(
    const char*  id,
    const char*  name,
    TypeCode_ptr  original_type
) = 0;

virtual TypeCode_ptr
create_exception_tc(
    const char*  id,
    const char*  name,
    const StructMemberSeq &  members
) = 0;

virtual TypeCode_ptr
create_interface_tc(
    const char*  id,
    const char*  name
) = 0;

virtual TypeCode_ptr
create_string_tc(
    CORBA::ULong bound
) = 0;

virtual TypeCode_ptr
```

```
create_wstring_tc(
    CORBA::ULong bound
) = 0;

virtual TypeCode_ptr
create_fixed_tc(
    CORBA::UShort digits,
    CORBA::Short scale
) = 0;

virtual TypeCode_ptr
create_sequence_tc(
    CORBA::ULong bound,
    TypeCode_ptr  element_type
) = 0;

virtual TypeCode_ptr
create_recursive_tc(
    const char*  id
) = 0;

virtual TypeCode_ptr
create_array_tc(
    CORBA::ULong length,
    TypeCode_ptr  element_type
) = 0;

virtual TypeCode_ptr
create_value_tc(
    const char*  id,
    const char*  name,
    ValueModifier type_modifier,
    TypeCode_ptr  concrete_base,
    const ValueMemberSeq &  members
) = 0;

virtual TypeCode_ptr
create_value_box_tc(
    const char*  id,
    const char*  name,
    TypeCode_ptr  original_type
) = 0;
```

```
    virtual TypeCode_ptr
    create_native_tc(
        const char*  id,
        const char*  name
    ) = 0;

    virtual TypeCode_ptr
    create_abstract_interface_tc(
        const char*  id,
        const char*  name
    ) = 0;


    virtual  ValueFactory
    register_value_factory(
        const char*  id,
        ValueFactory factory
    ) = 0;

    virtual void
    unregister_value_factory(
        const char*  id
    ) = 0;

    virtual  ValueFactory
    lookup_value_factory(
        const char*  id
    ) = 0;

};
```

ORB::create_abstract_interface_tc()

```
virtual TypeCode_ptr create_abstract_interface_tc(
    const char*  id,
    const char*  name
) = 0;
```

Returns a pointer to a new TypeCode of kind `tk_abstract_interface` representing an IDL abstract interface.

Parameters

`id`	The repository ID that globally identifies the TypeCode object.
`name`	The simple name identifying the TypeCode object within its enclosing scope.

See Also

CORBA::TypeCode
CORBA::TCKind

ORB::create_alias_tc()

```
virtual TypeCode_ptr create_alias_tc(
    const char*  id,
    const char*  name,
    TypeCode_ptr  original_type
) = 0;
```

Returns a pointer to a new TypeCode of kind `tk_alias` representing an IDL alias.

Parameters

`id`	The repository ID that globally identifies the TypeCode object.
`name`	The simple name identifying the TypeCode object within its enclosing scope.
`original_type`	A pointer to the actual TypeCode object this alias represents.

See Also

CORBA::TypeCode
CORBA::TCKind

ORB::create_array_tc()

```
virtual TypeCode_ptr create_array_tc(
    CORBA::ULong length,
    TypeCode_ptr  element_type
) = 0;
```

Returns a pointer to a new TypeCode of kind `tk_array` representing an IDL array.

Parameters

`length`	The length of the array.
`element_type`	The data type for the elements of the array.

See Also CORBA::TypeCode
CORBA::TCKind

ORB::create_context_list()

```
void create_context_list(ContextList_out list);
```

Creates an empty `ContextList` object for use with a DII request. You can add context strings to the list using `ContextList::add()` and then pass the list as a parameter to `Object::_create_request()`.

Parameters

`list`	A reference to the new `ContextList`.

See Also CORBA::ContextList
CORBA::Object::_create_request()

ORB::create_enum_tc()

```
virtual TypeCode_ptr create_enum_tc(
    const char*  id,
    const char*  name,
    const EnumMemberSeq &  members
) = 0;
```

Returns a pointer to a new `TypeCode` of kind `tk_enum` representing an IDL enumeration.

Parameters

`id`	The repository ID that globally identifies the `TypeCode` object.
`name`	The simple name identifying the `TypeCode` object within its enclosing scope.
`members`	The sequence of enumeration members.

See Also CORBA::TypeCode

CORBA::TCKind

ORB::create_environment()

```
void create_environment(
    Environment_out environment
);
```

Gets a newly created Environment object.

Parameters

new_env	New environment created.

See Also CORBA::Environment

ORB::create_exception_list()

```
void create_exception_list(
    ExceptionList_out list
);
```

Creates an empty ExceptionList object for use with a DII request. You can add user-defined exceptions to the list using ExceptionList::add() and then pass the list as a parameter to Object::_create_request().

Parameters

list	A reference to the new ExceptionList.

See Also
CORBA::ExceptionList
CORBA::Object::_create_request()

ORB::create_exception_tc()

```
virtual TypeCode_ptr create_exception_tc(
    const char*  id,
    const char*  name,
    const StructMemberSeq &  members
) = 0;
```

Returns a pointer to a new TypeCode of kind tk_except representing an IDL exception.

Parameters

id	The repository ID that globally identifies the TypeCode object.
name	The simple name identifying the TypeCode object within its enclosing scope.
members	The sequence of members.

See Also CORBA::TypeCode
CORBA::TCKind

ORB::create_fixed_tc()

```
virtual TypeCode_ptr create_fixed_tc(
    CORBA::UShort digits,
    CORBA::Short scale
) = 0;
```

Returns a pointer to a new TypeCode of kind tk_fixed representing an IDL fixed point type.

Parameters

digits	The number of digits for the fixed point type.
scale	The scale of the fixed point type.

See Also CORBA::TypeCode
CORBA::TCKind

ORB::create_interface_tc()

```
virtual TypeCode_ptr create_interface_tc(
    const char*  id,
    const char*  name
) = 0;
```

Returns a pointer to a new TypeCode representing an IDL interface.

Parameters

`id`	The repository ID that globally identifies the `TypeCode` object.
`name`	The simple name identifying the `TypeCode` object within its enclosing scope.

See Also

`CORBA::TypeCode`
`CORBA::TCKind`

ORB::create_list()

```
void create_list(
    Long count,
    NVList_out list
);
```

Allocates space for an empty `NVList` of the size specified by `count` to contain `NamedValue` objects. A list of `NamedValue` object can be used to describe arguments to a request when using the Dynamic Invocation Interface. You can add `NamedValue` items to list using the `NVList::add_item()` routine.

Parameters

`count`	Number of elements anticipated for the new `NVList`. This is a hint to help with storage allocation.
`list`	A pointer to the start of the list. The caller must release the reference when it is no longer needed, or assign it to an `NVList_var` variable for automatic management.

See Also

`CORBA::NVList`
`CORBA::NamedValue`
`CORBA::ORB::create_operation_list()`
`CORBA::Request()`

ORB::create_named_value()

```
void create_named_value(
    NamedValue_out value
);
```

Creates NamedValue objects you can use as return value parameters in the Object::_create_request() method.

Parameters

value	A pointer to the NamedValue object created. You must release the reference when it is no longer needed, or assign it to a NamedValue_var variable for automatic management.

See Also

CORBA::NVList
CORBA::NamedValue
CORBA::Any
CORBA::ORB::create_list()

ORB::create_native_tc()

```
virtual TypeCode_ptr create_native_tc(
    const char*  id,
    const char*  name
) = 0;
```

Returns a pointer to a new TypeCode of kind tk_native representing an IDL native type.

Parameters

id	The repository ID that globally identifies the TypeCode object.
name	The simple name identifying the TypeCode object within its enclosing scope.

See Also

CORBA::TypeCode
CORBA::TCKind

ORB::create_operation_list()

```
void create_operation_list(
    OperationDef_ptr operation,
    NVList_out list
);
```

Creates an NVList and returns it in the list parameter, initialized with the argument descriptions for the operation specified in operation.

Parameters

`operation`	A pointer to the interface repository object describing the operation.
`list`	A pointer to the start of the list. The caller must release the reference when it is no longer needed, or assign it to a `NVList_var` variable for automatic management. The returned `NVList` is of the correct length with one element per argument, and each `NamedValue` element of the list has a valid name and valid flags (denoting the argument passing mode).

Each element in the list is of type `NamedValue` whose `value` member (of type `CORBA::Any`) has a valid type that denotes the type of the argument. The value of the argument is not filled in.

Use of this method requires that the relevant IDL file be compiled with the `-R` option.

See Also

`CORBA::NVList`
`CORBA::NamedValue`
`CORBA::Any`
`CORBA::ORB::create_list()`

ORB::create_policy()

```
Policy_ptr create_policy(
    PolicyType type,
    const Any& value
);
```

Returns a reference to a newly created `Policy` object.

Parameters

`type`	The `PolicyType` of the `Policy` object to be created.
`value`	The value for the initial state of the `Policy` object created.

Exceptions

PolicyError	The requested policy type or initial state for the policy is not supported. The appropriate reason as described in the PolicyErrorCode.

See Also

CORBA::Policy
CORBA::PolicyType
CORBA::PolicyErrorCode

ORB::create_recursive_tc()

```
virtual TypeCode_ptr create_recursive_tc(
    const char*  id
) = 0;
```

Returns a pointer to a recursive TypeCode, which serves as a place holder for a concrete TypeCode during the process of creating type codes that contain recursion. After the recursive TypeCode has been properly embedded in the enclosing TypeCode, which corresponds to the specified repository id, it will act as a normal TypeCode.

Parameters

id	The repository ID of the enclosing type for which the recursive TypeCode is serving as a place holder.

Invoking operations on the recursive TypeCode before it has been embedded in the enclosing TypeCode will result in undefined behavior.

Examples

The following IDL type declarations contains TypeCode recursion:

```
// IDL
struct foo {
    long value;
    sequence<foo> chain;
};

valuetype V {
public V member;
};
```

To create a TypeCode for valuetype V, you invoke the TypeCode creation functions as follows:

```
// C++
TypeCode_var recursive_tc = orb->create_recursive_tc("IDL:V:1.0");
ValueMemberSeq v_seq;
v_seq.length(1);
v_seq[0].name = string_dup(ìmemberî);
v_seq[0].type = recursive_tc;
v_seq[0].access = PUBLIC_MEMBER;
TypeCode_var v_val_tc = orb->create_value_tc(
    "IDL:V:1.0",
    "V",
    VM_NONE,
    TypeCode::_nil(),
    v_seq
);
```

See Also CORBA::TypeCode

ORB::create_sequence_tc()

```
virtual TypeCode_ptr create_sequence_tc(
    CORBA::ULong bound,
    TypeCode_ptr  element_type
) = 0;
```

Returns a pointer to a new TypeCode of kind `tk_sequence` representing an IDL sequence.

Parameters

bound	The upper bound of the sequence.
element_type	The data type for the elements of the sequence.

See Also CORBA::TypeCode
CORBA::TCKind

ORB::create_string_tc()

```
virtual TypeCode_ptr create_string_tc(
    CORBA::ULong bound
) = 0;
```

Returns a pointer to a new TypeCode of kind tk_string representing an IDL string.

Parameters

bound	The upper bound of the string.

See Also CORBA::TypeCode
CORBA::TCKind

ORB::create_struct_tc()

```
virtual TypeCode_ptr create_struct_tc(
    const char*  id,
    const char*  name,
    const StructMemberSeq &  members
) = 0;
```

Returns a pointer to a new TypeCode of kind tk_struct representing an IDL structure.

Parameters

id	The repository ID that globally identifies the TypeCode object.
name	The simple name identifying the TypeCode object within its enclosing scope.
members	The sequence of structure members.

See Also CORBA::TypeCode
CORBA::TCKind

ORB::create_union_tc()

```
virtual TypeCode_ptr create_union_tc(
    const char*  id,
    const char*  name,
    TypeCode_ptr  discriminator_type,
    const UnionMemberSeq &  members
) = 0;
```

Returns a pointer to a TypeCode of kind tk_union representing an IDL union.

Parameters

`id`	The repository ID that globally identifies the TypeCode object.
`name`	The simple name identifying the TypeCode object within its enclosing scope.
`discriminator_type`	The union discriminator type.
`members`	The sequence of union members.

See Also CORBA::TypeCode
CORBA::TCKind

ORB::create_value_box_tc()

```
virtual TypeCode_ptr create_value_box_tc(
    const char*  id,
    const char*  name,
    TypeCode_ptr  original_type
) = 0;
```

Returns a pointer to a new TypeCode of kind `tk_value_box` representing an IDL boxed value.

Parameters

`id`	The repository ID that globally identifies the TypeCode object.
`name`	The simple name identifying the TypeCode object within its enclosing scope.
`original_type`	A pointer to the original TypeCode object this boxed value represents.

See Also CORBA::TypeCode
CORBA::TCKind

ORB::create_value_tc()

```
virtual TypeCode_ptr create_value_tc(
    const char*  id,
    const char*  name,
```

```
    ValueModifier type_modifier,
    TypeCode_ptr  concrete_base,
    const ValueMemberSeq &  members
) = 0;
```

Returns a pointer to a TypeCode of kind `tk_value` representing an IDL value type.

Parameters

`id`	The repository ID that globally identifies the TypeCode object.
`name`	The simple name identifying the TypeCode object within its enclosing scope.
`type_modifier`	A value type modifier.
`concrete_base`	A TypeCode for the immediate concrete value type base of the value type for which the TypeCode is being created. If the value type does not have a concrete base, use a nil TypeCode reference.
`members`	The sequence of value type members.

See Also CORBA::TypeCode
CORBA::TCKind

ORB::create_wstring_tc()

```
virtual TypeCode_ptr create_wstring_tc(
    CORBA::ULong bound
) = 0;
```

Returns a pointer to a new TypeCode of kind `tk_wstring` representing an IDL wide string.

Parameters

bound	The upper bound of the string.

See Also CORBA::TypeCode
CORBA::TCKind

ORB::destroy()

```
void destroy();
```

This thread operation destroys the ORB so that its resources can be reclaimed by the application.

If `destroy()` is called on an ORB that has not been shut down (see `shutdown()`) it will start the shut down process and block until the ORB has shut down before it destroys the ORB. For maximum portability and to avoid resource leaks, applications should always call `shutdown()` and `destroy()` on all ORB instances before exiting.

After an ORB is destroyed, another call to `ORB_init()` with the same ORB ID will return a reference to a newly constructed ORB.

Exceptions

`BAD_INV_ORDER`, minor code `3`	An application calls `destroy()` in a thread that is currently servicing an invocation because blocking would result in a deadlock.
`OBJECT_NOT_EXIST`	An operation is invoked on a destroyed ORB reference.

The exception is raise if

See Also

`CORBA::ORB::run()`
`CORBA::ORB::shutdown()`
`CORBA::ORB_init()`

ORB::_duplicate()

```
static ORB_ptr _duplicate(
    ORB_ptr obj
);
```

Returns a new reference to `obj` and increments the reference count of the object. Because object references are opaque and ORB-dependent, it is not possible for your application to allocate storage for them. Therefore, if more than one copy of an object reference is needed, use this method to create a duplicate.

Parameters

`obj`	Pointer to the object to duplicate.

See Also `CORBA::release()`

ORB::get_default_context()

```
void get_default_context(Context_out context);
```

Obtains a `CORBA::Context` object representing the default context of the process.

Parameters

`context`	The default context of the process.

See Also `CORBA::Context`
`CORBA::NVList`

ORB::get_next_response()

```
void get_next_response(
    Request_out request
);
```

Gets the next response for a request that has been sent.

Parameters

`request`	A pointer to the `Request` whose completion is being reported.

You can call `get_next_response()` successively to determine the outcomes of the individual requests from `send_multiple_requests_deferred()` calls. The order in which responses are returned is not necessarily related to the order in which the requests are completed.

Exceptions

`WrongTransaction`	The thread invoking this method has a non-null transaction context that differs from that of the request *and* the request has an associated transaction context.

See Also `CORBA::ORB::send_multiple_requests_deferred()`

CORBA::Request::get_response()
CORBA::Request::send_deferred()
CORBA::ORB::poll_next_response()

ORB::get_service_information()

```
Boolean get_service_information(
    ServiceType svc_type,
    ServiceInformation_out svc_info
);
```

Gets the service information about CORBA facilities and services that this ORB supports. Returns 1 (true) if service information is available for the svc_type and returns 0 (false) otherwise.

Parameters

svc_type	The service type for which information is being requested.
svc_info	The service information available for svc_type, if that information is available.

See Also CORBA::ServiceInformation

ORB::list_initial_services()

```
ObjectIdList* list_initial_services();
```

Returns a sequence of ObjectId strings, each of which names a service provided by Orbix. This method allows your application to determine which objects have references available. Before you can use some services such as the naming service in your application you have to first obtain an object reference to the service.

The ObjectIdList may include the following names:

```
DynAnyFactory
IT_Configuration
InterfaceRepository
NameService
ORBPolicyManager
POACurrent
PSS
```

```
RootPOA
SecurityCurrent
TradingService
TransactionCurrent
```

See Also CORBA::ORB::resolve_initial_references()

ORB::lookup_value_factory()

```
virtual  ValueFactory lookup_value_factory(
    const char*  id
) = 0;
```

Returns a pointer to the factory method.

Parameters

`id` A repository ID that identifies a value type factory method.

Your application assumes ownership of the returned reference to the factory. When you are done with the factory, invoke ValueFactoryBase::_remove_ref() once on that factory.

See Also CORBA::ValueFactory
CORBA::ORB::register_value_factory()
CORBA::ORB::unregister_value_factory()

Object::_nil()

```
static ORB_ptr _nil();
```

Returns a nil object reference.

See Also CORBA::is_nil()

ORB::ObjectId

```
typedef char* ObjectId;
```

The name that identifies an object for a service. `ObjectId` strings uniquely identify each service used by an ORB.

See Also CORBA::ORB::ObjectIdList

CORBA::ORB::list_initial_services()

ORB::ObjectIdList Sequence Class

```
class ObjectIdList {
public:
    // default constructor
    ObjectIdList();
    // initial maximum length constructor
    ObjectIdList(ULong max);
    // data constructor
    ObjectIdList(
        ULong max,
        ULong length,
        ObjectId *data,
        Boolean release = FALSE
    );
    // copy constructor
    ObjectIdList(const ObjectIdList&);

     // destructor
    ~ObjectIdList();

    // assignment operator
    ObjectIdList &operator=(const ObjectIdList&);

    ULong maximum() const;
    void length(ULong);
    ULong length() const;

    // subscript operators
    ObjectId &operator[](ULong index);
    const ObjectId &operator[](ULong index) const;

    Boolean release() const;
    void replace(
        ULong max,
        ULong length,
        ObjectId *data,
        Boolean release = FALSE
    );
```

```
    // buffer reference
    ObjectId* get_buffer(Boolean orphan = FALSE);
    // buffer access
    const ObjectId* get_buffer() const;
};
```

A sequence of ObjectId objects.

See Also

CORBA::ORB::ObjectId
CORBA::ORB::list_initial_services()

"About Sequences"

ORB::object_to_string()

```
char *object_to_string(
    Object_var obj
);
```

Returns a string representation of an object reference. An object reference can be translated into a string by this method and the resulting value stored or communicated in whatever ways strings are manipulated.

Parameters

obj	Object reference to be translated to a string.

Use string_to_object() to translate the string back to the corresponding object reference.

A string representation of an object reference has the prefix IOR: followed by a series of hexadecimal octets. The hexadecimal strings are generated by first turning an object reference into an *interoperable object reference* (IOR), and then encapsulating the IOR using the encoding rules of *common data representation* (CDR). The content of the encapsulated IOR is then turned into hexadecimal digit pairs, starting with the first octet in the encapsulation and going until the end. The high four bits of each octet are encoded as a hexadecimal digit, then the low four bits are encoded.

Note: Because an object reference is opaque and may differ from ORB to ORB, the object reference itself is not a convenient value for storing references to objects in persistent storage or communicating references by means other than invocation.

See Also CORBA::ORB::string_to_object()

ORB::perform_work()

```
void perform_work();
```

A thread function that provides execution resources to your application if called by the main thread. This function does nothing if called by any other thread.

You can use perform_work() and work_pending() for a simple polling loop that multiplexes the main thread among the ORB and other activities. Such a loop would most likely be used in a single-threaded server. A multi-threaded server would need a polling loop only if there were both ORB and other code that required use of the main thread. Here is a simple example of such a polling loop:

```
// C++
for (;;) {
    if (orb->work_pending()) {
        orb->perform_work();
    };
    // do other things
    // sleep
};
```

Exceptions

BAD_INV_ORDER, minor code 4	The method is called after the ORB has shut down. You can catch this exception to determine when to terminate a polling loop.

See Also CORBA::ORB::run()
CORBA::ORB::work_pending()

ORB::poll_next_response()

```
Boolean poll_next_response();
```

Returns 1 (true) if any request has completed or returns 0 (false) if none have completed. This method returns immediately, whether any request has completed or not.

You can call this method successively to determine whether the individual requests specified in a `send_multiple_requests_oneway()` or `send_multiple_requests_deferred()` call have completed successfully.

Alternatively you can call `Request::poll_response()` on the individual `Request` objects in the sequence of requests passed to `send_multiple_requests_oneway()` or `send_multiple_requests_deferred()`.

See Also

`CORBA::ORB::get_next_response()`
`CORBA::ORB::send_multiple_requests_oneway()`
`CORBA::ORB::send_multiple_requests_deferred()`
`CORBA::Request::poll_response()`

ORB::register_value_factory()

```
virtual  ValueFactory register_value_factory(
    const char*  id,
    ValueFactory factory
) = 0;
```

Registers a value type factory method with the ORB for a particular value type. The method returns a null pointer if no previous factory was registered for the type. If a factory is already registered for the value type, the method replaces the factory and returns a pointer to the previous factory for which the caller assumes ownership.

Parameters

`id`	A repository ID that identifies the factory.
`factory`	The application-specific factory method that the ORB calls whenever it needs to create the value type during the unmarshaling of value instances.

When a value type factory is registered with the ORB, the ORB invokes ValueFactoryBase::_add_ref() once on the factory before returning from register_value_factory(). When the ORB is done using that factory, the reference count is decremented once with ValueFactoryBase::_remove_ref(). This can occur in any of the following circumstances:

- If the factory is explicitly unregistered via unregister_value_factory(), the ORB invokes ValueFactoryBase::_remove_ref() once on the factory.
- If the factory is implicitly unregistered due to a call to shutdown(), the ORB invokes ValueFactoryBase::_remove_ref() once on each registered factory.
- If you replace a factory by calling this register_value_factory() again, you should invoke ValueFactoryBase::_remove_ref() once on the previous factory.

See Also

CORBA::ValueFactory
CORBA::ORB::lookup_value_factory()
CORBA::ORB::unregister_value_factory()

ORB::RequestSeq Sequence

```
class RequestSeq {
public:
    // default constructor
    RequestSeq();
    // initial maximum length constructor
    RequestSeq(ULong max);
    // data constructor
    RequestSeq(
        ULong max,
        ULong length,
        Request *data,
        Boolean release = FALSE
    );
    // copy constructor
    RequestSeq(const RequestSeq&);

     // destructor
    ~RequestSeq();
```

```
    // assignment operator
    RequestSeq &operator=(const RequestSeq&);

    ULong maximum() const;
    void length(ULong);
    ULong length() const;

    // subscript operators
    Request &operator[](ULong index);
    const Request &operator[](ULong index) const;

    Boolean release() const;
    void replace(
        ULong max,
        ULong length,
        Request *data,
        Boolean release = FALSE
    );

    // buffer reference
    Request* get_buffer(Boolean orphan = FALSE);
    // buffer access
    const Request* get_buffer() const;
};
```

A sequence of Request objects.

See Also

CORBA::Request
CORBA::ORB::send_multiple_requests_oneway()
CORBA::ORB::send_multiple_requests_deferred()

"About Sequences"

ORB::resolve_initial_references()

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

Returns an object reference for a desired service.

Parameters

`id`	The name of the desired service. Use `list_initial_services()` to obtain the list of services supported.

Applications require a portable means by which to obtain some initial object references such as the root POA, the interface repository, and various object services instances. The functionality of `resolve_initial_references()` and `list_initial_services()` is like a simplified, local version of the naming service that has only a small set of objects in a flattened single-level name space.

The object reference returned must be narrowed to the correct object type. For example, the object reference returned from resolving the `id` name `InterfaceRepository` must be narrowed to the type `CORBA::Repository`.

See Also `CORBA::ORB::list_initial_services()`

ORB::run()

```
void run();
```

A thread method that enables the ORB to perform work using the main thread. If called by any thread other than the main thread, this method simply waits until the ORB has shut down.

This method provides execution resources to the ORB so that it can perform its internal functions. Single threaded ORB implementations, and some multi-threaded ORB implementations need to use the main thread. For maximum portability, your applications should call either `run()` or `perform_work()` on the main thread.

`run()` returns after the ORB has completed the shutdown process, initiated when some thread calls `shutdown()`.

See Also

`CORBA::ORB::perform_work()`
`CORBA::ORB::work_pending()`
`CORBA::ORB::shutdown()`
`CORBA::ORB::destroy()`

"Threading and Synchronization Toolkit Overview"

ORB::send_multiple_requests_deferred()

```
void send_multiple_requests_deferred(
    const RequestSeq &req
);
```

Initiates a number of requests in parallel.

Parameters

`req`	A sequence of requests.

The method does not wait for the requests to finish before returning to the caller. The caller can use `get_next_response()` or `Request::get_response()` to determine the outcome of the requests. Memory leakage will result if one of these methods is not called for a request issued with `send_multiple_requests_oneway()` or `Request::send_deferred()`.

See Also

`CORBA::ORB::send_multiple_requests_oneway()`
`CORBA::Request::get_response()`
`CORBA::Request::send_deferred()`
`CORBA::ORB::get_next_response()`

ORB::send_multiple_requests_oneway()

```
void send_multiple_requests_oneway(
    const RequestSeq &req
);
```

Initiates a number of requests in parallel. It does not wait for the requests to finish before returning to the caller.

Parameters

`req`	A sequence of requests. The operations in this sequence do not have to be IDL oneway operations. The caller does not expect a response, nor does it expect out or inout parameters to be updated.

See Also

`CORBA::Request::send_oneway()`
`CORBA::ORB::send_multiple_requests_deferred()`

ORB::shutdown()

```
void shutdown(
    Boolean wait_for_completion
);
```

This thread method instructs the ORB to shut down in preparation for ORB destruction.

Parameters

`wait_for_completion`	Designates whether or not to wait for completion before continuing. If the value is 1 (true), this method blocks until all ORB processing has completed, including request processing and object deactivation or other methods associated with object adapters. If the value is 0 (false), then shut down may not have completed upon return of the method.

While the ORB is in the process of shutting down, the ORB operates as normal, servicing incoming and outgoing requests until all requests have been completed. Shutting down the ORB causes all object adapters to be shut down because they cannot exist without an ORB.

Once an ORB has shutdown, you can invoke only object reference management methods including `CORBA::_duplicate()`, `release()`, and `is_nil()` on the ORB or any object reference obtained from the ORB. An application may also invoke `ORB::destroy()` on the ORB itself. Invoking any other method raises exception `BAD_INV_ORDER` system with the OMG minor code 4.

Exceptions

`BAD_INV_ORDER`, minor code 3	An application calls this method in a thread that is currently servicing an invocation because blocking would result in a deadlock.

See Also

`CORBA::ORB::run()`
`CORBA::ORB::destroy()`

ORB::string_to_object()

```
Object_var string_to_object(
    const char *obj_ref_string
);
```

Returns an object reference by converting a string representation of an object reference.

Parameters

`obj_ref_string` String representation of an object reference to be converted.

To guarantee that an ORB will understand the string form of an object reference, the string must have been produced by a call to `object_to_string()`.

See Also `CORBA::ORB::object_to_string()`

ORB::unregister_value_factory()

```
virtual void unregister_value_factory(
    const char*  id
) = 0;
```

Unregisters a value type factory method from the ORB.

Parameters

`id` A repository ID that identifies a value type factory method.

See Also
`CORBA::ValueFactory`
`CORBA::ORB::lookup_value_factory()`
`CORBA::ORB::register_value_factory()`

ORB::work_pending()

```
Boolean work_pending();
```

This thread method returns an indication of whether the ORB needs the main thread to perform some work. A return value of 1 (true) indicates that the ORB needs the main thread to perform some work and a return value of 0 (false) indicates that the ORB does not need the main thread.

Exceptions

`BAD_INV_ORDER`, The method is called after the ORB has shutdown. minor code 4

See Also

`CORBA::ORB::run()`
`CORBA::ORB::perform_work()`

CORBA::Policy Interface

An ORB or CORBA service may choose to allow access to certain choices that affect its operation. This information is accessed in a structured manner using interfaces derived from the `Policy` interface defined in the CORBA module. A CORBA service is not required to use this method of accessing operating options, but may choose to do so.

This chapter is divided into the following sections:

- "Quality of Service Framework"
- "Policy Methods"

The following policies are available. These are classes that inherit from the `CORBA::Policy` class:

Table 7: *Policies*

Category	Policy
CORBA and IT_CORBA	`CORBA::ConstructionPolicy` `IT_CORBA::WellKnownAddressingPolicy`

PortableServer and IT_PortableServer	`PortableServer::ThreadPolicy`
	`PortableServer::LifespanPolicy`
	`PortableServer::IdUniquenessPolicy`
	`PortableServer::IdAssignmentPolicy`
	`PortableServer::ImplicitActivationPolicy`
	`PortableServer::ServantRetentionPolicy`
	`PortableServer::RequestProcessingPolicy`
	`IT_PortableServer::ObjectDeactivationPolicy`
	`IT_PortableServer::PersistenceModePolicy`

Messaging	`RebindPolicy`
	`SyncScopePolicy`
	`RoutingPolicy`

You create instances of a policy by calling `CORBA::ORB::create_policy()`.

Quality of Service Framework

A `Policy` is the key component for a standard *Quality of Service framework* (*QoS*). In this framework, all qualities are defined as interfaces derived from `CORBA::Policy`. This framework is how all service-specific qualities are defined. The components of the framework include:

`Policy`	This base interface from which all QoS objects derive.
`PolicyList`	A sequence of `Policy` objects.
`PolicyManager`	An interface with operations for querying and overriding QoS policy settings.
Policy Transport Mechanisms	Mechanisms for transporting policy values as part of interoperable object references and within requests. These include: • `TAG_POLICIES` - A Profile Component containing the sequence of QoS policies exported with the object reference by an object adapter. • `INVOCATION_POLICIES` - A Service Context containing a sequence of QoS policies in effect for the invocation.

Most policies are appropriate only for management at either the server or client, but not both. Server-side policies are associated with a POA. Client-side policies are divided into ORB-level, thread-level, and object-level policies. At the thread and ORB levels, use the `PolicyManager` interface to query the current set of policies and override these settings.

POA Policies for Servers

Server-side policy management is handled by associating QoS `Policy` objects with a POA. Since all QoS are derived from interface `Policy`, those that are applicable to server-side behavior can be passed as arguments to `POA::create_POA()`. Any such policies that affect the behavior of requests (and therefore must be accessible by the ORB at the client side) are exported within the object references that the POA creates. It is clearly noted in a POA policy definition when that policy is of interest to the client. For those policies

that can be exported within an object reference, the absence of a value for that policy type implies that the target supports any legal value of that PolicyType.

ORB-level Policies for Clients

You obtained the ORB's locality-constrained PolicyManager through an invocation of CORBA::ORB::resolve_initial_references(), specifying an identifier of ORBPolicyManager. This PolicyManager has operations through which a set of policies can be applied and the current overriding policy settings can be obtained. Policies applied at the ORB level override any system defaults.

Thread-level Policies for Clients

You obtained a thread's locality-constrained PolicyCurrent through an invocation of CORBA::ORB::resolve_initial_references(), specifying an identifier of PolicyCurrent. Policies applied at the thread-level override any system defaults or values set at the ORB level. When accessed from a newly spawned thread, the PolicyCurrent initially has no overridden policies. The PolicyCurrent also has no overridden values when a POA with ThreadPolicy of ORB_CONTROL_MODEL dispatches an invocation to a servant. Each time an invocation is dispatched through a SINGLE_THREAD_MODEL POA, the thread-level overrides are reset to have no overridden values.

Object-level Policies for Clients

Operations are defined on the base Object interface through which a set of policies can be applied. Policies applied at the object level override any system defaults or values set at the ORB or thread levels. In addition, accessors are defined for querying the current overriding policies set at the object level, and for obtaining the current effective client-side policy of a given PolicyType. The effective client-side policy is the value of a PolicyType that would be in effect if a request were made. This is determined by checking for overrides at the object level, then at the thread level, and finally at the ORB level. If no overriding policies are set at any

level, the system-dependent default value is returned. Portable applications are expected to override the ORB-level policies since default values are not specified in most cases.

Policy Methods

The Policy interface is as follows:

```
// IDL in module CORBA
interface Policy {
    readonly attribute PolicyType policy_type;
    Policy copy();
    void destroy();
};
```

Policy::policy_type Attribute

```
// IDL
readonly attribute PolicyType policy_type;
```

This read-only attribute returns the constant value of type PolicyType that corresponds to the type of the Policy object.

Policy::copy()

```
// IDL
Policy copy();
```

This operation copies the Policy object. The copy does not retain any relationships that the original policy had with any domain, or object.

Policy::destroy()

```
// IDL
void destroy();
```

This operation destroys the Policy object. It is the responsibility of the Policy object to determine whether it can be destroyed.

Enhancement Orbix guarantees to always destroy all local objects it creates when the last reference to them is released so you do not have to call `destroy()`. However, code that relies on this feature is not strictly CORBA compliant and may leak resources with other ORBs. (According to the CORBA specification, simply calling `CORBA::release()` on all references to a policy object does not delete the object or its components so each policy object created must be explicitly destroyed to avoid memory leaks.)

Exceptions

`NO_PERMISSION` The policy object determines that it cannot be destroyed.

CORBA::PolicyCurrent Class

The PolicyCurrent interface allows access to policy settings at the current programming context level. Within a client, you obtain a PolicyCurrent object reference to set the quality of service for all invocations in the current thread. You obtain a reference to this interface by invoking ORB::resolve_initial_references() with the ObjectId PolicyCurrent.

The PolicyCurrent interface is derived from the PolicyManager and the Current interfaces. The PolicyManager interface allows you to change the policies for each invocation and the Current interface allows control from the current thread.

Policies applied at the thread level override any system defaults or values set at the ORB level. When accessed from a newly spawned thread, the PolicyCurrent initially has no overridden policies. The PolicyCurrent also has no overridden values when a POA with ThreadPolicy of ORB_CONTROL_MODEL dispatches an invocation to a servant. Each time an invocation is dispatched through a POA of the SINGLE_THREAD_MODEL, the thread-level overrides are reset to have no overridden values.

```
class IT_ART_API PolicyCurrent :
    public virtual PolicyManager,
    public virtual Current
{
public:
    typedef CORBA::PolicyCurrent_ptr _ptr_type;
    typedef CORBA::PolicyCurrent_var _var_type;
    virtual ~PolicyCurrent();
    static PolicyCurrent_ptr _narrow(
        CORBA::Object_ptr obj
    );
    static PolicyCurrent_ptr _unchecked_narrow(
        CORBA::Object_ptr obj
    );
    inline static PolicyCurrent_ptr _duplicate(
        PolicyCurrent_ptr p
    );
    inline static PolicyCurrent_ptr _nil();
```

```
    static const IT_FWString _it_fw_type_id;
};
```

PolicyCurrent::_duplicate()

```
inline static PolicyCurrent_ptr _duplicate(
    PolicyCurrent_ptr p
);
```

Returns a duplicate object reference and increments the reference count of the object.

Parameters

p	The current object reference to duplicate.

See Also

"About Standard Functions for all Interfaces"

PolicyCurrent::_narrow()

```
static PolicyCurrent_ptr _narrow(
    CORBA::Object_ptr obj
);
```

Returns a new object reference to a `PolicyCurrent` object given an existing reference.

Parameters

obj	A reference to an object.

See Also

CORBA::PolicyCurrent::_unchecked_narrow()
"About Standard Functions for all Interfaces"

PolicyCurrent::_nil()

```
inline static PolicyCurrent_ptr _nil();
```

Returns a nil object reference to a `PolicyCurrent` object.

See Also

"About Standard Functions for all Interfaces"

PolicyCurrent::~PolicyCurrent() Destructor

```
virtual ~PolicyCurrent();
```

The destructor for the object.

PolicyCurrent::_unchecked_narrow()

```
static PolicyCurrent_ptr _unchecked_narrow(
    CORBA::Object_ptr obj
);
```

Returns a new object reference to a `PolicyCurrent` object given an existing reference.

Parameters

`obj`	A reference to an object.

See Also

`CORBA::PolicyCurrent::_narrow()`

"About Standard Functions for all Interfaces"

CORBA::PolicyManager Class

PolicyManager is an interface with operations for querying and overriding QoS policy settings. It includes mechanisms for obtaining policy override management operations at each relevant application scope. You obtain the ORB's PolicyManager by invoking ORB::resolve_initial_references() with the ObjectId ORBPolicyManager.

You use a CORBA::PolicyCurrent object, derived from CORBA::Current, for managing the thread's QoS policies. You obtain a reference to this interface by invoking ORB::resolve_initial_references() with the ObjectId PolicyCurrent.

- Accessor operations on CORBA::Object allow querying and overriding of QoS at the object reference scope.
- The application of QoS on a POA is done through the currently existing mechanism of passing a PolicyList to POA::create_POA().

```
class IT_ART_API PolicyManager : public virtual CORBA::Object {
public:
    typedef CORBA::PolicyManager_ptr _ptr_type;
    typedef CORBA::PolicyManager_var _var_type;
    virtual ~PolicyManager();
    static PolicyManager_ptr _narrow(
        CORBA::Object_ptr obj
    );
    static PolicyManager_ptr _unchecked_narrow(
        CORBA::Object_ptr obj
    );
    inline static PolicyManager_ptr _duplicate(
        PolicyManager_ptr p
    );
    inline static PolicyManager_ptr _nil();

    virtual PolicyList* get_policy_overrides(
        const PolicyTypeSeq &  ts
    ) = 0;
    virtual void set_policy_overrides(
        const PolicyList &  policies,
        SetOverrideType set_add
```

```
    ) = 0;
    static const IT_FWString _it_fw_type_id;
};
```

PolicyManager::_duplicate()

```
inline static PolicyManager_ptr _duplicate(
    PolicyManager_ptr p
);
```

Returns a duplicate object reference and increments the reference count of the object.

Parameters

`p`	The current object reference to duplicate.

See Also

"About Standard Functions for all Interfaces"

PolicyManager::get_policy_overrides()

```
virtual PolicyList* get_policy_overrides(
    const PolicyTypeSeq &  ts
) = 0;
```

Parameters Returns a list containing the overridden polices for the requested policy types. This returns only those policy overrides that have been set at the specific scope corresponding to the target `PolicyManager` (no evaluation is done with respect to overrides at other scopes). If none of the requested policy types are overridden at the target `PolicyManager`, an empty sequence is returned.

Parameters

`ts`	A sequence of policy types to get. If the specified sequence is empty, the method returns all policy overrides at this scope.

See Also `CORBA::PolicyManager::set_policy_overrides()`

PolicyManager::_narrow()

```
static PolicyManager_ptr _narrow(
    CORBA::Object_ptr obj
);
```

Returns a new object reference to a `PolicyManager` object given an existing reference.

Parameters

`obj`	A reference to an object.

See Also

`CORBA::PolicyManager::_unchecked_narrow()`

"About Standard Functions for all Interfaces"

PolicyManager::_nil()

```
inline static PolicyManager_ptr _nil();
```

Returns a nil object reference to a `PolicyManager` object.

See Also

"About Standard Functions for all Interfaces"

PolicyManager::~PolicyManager() Destructor

```
virtual ~PolicyManager();
```

The destructor for the object.

PolicyManager::set_policy_overrides()

```
virtual void set_policy_overrides(
    const PolicyList &  policies,
    SetOverrideType set_add
) = 0;
```

Modifies the current set of overrides with the requested list of policy overrides.

Parameters

`policies`	A sequence of references to policy objects.
`set_add`	Indicates whether the policies in the `policies` parameter should be added to existing overrides in the `PolicyManager` or used to replace existing overrides: • Use `ADD_OVERRIDE` to add policies onto any other overrides that already exist in the `PolicyManager`. • Use `SET_OVERRIDE` to create a clean `PolicyManager` free of any other overrides.

Invoking the method with an empty sequence of policies and a mode of `SET_OVERRIDE` removes all overrides from a `PolicyManager`.

There is no evaluation of compatibility with policies set within other policy managers.

Exceptions

`NO_PERMISSION`	Only certain policies that pertain to the invocation of an operation at the client end can be overridden using this operation. This exception is raised if you attempt to override any other policy.
`InvalidPolicied`	The request would put the set of overriding policies for the target `PolicyManager` in an inconsistent state. No policies are changed or added.

PolicyManager::_unchecked_narrow()

```
static PolicyManager_ptr _unchecked_narrow(
    CORBA::Object_ptr obj
);
```

Returns a new object reference to a `PolicyManager` object given an existing reference.

Parameters

`obj`	A reference to an object.

See Also `CORBA::PolicyManager::_narrow()`

“About Standard Functions for all Interfaces”

CORBA::PrimitiveDef Interface

Interface `PrimitiveDef` represents an IDL primitive type such as `short`, `long`, and others. `PrimitiveDef` objects are anonymous (unnamed) and owned by the interface repository.

Objects of type `PrimitiveDef` cannot be created directly. You can obtain a reference to a `PrimitiveDef` by calling `Repository::get_primitive()`.

```
// IDL in module CORBA.
interface PrimitiveDef: IDLType {
    readonly attribute PrimitiveKind kind;
};
```

See Also

`CORBA::PrimitiveKind`
`CORBA::IDLType`
`CORBA::Repository::get_primitive()`

PrimitiveDef::kind Attribute

```
// IDL
readonly attribute PrimitiveKind kind;
```

Identifies which of the IDL primitive types is represented by this `PrimitiveDef`.

A `PrimitiveDef` with a `kind` of type `pk_string` represents an unbounded string, a bounded string is represented by the interface `StringDef`. A `PrimitiveDef` with a `kind` of type `pk_objref` represents the IDL type `Object`. A `PrimitiveDef` with a `kind` of type `pk_value_base` represents the IDL type `ValueBase`.

See Also

`CORBA::IDLType`
`CORBA::Object`
`CORBA::StringDef`

CORBA::Repository Interface

The interface repository itself is a container for IDL type definitions. Each interface repository is represented by a global root `Repository` object.

The `Repository` interface describes the top-level object for a repository name space. It contains definitions of constants, typedefs, exceptions, interfaces, value types, value boxes, native types, and modules.

You can use the `Repository` operations to look up any IDL definition, by either name or identity, that is defined in the global name space, in a module, or in an interface. You can also use other `Repository` operations to create information for the interface repository. See Table 8:

Table 8: *Operations of the Repository Interface*

Read Operations	Write Operations
`describe_contents()`	`create_array()`
`get_canonical_typecode()`	`create_fixed()`
`get_primitive()`	`create_sequence()`
`lookup_id()`	`create_string()`
	`create_wstring()`

The five `create_`*`type`* operations create new interface repository objects defining anonymous types. Each anonymous type definition must be used in defining exactly one other object. Because the interfaces for these anonymous types are not derived from `Contained`, it is your responsibility to invoke in your application `destroy()` on the returned object if it is not successfully used in creating a definition that is derived from `Contained`.

The `Repository` interface is as follows:

```
// IDL in module CORBA.
interface Repository : Container {
    Contained lookup_id(
        in RepositoryId search_id
    );
    TypeCode get_canonical_typecode(
        in TypeCode tc
```

```
    );
    PrimitiveDef get_primitive(
        in PrimitiveKind kind
    );
    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
    );
    FixedDef create_fixed(
        in unsigned short digits,
        in short scale
    );
};
```

The inherited `describe_contents()` is also described.

Note that although a `Repository` does not have a `RepositoryId` associated with it (because it derives only from `Container` and not from `Contained`) you can assume that its default `RepositoryId`. is an empty string. This allows a value to be assigned to the `defined_in` field of each description structure for `ModuleDef`, `InterfaceDef`, `ValueDef`, `ValueBoxDef`, `TypedefDef`, `ExceptionDef` and `ConstantDef` that may be contained immediately within a `Repository` object.

See Also `CORBA::Container`

Repository::create_array()

```
// IDL
ArrayDef create_array(
    in unsigned long length,
    in IDLType element_type
```

```
);
```

Returns a new array object defining an anonymous (unnamed) type. The new array object must be used in the definition of exactly one other object. It is deleted when the object it is contained in is deleted. If the created object is not successfully used in the definition of a `Contained` object, it is your application's responsibility to delete it.

Parameters

`length`	The number of elements in the array.
`element_type`	The type of element that the array will contain.

See Also

`CORBA::ArrayDef`
`CORBA::IRObject`

Repository::create_fixed()

```
// IDL
FixedDef create_fixed (
    in unsigned short digits,
    in short scale
);
```

Returns a new fixed-point object defining an anonymous (unnamed) type. The new object must be used in the definition of exactly one other object. It is deleted when the object it is contained in is deleted. If the created object is not successfully used in the definition of a `Contained` object, it is your application's responsibility to delete it.

Parameters

`digits`	The number of digits in the fixed-point number. Valid values must be between 1 and 31, inclusive.
`scale`	The scale.

Repository::create_sequence()

```
// IDL
SequenceDef create_sequence (
    in unsigned long bound,
```

```
    in IDLType element_type
);
```

Returns a new sequence object defining an anonymous (unnamed) type. The new sequence object must be used in the definition of exactly one other object. It is deleted when the object it is contained in is deleted. If the created object is not successfully used in the definition of a `Contained` object, it is your application's responsibility to delete it.

Parameters

bound	The number of elements in the sequence. A bound of `0` indicates an unbounded sequence.
`element_type`	The type of element that the sequence will contain.

See Also `CORBA::SequenceDef`

Repository::create_string()

```
// IDL
StringDef create_string(
    in unsigned long bound
);
```

Returns a new string object defining an anonymous (unnamed) type. The new string object must be used in the definition of exactly one other object. It is deleted when the object it is contained in is deleted. If the created object is not successfully used in the definition of a `Contained` object, it is your application's responsibility to delete it.

Parameters

bound	The maximum number of characters in the string. (This cannot be `0`.)

Use `get_primitive()` to create unbounded strings.

See Also
`CORBA::StringDef`
`CORBA::Repository::get_primitive()`

Repository::create_wstring()

```
// IDL
StringDef create_wstring (
    in unsigned long bound
);
```

Returns a new wide string object defining an anonymous (unnamed) type. The new wide string object must be used in the definition of exactly one other object. It is deleted when the object it is contained in is deleted. If the created object is not successfully used in the definition of a `Contained` object, it is your application's responsibility to delete it.

Parameters

bound	The maximum number of characters in the string. (This cannot be `0`.)

Use `get_primitive()` to create unbounded strings.

See Also

`CORBA::WstringDef`
`CORBA::Repository::get_primitive()`

Repository::describe_contents()

```
// IDL
sequence<Description> describe_contents(
    in InterfaceName restrict_type,
    in boolean  exclude_inherited,
    in long  max_returned_objs
);
```

The operation `describe_contents()` is inherited from interface `Container`. It returns a sequence of `Container::Description` structures; one such structure for each top level item in the repository.

Parameters

`restrict_type`	If this is set to `dk_all`, then all of the contained interface repository objects are returned. If set to the `DefinitionKind` for a particular interface repository kind, it returns only objects of that kind. For example, if set to `dk_Operation`, then it returns contained operations only.
`exclude_inherited`	Applies only to interfaces. If true, no inherited objects are returned. If false, objects are returned even if they are inherited.
`max_returned_objs`	The number of objects that can be returned in the call. Setting a value of `-1` means return all contained objects.

See Also

`CORBA::Container::describe_contents()`
`CORBA::Container::Description`
`CORBA::DefinitionKind`

Repository::get_canonical_typecode()

```
// IDL
TypeCode get_canonical_typecode(
    in TypeCode tc
);
```

Returns a `TypeCode` that is equivalent to `tc` that also includes all repository ids, names, and member names.

Parameters

`tc`	The `TypeCode` to lookup.

If the top level `TypeCode` does not contain a `RepositoryId` (such as array and sequence type codes or type codes from older ORBs) or if it contains a `RepositoryId` that is not found in the target `Repository`, then a new `TypeCode` is constructed by recursively calling `get_canonical_typecode()` on each member `TypeCode` of the original `TypeCode`.

Repository::get_primitive()

```
// IDL
PrimitiveDef get_primitive(
    in PrimitiveKind kind
);
```

Returns a reference to a `PrimitiveDef` of the specified `PrimitiveKind`. All `PrimitiveDef` objects are owned by the `Repository`, one primitive object per primitive type (for example, `short`, `long`, `unsigned short`, `unsigned long` and so on).

Parameters

`kind`	The kind of primitive to get.

See Also `CORBA::PrimitiveDef`

Repository::lookup_id()

```
// IDL
Contained lookup_id(
    in RepositoryId search_id
);
```

Returns an object reference to a `Contained` object within the repository given its `RepositoryId`. If the repository does not contain a definition for the given ID, a nil object reference is returned.

Parameters

`search_id`	The `RepositoryId` of the IDL definition to lookup.

See Also `CORBA::Contained`

CORBA::Request Class

This class is the key support class for the Dynamic Invocation Interface (DII), whereby an application may issue a request for any interface, even if that interface was unknown at the time the application was compiled.

Orbix allows invocations, that are instances of class `Request`, to be constructed by specifying at runtime the target object reference, the operation name and the parameters. Such calls are termed dynamic because the IDL interfaces used by a program do not have to be statically determined at the time the program is designed and implemented.

You create a request using methods `Object::_create_request()` or `Object::_request()`.

```
class Request {
public:
    Object_ptr target() const;
    const char *operation() const;
    NVList_ptr arguments();
    NamedValue_ptr result();
    Environment_ptr env();
    ExceptionList_ptr exceptions();
    ContextList_ptr contexts();
    void ctx(Context_ptr);
    Context_ptr ctx() const;

    // argument manipulation helper functions
    Any &add_in_arg();
    Any &add_in_arg(const char* name);
    Any &add_inout_arg();
    Any &add_inout_arg(const char* name);
    Any &add_out_arg();
    Any &add_out_arg(const char* name);
    void set_return_type(TypeCode_ptr tc);
    Any &return_value();
    void invoke();
    void send_oneway();
    void send_deferred();
    void get_response();
```

```
    Boolean poll_response();

    // additional Messaging functions
    virtual void sendc(CORBA::Object_ptr  handler) = 0;
    virtual CORBA::Object_ptr sendp() = 0;
    virtual void prepare(CORBA::Object_ptr  p) = 0;

};
```

See Also CORBA::Object::_request()
CORBA::Object::_create_request()

Request::add_in_arg()

```
Any &add_in_arg();
```

```
Any &add_in_arg(
    const char* name
);
```

Returns an any value for the input argument that is added.

Parameters

name	The name for the argument that is added to the request.

See Also CORBA::Request::arguments()
CORBA::Request::add_inout_arg()
CORBA::Request::add_out_arg()

Request::add_inout_arg()

```
Any &add_inout_arg();
```

```
Any &add_inout_arg(
    const char* name
);
```

Returns an any value for the in/out argument that is added.

Parameters

name	The name for the argument that is added to the request.

See Also CORBA::Request::arguments()

CORBA::Request::add_in_arg()
CORBA::Request::add_out_arg()

Request::add_out_arg()

```
Any &add_out_arg();
```

```
Any &add_out_arg(
    const char* name
);
```

Returns an any value for the output argument that is added.

Parameters

name	The name for the argument that is added to the request.

See Also

CORBA::Request::arguments()
CORBA::Request::add_in_arg()
CORBA::Request::add_inout_arg()

Request::arguments()

```
NVList_ptr arguments();
```

Returns the arguments to the requested operation in an NVList. Ownership of the return value is maintained by the Request and must not be freed by the caller. You can add additional arguments to the request using the add_*_arg() helper methods.

See Also

CORBA::NVList
CORBA::Request::add_in_arg()
CORBA::Request::add_inout_arg()
CORBA::Request::add_out_arg()

Request::contexts()

```
ContextList_ptr contexts();
```

Returns a pointer to a list of contexts for the request. Ownership of the return value is maintained by the Request and must not be freed by the caller.

See Also

CORBA::ContextList

Request::ctx()

```
Context_ptr ctx() const;
```

Returns the Context associated with a request. Ownership of the return value is maintained by the Request and must not be freed by the caller.

```
void ctx(
    Context_ptr c
);
```

Inserts a Context into a request.

Parameters

c	The context to insert with the request.

See Also CORBA::Context

Request::env()

```
Environment_ptr env();
```

Returns the Environment associated with the request from which exceptions raised in DII calls can be accessed. Ownership of the return value is maintained by the Request and must not be freed by the caller.

See Also CORBA::Environment

Request::exceptions()

```
ExceptionList_ptr exceptions();
```

Returns a pointer to list of possible application-specific exceptions for the request. Ownership of the return value is maintained by the Request and must not be freed by the caller.

See Also CORBA::ExceptionList

Request::get_response()

```
void get_response();
```

Determines whether a request has completed successfully. It returns only when the request, invoked previously using send_deferred(), has completed.

See Also

CORBA::Request::result()
CORBA::Request::send_deferred()

Request::invoke()

```
void invoke();
```

Instructs the ORB to make a request. The parameters to the request must already be set up. The caller is blocked until the request has been processed by the target object or an exception occurs.

To make a non-blocking request, see send_deferred() and send_oneway().

See Also

CORBA::Request::send_oneway()
CORBA::Request::send_deferred()
CORBA::Request::result()

Request::operation()

```
const char *operation() const;
```

Returns the operation name of the request. Ownership of the return value is maintained by the Request and must not be freed by the caller.

Request::poll_response()

```
Boolean poll_response();
```

Returns 1 (true) if the operation has completed successfully and indicates that the return value and out and inout parameters in the request are valid. Returns 0 (false) otherwise. The method returns immediately.

If your application makes an operation request using send_deferred(), it can call poll_response() to determine whether the operation has completed. If the operation has completed, you can get the result by calling Request::result().

See Also

CORBA::Request::send_deferred()
CORBA::Request::get_response()
CORBA::Request::result()

Request::prepare()

```
virtual void prepare(
    CORBA::Object_ptr  p
) = 0;
```

Associates an initialized Request with a previous operation that was initiated via sendp(). The Request must be created and associated with the operation's out arguments and return value prior to calling prepare(). Once prepare() has been called, it is as if that prepared Request was the one that actually had sendp() used.

Parameters

p	An object reference.

This function along with sendp() and sendc() enable dynamic time-Independent invocations and dynamic use of the Messaging callback model.

Exceptions

BAD_INV_ORDER	prepare() is invoked on a Request that had previously been used for a send or one of its variants.
BAD_PARAM	prepare() is invoked with an object reference that was not previously returned from an invocation of sendp().

See Also

CORBA::Request::sendp()
CORBA::Request::sendc()

Request::result()

```
NamedValue_ptr result();
```

Returns the result of the operation request in a NamedValue. Ownership of the return value is maintained by the Request and must not be freed by the caller.

Request::return_value()

```
Any &return_value();
```

Returns an any value for the returned value of the operation.

Request::sendc()

```
virtual void sendc(
    CORBA::Object_ptr  handler
) = 0;
```

Initiates an operation according to the information in the Request.

Parameters

handler	Pass in the callback Messaging::ReplyHandler as a base CORBA::Object. The results of invocations made with sendc() will be available through this handler.

A truly dynamic client can implement the ReplyHandler using the DSI.

Exceptions A system exception may be raise if a failure is detected before control is returned to the client, but this is not guaranteed. Any other exceptions are passed to the ReplyHandler.

See Also CORBA::Request::sendp()
CORBA::Request::prepare()

Request::send_deferred()

```
void send_deferred();
```

Instructs the ORB to make the request. The arguments to the request must already be set up. The caller is not blocked, and thus may continue in parallel with the processing of the call by the target object.

To make a blocking request, use `invoke()`. You can use `poll_response()` to determine whether the operation completed.

See Also

`CORBA::Request::send_oneway()`
`CORBA::ORB::send_multiple_requests_deferred()`
`CORBA::Request::invoke()`
`CORBA::Request::poll_response()`
`CORBA::Request::get_response()`

Request::send_oneway()

```
void send_oneway();
```

Instructs Orbix to make the oneway request. The arguments to the request must already be set up. The caller is not blocked, and thus may continue in parallel with the processing of the call by the target object.

You can use this method even if the operation has not been defined to be oneway in its IDL definition, however, do not expect any output or inout parameters to be updated.

To make a blocking request, use `invoke()`.

See Also

`CORBA::Request::send_deferred()`
`CORBA::ORB::send_multiple_requests_oneway()`
`CORBA::Request::invoke()`
`CORBA::Request::poll_response()`
`CORBA::Request::get_response()`

Request::sendp()

```
virtual CORBA::Object_ptr sendp() = 0;
```

Initiates an operation according to the information in the `Request`. The results of invocations made with `sendp()` will be available once the caller uses

get_response() or get_next_response(). The out parameters and return value of the initiated operation must not be used before the operation is done.

Exceptions A system exception may be raise if a failure is detected before control is returned to the client, but this is not guaranteed. Any other exceptions will be raised when get_response() is called.

See Also CORBA::Request::sendc()
CORBA::Request::prepare()

Request::set_return_type()

```
void set_return_type(
    TypeCode_ptr tc
);
```

Sets the TypeCode associated with a Request object. When using the DII with the Internet Inter-ORB Protocol (IIOP), you must set the return type of a request before invoking the request.

Parameters

tc	The TypeCode for the return type of the operation associated with the Request object.

Request::target()

```
Object_ptr target() const;
```

Gets the target object of the Request. Ownership of the return value is maintained by the Request and must not be freed by the caller.

CORBA::SequenceDef Interface

Interface SequenceDef represents an IDL sequence definition in the interface repository. It inherits from the interface IDLType.

```
// IDL in module CORBA.
interface SequenceDef : IDLType {
    attribute unsigned long bound;
    readonly attribute TypeCode element_type;
    attribute IDLType element_type_def;
};
```

The inherited type attribute is also described.

See Also

CORBA::IDLType
CORBA::Repository::create_sequence()

SequenceDef::bound Attribute

```
// IDL
attribute unsigned long bound;
```

The maximum number of elements in the sequence. A bound of 0 indicates an unbounded sequence.

Changing the bound attribute will also update the inherited type attribute.

See Also

CORBA::SequenceDef::type

SequenceDef::element_type Attribute

```
// IDL
readonly attribute TypeCode element_type;
```

The type of element contained within this sequence. The attribute element_type_def contains the same information.

See Also

CORBA::SequenceDef::element_type_def

SequenceDef::element_type_def Attribute

```
// IDL
attribute IDLType element_type_def;
```

Describes the type of element contained within this sequence. The attribute `element_type` contains the same information. Setting the `element_type_def` attribute also updates the `element_type` and `IDLType::type` attributes.

See Also

`CORBA::SequenceDef::element_type`
`CORBA::IDLType::type`

SequenceDef::type Attribute

```
// IDL
readonly attribute TypeCode type;
```

The `type` attribute is inherited from interface `IDLType`. This attribute is a `tk_sequence` `TypeCode` that describes the sequence. It is updated automatically whenever the attributes `bound` or `element_type_def` are changed.

See Also

`CORBA::SequenceDef::element_type_def`
`CORBA::SequenceDef::bound`

CORBA::ServerRequest Class

Class `ServerRequest` describes a Dynamic Skeleton Interface (DSI) operation request. It is analogous to the `Request` class used in the Dynamic Invocation Interface (DII).

An instance of `ServerRequest` is created by the ORB when it receives an incoming request that is to be handled by the DSI—that is, an instance of the `PortableServer::DynamicImplementation` class has been registered to handle the target interface.

An instance of `ServerRequest` is a pseudo-object so an instance of a `ServerRequest` cannot be transmitted in an IDL operation.

You should not define derived classes of `ServerRequest`.

The following code is the complete class definition:

```
// in CORBA namespace
class ServerRequest {
public:
    const char* operation() const;
    void arguments(
        NVList_ptr& parameters
    );
    Context_ptr ctx();
    void set_result(
        const Any& value
    );
    void set_exception(
        const Any& value
    );
};
```

ServerRequest::arguments()

```
void arguments(
    NVList_ptr& parameters
);
```

Allows a redefinition of the following method to specify the values of incoming arguments:

`PortableServer::DynamicImplementation::invoke()`

Parameters

`parameters`	Obtains output and input arguments.

This method must be called *exactly* once in each execution of `invoke()`.

See Also `CORBA::ServerRequest::params()`
`PortableServer::DynamicImplementation::invoke()`

ServerRequest::ctx()

```
Context_ptr ctx();
```

Returns the `Context` associated with the call.

This function can be called once or not at all. If it is called, it must be called before `params()` or `ServerRequest::arguments()`.

See Also `CORBA::Context`

ServerRequest::operation()

```
const char* operation() const;
```

Parameters Returns the name of the operation being invoked.

This method must be called at least once in each execution of the dynamic implementation routine, that is, in each redefinition of the method:

`PortableServer::DynamicImplementation::invoke()`

See Also `CORBA::ServerRequest::op_name()`
`PortableServer::DynamicImplementation::invoke()`

See Also

ServerRequest::set_exception()

```
void set_exception(
    const Any& value
);
```

Allows (a redefinition of) `PortableServer::DynamicImplementation::invoke()` to return an exception to the caller.

Parameters

`value`	A pointer to an `Any`, which holds the exception returned to the caller.

See Also `CORBA::Environment()`
`PortableServer::DynamicImplementation::invoke()`

ServerRequest::set_result()

```
void set_result(
    const Any& value
);
```

Allows `PortableServer::DynamicImplementation::invoke()` to return the result of an operation request in an `Any`.

Parameters

`value`	A pointer to a `Any`, which holds the result returned to the caller.

This method must be called once for operations with non-`void` return types and not at all for operations with `void` return types. If it is called, then `set_exception()` cannot be used.

See Also `CORBA::ServerRequest::set_exception()`

CORBA::StringDef Interface

Interface `StringDef` represents an IDL bounded string type in the interface repository. A `StringDef` object is anonymous, which means it is unnamed.

Use `Repository::create_string()` to obtain a new `StringDef`. Use `Repository::get_primitive()` for unbounded strings.

```
// IDL in module CORBA.
interface StringDef : IDLType {
    attribute unsigned long bound;
};
```

The inherited `type` attribute is also described.

See Also

`CORBA::IDLType`
`CORBA::Repository::create_string()`

StringDef::bound Attribute

```
// IDL
attribute unsigned long bound;
```

Specifies the maximum number of characters in the string. This cannot be zero.

StringDef::type Attribute

```
// IDL
readonly attribute TypeCode type;
```

The `type` attribute is inherited from interface `IDLType`. This attribute is a `tk_string` `TypeCode` that describes the string.

See Also

`CORBA::IDLType::type`

CORBA::String_var Class

The class String_var implements the _var type for IDL strings required by the standard C++ mapping. The String_var class contains a char* value and ensures that this is properly freed when a String_var object is deallocated, for example when exectution goes out of scope.

```
class String_var {
public:
    String_var();
    String_var(char *p);
    String_var(const char *p);
    String_var(const String_var &s);
    ~String_var();
    String_var & operator=(char *p);
    String_var & operator=(const char *p);
    String_var & operator=(const String_var &s);
    operator char*();
    operator const char*() const;
    const char* in() const;
    char*& inout();
    char*& out();
    char* _retn();
    char & operator[](ULong index);
    char operator[](ULong index) const;
};
```

String_var::char*()

```
operator char*();
operator const char*() const;
```

Converts a String_var object to a char*.

See Also CORBA::String_var::operator=()

String_var::in()

```
const char* in() const;
```

Returns the proper string for use as an input parameter.

See Also

CORBA::String_var::out()
CORBA::String_var::inout()
CORBA::String_var::_retn()

String_var::inout()

```
char*& inout();
```

Returns the proper string for use as an inout parameter.

See Also

CORBA::String_var::in()
CORBA::String_var::out()
CORBA::String_var::_retn()

String_var::operator=() Assignment Operators

```
String_var &operator=(
    char *p
);

String_var &operator=(
    const char *p
);

String_var &operator=(
    const String_var &s
);
```

Assignment operators allow you to assign values to a `String_var` from a `char*` or from another `String_var` type.

Parameters

`p`	A character string to assign to the `String_var`.
`s`	A `String_var` to assign to the `String_var`.

See Also

CORBA::String_var::char*()

String_var::operator[]() Subscript Operators

```
char &operator[](
    ULong index
);

char operator[](
    ULong index
) const;
```

Return the character at the given location of the string. Subscript operators allow access to the individual characters in the string.

Parameters

`index`	The index location in the string.

String_var::out()

```
char*& out();
```

Returns the proper string for use as an output parameter.

See Also

CORBA::String_var::in()
CORBA::String_var::inout()
CORBA::String_var::_retn()

String_var::String_var() Constructors

```
String_var();
```

The default constructor.

```
String_var(
    char *p
);

String_var(
    const char *p
);
```

Constructors that convert from a `char*` to a `String_var`.

```
String_var(
    const String_var &s
);
```

The copy constructor.

Parameters

p	The character string to convert to a `String_var`. The `String_var` assumes ownership of the parameter.
s	The original `String_var` that is copied.

See Also CORBA::String_var::~String_var()

String_var::~String_var() Destructor

```
~String_var();
```

The destructor.

See Also CORBA::String_var::String_var()

String_var::_retn()

```
char* _retn();
```

Returns the proper string for use as a method's return value.

See Also CORBA::String_var::inout()
CORBA::String_var::in()
CORBA::String_var::out()

CORBA::StructDef Interface

Interface StructDef describes an IDL structure in the interface repository.

```
// IDL in module CORBA.
interface StructDef : TypedefDef, Container {
    attribute StructMemberSeq members;
};
```

The inherited operation describe() is also described.

See Also

CORBA::Contained
CORBA::Container::create_struct()

StructDef::describe()

```
// IDL
Description describe();
```

describe(returns a Contained::Description structure. describe() is inherited from Contained (which TypedefDef inherits).

The DefinitionKind for the kind member is dk_Struct. The value member is an any whose TypeCode is _tc_TypeDescription and whose value is a structure of type TypeDescription.

See Also

CORBA::TypedefDef::describe()

StructDef::members Attribute

```
// IDL
attribute StructMemberSeq members;
```

Describes the members of the structure.

You can modify this attribute to change the members of a structure. Only the name and type_def fields of each StructMember should be set (the type field should be set to _tc_void and it will be set automatically to the TypeCode of the type_def field).

See Also CORBA::TypedefDef

CORBA::TypeCode Class

The class TypeCode is used to describe IDL type structures at runtime. A TypeCode is a value that represents an IDL invocation argument type or an IDL attribute type. A TypeCode is typically used as follows:

- In the dynamic invocation interface (DII) to indicate the type of an actual argument.
- By the interface repository to represent the type specification that is part of an OMG IDL declaration.
- To describe the data held by an any type.

A TypeCode consists of a *kind* that classifies the TypeCode as to whether it is a basic type, a structure, a sequence and so on. See the data type TCKind for all possible kinds of TypeCode objects.

A TypeCode may also include a sequence of parameters. The parameters give the details of the type definition. For example, the IDL type sequence<long, 20> has the kind tk_sequence and has parameters long and 20.

You typically obtain a TypeCode from the interface repository or it may be generated by the IDL compiler. You do not normally create a TypeCode in your code so the class contains no constructors, only methods to decompose the components of an existing TypeCode. However, if your application does require that you create a TypeCode, see the set of create_*Type*_tc() methods in the ORB class.

For functions that require TypeCode parameters, such as with the DII, you can use the appropriate constant from the following list:

```
CORBA::_tc_any                  CORBA::_tc_octet
CORBA::_tc_boolean              CORBA::_tc_Principal
CORBA::_tc_char                 CORBA::_tc_short
CORBA::_tc_double               CORBA::_tc_string
CORBA::_tc_float                CORBA::_tc_TypeCode
CORBA::_tc_long                 CORBA::_tc_ulong
CORBA::_tc_longdouble           CORBA::_tc_ulonglong
CORBA::_tc_longlong             CORBA::_tc_ushort
CORBA::_tc_NamedValue           CORBA::_tc_void
CORBA::_tc_null                 CORBA::_tc_wchar
CORBA::_tc_Object               CORBA::_tc_wstring
```

The class TypeCode contains the following methods:

```
// C++
class TypeCode {
public:
    class Bounds : public UserException { ... };
    class BadKind : public UserException { ... };
    Boolean equal(TypeCode_ptr) const;
    Boolean equivalent(TypeCode_ptr) const;
    TCKind kind() const;
    TypeCode_ptr get_compact_typecode() const;
    const char* id() const;
    const char* name() const;
    ULong member_count() const;
    const char* member_name(ULong index) const;
    TypeCode_ptr member_type(ULong index) const;
    Any* member_label(ULong index) const;
    TypeCode_ptr discriminator_type() const;
    Long default_index() const;
    ULong length() const;
    TypeCode_ptr content_type() const;
    UShort fixed_digits() const;
    Short fixed_scale() const;
    Visibility member_visibility(ULong index) const;
    ValueModifier type_modifier() const;
    TypeCode_ptr concrete_base_type() const;

    static TypeCode_ptr _duplicate(TypeCode_ptr tc);
```

```
    static TypeCode_ptr _nil();

};
```

See Also CORBA::TCKind

TypeCode::BadKind Exception

```
class BadKind : public UserException { ... };
```

The BadKind exception is raised if a TypeCode member method is invoked for a kind that is not appropriate.

TypeCode::Bounds Exception

```
class Bounds : public UserException { ... };
```

The Bounds exception is raised if an attempt is made to use an index for a type's member that is greater than or equal to the number of members for the type.

The type of IDL constructs that have members include enumerations, structures, unions, value types, and exceptions. Some of the TypeCode methods return information about specific members of these IDL constructs. The first member has index value 0, the second has index value 1, and so on up to *n*-1 where *n* is the count of the total number of members.

The order in which members are presented in the interface repository is the same as the order in which they appeared in the IDL specification.

This exception is not the same as the CORBA::Bounds exception.

See Also
CORBA::TypeCode::member_count()
CORBA::TypeCode::member_label()
CORBA::TypeCode::member_name()
CORBA::TypeCode::member_type()
CORBA::TypeCode::member_visibility()

TypeCode::concrete_base_type()

```
TypeCode_ptr concrete_base_type() const;
```

Returns a TypeCode for the concrete base if the value type represented by this TypeCode has a concrete base value type. Otherwise it returns a nil TypeCode reference. This method is valid to use only if the kind of TypeCode has a TCKind value of tk_value.

Exceptions

BadKind	The kind of TypeCode is not valid for this method.

TypeCode::content_type()

```
TypeCode_ptr content_type() const;
```

For sequences and arrays this method returns a reference to the element type. For aliases it returns a reference to the original type. For a boxed value type it returns a reference to the boxed type. This method is valid to use if the kind of TypeCode is one of the following TCKind values:

```
tk_alias
tk_array
tk_sequence
tk_value_box
```

Exceptions

BadKind	The kind of TypeCode is not valid for this method.

TypeCode::default_index()

```
Long default_index() const;
```

Returns the index of the default union member, or -1 if there is no default member. This method is valid to use only if the kind of TypeCode has a TCKind value of tk_union.

Exceptions

BadKind	The kind of TypeCode is not valid for this method.

See Also CORBA::TypeCode::member_label()

TypeCode::discriminator_type()

```
TypeCode_ptr discriminator_type() const;
```

Returns a TypeCode for the union discriminator type. This method is valid to use only if the kind of TypeCode has a TCKind value of tk_union.

Exceptions

BadKind	The kind of TypeCode is not valid for this method.

See Also CORBA::TypeCode::default_index()
CORBA::TypeCode::member_label()

TypeCode::_duplicate()

```
static TypeCode_ptr _duplicate(
    TypeCode_ptr obj
);
```

Increments the reference count of obj and returns a new reference to the TypeCode object.

Parameters

obj	A reference to the original TypeCode to duplicate.

See Also CORBA::release()

TypeCode::equal()

```
Boolean equal(
    TypeCode_ptr tc
) const;
```

Returns 1 (true) if this TypeCode and the tc parameter are equal. Returns 0 (false) otherwise. Two type codes are equal if the set of legal operations is the same and invoking an operation from one set returns the same results as invoking the operation from the other set.

Parameters

tc	The TypeCode to compare.

See Also CORBA::TypeCode::equivalent()

TypeCode::equivalent()

```
Boolean equivalent(
    TypeCode_ptr tc
) const;
```

Returns 1 (true) if this TypeCode and the tc parameter are equivalent. Returns 0 (false) otherwise.

Parameters

tc	The TypeCode to compare.

equivalent() is typically used by the ORB to determine type equivalence for values stored in an IDL any. You can use equal() to compare type codes in your application. equivalent() would return true if used to compare a type and an alias of that type while equal() would return false.

See Also CORBA::TypeCode::equal()

TypeCode::fixed_digits()

```
UShort fixed_digits() const;
```

Returns the number of digits in the fixed point type. This method is valid to use only if the kind of TypeCode has a TCKind value of tk_fixed.

Exceptions

BadKind	The kind of TypeCode is not valid for this method.

See Also CORBA::TypeCode::fixed_scale()

TypeCode::fixed_scale()

```
Short fixed_scale() const;
```

Returns the scale of the fixed point type. This method is valid to use only if the kind of TypeCode has a TCKind value of tk_fixed.

Exceptions

BadKind	The kind of TypeCode is not valid for this method.

See Also CORBA::TypeCode::fixed_digits()

TypeCode::get_compact_typecode()

```
TypeCode_ptr get_compact_typecode() const;
```

Removes all optional name and member name fields from the TypeCode and returns a reference to the compact TypeCode. This method leaves all alias type codes intact.

TypeCode::id()

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

Returns the RepositoryId that globally identifies the type.

Type codes that always have a RepositoryId. include object references, value types, boxed value types, native, and exceptions. Other type codes that also always have a RepositoryId and are obtained from the interface repository or ORB::create_operation_list() include structures, unions, enumerations, and aliases. In other cases id() could return an empty string.

The TypeCode object maintains the memory of the return value; this return value must not be freed by the caller.

This method is valid to use if the kind of TypeCode has a TCKind value of one of the following:

```
tk_abstract_interface
tk_alias
tk_enum
tk_except
tk_native
tk_objref
tk_struct
tk_union
tk_value
tk_value_box
```

Exceptions

BadKind	The kind of TypeCode is not valid for this method.

TypeCode::kind()

```
TCKind kind() const;
```

Returns the kind of the TypeCode which is an enumerated value of type TCKind. You can use kind() on any TypeCode to help determine which other TypeCode methods can be invoked on the TypeCode.

See Also CORBA::TCKind

TypeCode::length()

```
ULong length() const;
```

For strings, wide strings, and sequences, length() returns the bound, with zero indicating an unbounded string or sequence. For arrays, length() returns the number of elements in the array. This method is valid to use if the kind of TypeCode has a TCKind value of one of the following:

```
tk_array
tk_sequence
tk_string
tk_wstring
```

Exceptions

BadKind	The kind of TypeCode is not valid for this method.

TypeCode::member_count()

```
ULong member_count() const;
```

Returns the number of members in the type. This method is valid to use if the kind of TypeCode has a TCKind value of one of the following:

```
tk_enum
tk_except
tk_struct
tk_union
tk_value
```

Exceptions

BadKind	The kind of TypeCode is not valid for this method.

TypeCode::member_label()

```
Any *member_label(
    ULong index
) const;
```

Returns the label of the union member. For the default member, the label is the zero octet. This method is valid to use only if the kind of TypeCode has a TCKind value of tk_union.

Parameters

index	The index indicating which union member you want.

Exceptions

BadKind	The kind of TypeCode is not valid for this method.
Bounds	The index parameter is greater than or equal to the number of members for the type.

See Also

CORBA::TypeCode::default_index()
CORBA::TypeCode::member_count()

TypeCode::member_name()

```
const char* member_name(
    ULong index
) const;
```

Returns the simple name of the member. Because names are local to a repository, the name returned from a TypeCode may not match the name of the member in any particular repository, and may even be an empty string.

Parameters

index	The index indicating which member to use.

This method is valid to use if the kind of TypeCode has a TCKind value of one of the following:

```
tk_enum
tk_except
tk_struct
tk_union
tk_value
```

The TypeCode object maintains the memory of the return value; this return value must not be freed by the caller.

Exceptions

BadKind	The kind of TypeCode is not valid for this method.
Bounds	The index parameter is greater than or equal to the number of members for the type.

See Also CORBA::TypeCode::member_count()

TypeCode::member_type()

```
TypeCode_ptr member_type(
    ULong index
) const;
```

Returns a reference to the TypeCode of the member identified by index.

Parameters

index	The index indicating which member you want.

This method is valid to use if the kind of TypeCode has a TCKind value of one of the following:

```
tk_except
tk_struct
tk_union
tk_value
```

Exceptions

BadKind	The kind of TypeCode is not valid for this method.
Bounds	The index parameter is greater than or equal to the number of members for the type.

See Also CORBA::TypeCode::member_count()

TypeCode::member_visibility()

```
Visibility member_visibility(
    ULong index
```

```
) const;
```

Returns the Visibility of a value type member. This method is valid to use only if the kind of TypeCode has a TCKind value of tk_value.

Parameters

index	The index indicating which value type member you want.

Exceptions

BadKind	The kind of TypeCode is not valid for this method.
Bounds	The index parameter is greater than or equal to the number of members for the type.

See Also

CORBA::Visibility
CORBA::TypeCode::member_count()

CORBA::TypeCode::member_count() TypeCode::name()

```
const char* name() const;
```

Returns the simple name identifying the type within its enclosing scope. Because names are local to a repository, the name returned from a TypeCode may not match the name of the type in any particular repository, and may even be an empty string.

The TypeCode object maintains the memory of the return value; this return value must not be freed by the caller.

This method is valid to use if the kind of TypeCode has a TCKind value of one of the following:

```
tk_abstract_interface
tk_alias
tk_enum
tk_except
tk_native
tk_objref
tk_struct
tk_union
tk_value
tk_value_box
```

Exceptions

BadKind	The kind of TypeCode is not valid for this method.

TypeCode::_nil()

```
static TypeCode_ptr _nil();
```

Returns a nil object reference for a TypeCode.

See Also CORBA::is_nil()

TypeCode::type_modifier()

```
ValueModifier type_modifier() const;
```

Returns the ValueModifier that applies to the value type represented by this TypeCode. This method is valid to use only if the kind of TypeCode has a TCKind value of tk_value.

Exceptions

BadKind	The kind of TypeCode is not valid for this method.

CORBA::TypedefDef Interface

The abstract interface `TypedefDef` is simply a base interface for interface repository interfaces that define named types. Named types are types for which a name must appear in their definition such as structures, unions, and so on. Interfaces that inherit from typedefDef include:

- `AliasDef`
- `EnumDef`
- `NativeDef`
- `StructDef`
- `UnionDef`
- `ValueBoxDef`

Anonymous types such as `PrimitiveDef`, `StringDef`, `SequenceDef` and `ArrayDef` do not inherit from `TypedefDef`.

```
//IDL in module CORBA.
interface TypedefDef : Contained, IDLType {};
```

The inherited operation `describe()` is described here.

TypedefDef::describe()

```
//IDL
Description describe();
```

Inherited from `Contained`, `describe()` returns a structure of type `Contained::Description`.

The `DefinitionKind` type for the `kind` member is `dk_Typedef`. The value member is an any whose `TypeCode` is `_tc_TypeDescription` and whose value is a structure of type `TypeDescription`.

See Also

`CORBA::Contained::describe()`
`CORBA::Contained::Description`
`CORBA::TypeDescription`

CORBA::UnionDef Interface

Interface UnionDef represents an IDL union in the interface repository.

```
// IDL in module CORBA.
interface UnionDef : TypedefDef {
    readonly attribute TypeCode discriminator_type;
    attribute IDLType discriminator_type_def;
    attribute UnionMemberSeq members;
};
```

The inherited operation describe() is also described.

See Also

CORBA::Contained
CORBA::TypedefDef
CORBA::Container::create_union()

UnionDef::describe()

```
// IDL
Description describe();
```

Inherited from Contained (which TypedefDef inherits), describe() returns a structure of type Contained::Description.

The DefinitionKind for the kind member is dk_Union. The value member is an any whose TypeCode is _tc_TypeDescription and whose value is a structure of type TypeDescription.

See Also

CORBA::TypedefDef::describe()

UnionDef::discriminator_type Attribute

```
// IDL
readonly attribute TypeCode discriminator_type;
```

Describes the discriminator type for this union. For example, if the union currently contains a long, the discriminator_type is _tc_long. The attribute discriminator_type_def contains the same information.

See Also CORBA::TypeCode

UnionDef::discriminator_type_def Attribute

```
// IDL
attribute IDLType discriminator_type_def;
```

Describes the discriminator type for this union. The attribute discriminator_type contains the same information.

Changing this attribute will automatically update the discriminator_type attribute and the IDLType::type attribute.

See Also CORBA::IDLType::type
CORBA::UnionDef::discriminator_type

UnionDef::members Attribute

```
// IDL
attribute UnionMemberSeq members;
```

Contains a description of each union member: its name, label, and type (type and type_def contain the same information).

The members attribute can be modified to change the union's members. Only the name, label and type_def fields of each UnionMember should be set (the type field should be set to _tc_void, and it will be set automatically to the TypeCode of the type_def field).

See Also CORBA::TypedefDef

CORBA::ValueBase Class

All value types have a conventional base type called ValueBase. ValueBase serves a similar role for value types that the Object class serves for interfaces. ValueBase serves as an abstract base class for all value type classes. You must implement concrete value type classes that inherit from ValueBase. ValueBase provides several pure virtual reference counting methods inherited by all value type classes.

```
namespace CORBA {
    class ValueBase {
    public:
        virtual ValueBase* _add_ref() = 0;
        virtual void _remove_ref() = 0;
        virtual ValueBase* _copy_value() = 0;
        virtual ULong _refcount_value() = 0;
        static ValueBase* _downcast(ValueBase*);
    protected:
        ValueBase();
        ValueBase(const ValueBase&);
        virtual ~ValueBase();
    ...
     };
}
```

The names of these methods begin with an underscore to keep them from clashing with your application-specific methods in derived value type classes.

See Also CORBA::ValueFactory

ValueBase::_add_ref()

```
virtual ValueBase* _add_ref() = 0;
```

Increments the reference count of a value type instance and returns a pointer to this value type.

See Also CORBA::ValueBase::_remove_ref()

ValueBase::_copy_value()

```
virtual ValueBase* _copy_value() = 0;
```

Makes a deep copy of the value type instance and returns a pointer to the copy. The copy has no connections with the original instance and has a lifetime independent of that of the original.

Portable applications should not assume covariant return types but should use downcasting to regain the most derived type of a copied value type. A covariant return type means that a class derived from `ValueBase` can override `_copy_value()` to return a pointer to the derived class rather than the base class, `ValueBase*`.

See Also CORBA::ValueBase::_downcast()

ValueBase::_downcast()

```
static ValueBase* _downcast(
    ValueBase* vt
);
```

Returns a pointer to the base type for a derived value type class.

Parameters

`vt`	Pointer to the value type class to be downcast.

ValueBase::_refcount_value()

```
virtual ULong _refcount_value() = 0;
```

Returns the current value of the reference count for this value type instance.

See Also CORBA::ValueBase::_add_ref()
CORBA::ValueBase::_remove_ref()

ValueBase::_remove_ref()

```
virtual _remove_ref() = 0;
```

Decrements the reference count of a value type instance and deletes the instance when the reference count drops to zero.

If you use `delete()` to destroy instances, you must use the `new` operator to allocate all value type instances.

See Also CORBA::ValueBase::_add_ref()

ValueBase::~ValueBase() Destructor

```
protected:
    virtual ~ValueBase();
```

The default destructor.

The destructor is protected to prevent direct deletion of instances of classes derived from `ValueBase`.

See Also CORBA::ValueBase::ValueBase()

ValueBase::ValueBase() Constructors

```
protected:
    ValueBase();
```

The default constructor.

```
protected:
    ValueBase(
        const ValueBase& vt
    );
```

The copy constructor. Creates a new object that is a copy of `vt`.

The copy constructor is protected to disallow copy construction of derived value type instances except from within derived class methods.

Parameters

`vt`	The original value type from which a copy is made.

See Also CORBA::ValueBase::~ValueBase()

CORBA::ValueBoxDef Interface

The ValueBoxDef interface describes an IDL value box type in the interface repository. A value box is a value type with no inheritance or operations and with a single state member. A value box is a shorthand IDL notation used to simplify the use of value types for simple containment. It behaves like an additional namespace that contains only one name.

```
// IDL in module CORBA.
interface ValueBoxDef : IDLType {
    attribute IDLType original_type_def;
};
```

The inherited type attribute is also described.

See Also CORBA::Container::create_value_box()

ValueBoxDef::original_type_def Attribute

```
// IDL
attribute IDLType original_type_def;
```

Identifies the IDL type_def that is being “boxed”. Setting the original_type_def attribute also updates the type attribute.

See Also CORBA::ValueBoxDef::type

ValueBoxDef::type Attribute

```
// IDL
readonly attribute TypeCode type;
```

Inherited from IDLType, this attribute is a tk_value_box TypeCode describing the value box.

See Also CORBA::IDLType::type

CORBA::ValueDef Interface

A ValueDef object represents an IDL value type definition in the interface repository. It can contain constants, types, exceptions, operations, and attributes.

A ValueDef used as a Container may only contain TypedefDef, (including definitions derived from TypedefDef), ConstantDef, and ExceptionDef definitions.

```
// IDL in module CORBA.
interface ValueDef : Container, Contained, IDLType {

    // read/write interface
    attribute InterfaceDef supported_interfaces;
    attribute InitializerSeq initializers;
    attribute ValueDef base_value;
    attribute ValueDefSeq abstract_base_values;
    attribute boolean is_abstract;
    attribute boolean is_custom;
    attribute boolean is_truncatable;

    // read interface
    boolean is_a(
        in RepositoryId id
    );
    struct FullValueDescription {
        Identifier name;
        RepositoryId id;
        boolean is_abstract;
        boolean is_custom;
        RepositoryId defined_in;
        VersionSpec version;
        OpDescriptionSeq operations;
        AttrDescriptionSeq attributes;
        ValueMemberSeq members;
        InitializerSeq initializers;
        RepositoryIdSeq supported_interfaces;
        RepositoryIdSeq abstract_base_values;
        boolean is_truncatable;
```

```
            RepositoryId base_value;
            TypeCode type;
        };
        FullValueDescription describe_value();
        ValueMemberDef create_value_member(
            in RepositoryId id,
            in Identifier name,
            in VersionSpec version,
            in IDLType type,
            in Visibility access
        );
        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
        );
    }; // End ValueDef Interface
```

The inherited describe() and contents() operations are also described.

See Also CORBA::Container::create_value()

ValueDef::abstract_base_values Attribute

```
// IDL
attribute ValueDefSeq abstract_base_values;
```

The abstract_base_values attribute lists the abstract value types from which this value inherits.

Exceptions

BAD_PARAM, minor code 5	The name attribute of any object contained by this ValueDef conflicts with the name attribute of any object contained by any of the specified bases.

ValueDef::base_value Attribute

```
// IDL
attribute ValueDef base_value;
```

The base_value attribute describes the value type from which this value inherits.

Parameters

BAD_PARAM, minor code 5	The name attribute of any object contained by the minor code 5 is raised if the name attribute of any object contained by this ValueDef conflicts with the name attribute of any object contained by any of the specified bases.

ValueDef::contents()

```
// IDL
ContainedSeq contents(
    in DefinitionKind limit_type,
    in boolean exclude_inherited
);
```

Inherited from Container, contents() returns the list of constants, types, and exceptions defined in this ValueDef and the list of attributes, operations, and members either defined or inherited in this ValueDef.

Parameters

`limit_type`	If set to `dk_all`, all of the contained objects in the `ValueDef` are returned. If set to the `DefinitionKind` for a specific interface type, it returns only interfaces of that type. For example, if set to, `dk_Operation`, then it returns contained operations only.
`exclude_inherited`	Applies only to interfaces. If true, only attributes, operations and members defined within this value type are returned. If false, all attributes, operations and members are returned.

See Also `CORBA::Container::contents()`

ValueDef::create_attribute()

```
// IDL
AttributeDef create_attribute(
    in RepositoryId id,
    in Identifier name,
    in VersionSpec version,
    in IDLType type,
    in AttributeMode mode
);
```

Returns a new `AttributeDef` object contained in the `ValueDef` on which it is invoked.

Parameters

`id`	The repository ID to use for the new `AttributeDef`. An `AttributeDef` inherits the `id` attribute from `Contained`.
`name`	The `name` to use for the new `AttributeDef`. An `AttributeDef` inherits the `name` attribute from `Contained`.
`version`	The `version` to use for the new `AttributeDef`. An `AttributeDef` inherits the `version` attribute from `Contained`.
`type`	The IDL data type for the new `AttributeDef`. Both the `type_def` and `type` attributes are set for `AttributeDef`.
`mode`	The read or read/write `mode` to use for the new `AttributeDef`.

The defined_in attribute (which the AttributeDef inherits from Contained) is initialized to identify the containing ValueDef.

Exceptions

BAD_PARAM, minor code 5	The name attribute of any object contained by minor code 2 is raised if an object with the specified id already exists in the repository.
BAD_PARAM, minor code 3	An object with the same name already exists in this ValueDef.

See Also

CORBA::AttributeDef
CORBA::Contained

ValueDef::create_operation()

```
// IDL
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
);
```

Returns a new OperationDef object contained in the ValueDef on which it is invoked.

Parameters

id	The repository ID to use for the new OperationDef. An OperationDef inherits the id attribute from Contained.
name	The name to use for the new OperationDef. An OperationDef inherits the name attribute from Contained.
version	The version to use for the new OperationDef. An OperationDef inherits the version attribute from Contained.

`result`	The IDL data type of the return value for the new `OperationDef`. Both the `result_def` and `result` attributes are set for the `OperationDef`.
`mode`	The `mode` to use for the new `OperationDef`. Specifies whether the operation is normal (`OP_NORMAL`) or oneway (`OP_ONEWAY`).
`params`	The parameters for this `OperationDef`.
`exceptions`	The list of exceptions to use for the `OperationDef`. These are exceptions the operation can raise.
`contexts`	The list of context identifiers to use for the `OperationDef`. These represent the context clause of the operation.

The `defined_in` attribute (which the `OperationDef` inherits from `Contained`) is initialized to identify the containing `ValueDef`.

Exceptions

`BAD_PARAM`, minor code `5`	The `name` attribute of any object contained by minor code 2 is raised if an object with the specified `id` already exists in the repository.
`BAD_PARAM`, minor code 3	An object with the same `name` already exists in this `ValueDef`.

See Also

`CORBA::OperationDef`
`CORBA::Contained`

ValueDef::create_value_member()

```
// IDL
ValueMemberDef create_value_member(
    in RepositoryId id,
    in Identifier name,
    in VersionSpec version,
    in IDLType type,
    in Visibility access
);
```

Returns a new `ValueMemberDef` contained in the `ValueDef` on which it is invoked.

Parameters

`id`	The repository ID to use for the new `ValueMemberDef`. An `ValueMemberDef` inherits the `id` attribute from `Contained`.
`name`	The `name` to use for the new `ValueMemberDef`. An `ValueMemberDef` inherits the `name` attribute from `Contained`.
`version`	The `version` to use for the new `ValueMemberDef`. An `ValueMemberDef` inherits the `version` attribute from `Contained`.
`type`	The IDL data type for the new `ValueMemberDef`. Both the `type_def` and `type` attributes are set for `ValueMemberDef`.
`access`	The visibility to use for the new `ValueMemberDef`. IDL value types can have state members that are either public or private.

The `defined_in` attribute (which the `ValueMemberDef` inherits from `Contained`) is initialized to identify the containing `ValueDef`.

Exceptions

`BAD_PARAM`, minor code `5`	The `name` attribute of any object contained by minor code 2 is raised if an object with the specified `id` already exists in the repository.
A `BAD_PARAM`, minor code `3`	An object with the same `name` already exists in this `ValueDef`.

See Also

`CORBA::ValueMemberDef`
`CORBA::Contained`

ValueDef::describe()

```
// IDL
ValueDescription describe();
```

Inherited from `Contained`, `describe()` for a `ValueDef` returns a `ValueDescription` object. Use `describe_value()` for a full description of the value.

See Also

`CORBA::ValueDescription`
`CORBA::Contained::describe()`
`CORBA::ValueDef::describe_value()`

ValueDef::describe_value()

```
// IDL
FullValueDescription describe_value();
```

Returns a FullValueDescription object describing the value, including its operations and attributes.

See Also

CORBA::FullValueDescription
CORBA::ValueDef::describe()

ValueDef::FullValueDescription Structure

```
// IDL
struct FullValueDescription {
    Identifier name;
    RepositoryId id;
    boolean is_abstract;
    boolean is_custom;
    RepositoryId defined_in;
    VersionSpec version;
    OpDescriptionSeq operations;
    AttrDescriptionSeq attributes;
    ValueMemberSeq members;
    InitializerSeq initializers;
    RepositoryIdSeq supported_interfaces;
    RepositoryIdSeq abstract_base_values;
    boolean is_truncatable;
    RepositoryId base_value;
    TypeCode type;
};
```

A full description of a value type in the interface repository.

name	The name of the value type.
id	The repository ID of the value type.
is_abstract	Has a value of 1 (true) if the value is an abstract value type. A value of 0 is false.
is_custom	Has a value of 1 (true) if the value uses custom marshalling. A value of 0 is false.

`defined_in`	The repository ID that identifies where this value type is defined.
`version`	The version of the value type.
`operations`	A list of operations that the value type supports.
`attributes`	A list of attributes that the value type supports.
`members`	A list of value type members.
`initializers`	A list of initializer values for the value type.
`supported_interfaces`	A list of interfaces this value type supports.
`abstract_base_values`	A list of repository IDs that identify abstract base values.
`is_truncatable`	Has a value of 1 (true) if the value type is truncatable. A value of 0 is false.
`base_value`	A repository ID that identifies a base value.
`type`	The IDL type of the value type.

See Also `CORBA::ValueDef::describe_value()`

ValueDef::initializers Attribute

```
// IDL
attribute InitializerSeq initializers;
```

Lists the initializers this value type supports.

ValueDef::is_a()

```
// IDL
boolean is_a(
    in RepositoryId id
);
```

Returns 1 (true) if this value type is either identical to or inherits, directly or indirectly, from the interface or value identified by the `id` parameter. Otherwise it returns 0 (false).

Parameters

`id`	The repository ID of the value type or interface to compare with this value type.

ValueDef::is_abstract Attribute

```
// IDL
attribute boolean is_abstract;
```

Returns 1 (true) if this value type is an abstract value type. Otherwise it returns 0 (false).

ValueDef::is_custom Attribute

```
// IDL
attribute boolean is_custom;
```

Returns 1 (true) if this value type uses custom marshalling. Otherwise it returns 0 (false).

ValueDef::is_truncatable Attribute

```
// IDL
attribute boolean is_truncatable;
```

Returns 1 (true) if this value type inherits safely (supports truncation) from another value. Otherwise it returns 0 (false).

ValueDef::supported_interfaces Attribute

```
// IDL
attribute InterfaceDef supported_interfaces;
```

Lists the interfaces that this value type supports.

Exceptions

BAD_PARAM, minor code 5	The name attribute of any object contained by the minor code 5 is raised if the name attribute of any object contained by this ValueDef conflicts with the name attribute of any object contained by any of the specified bases.

CORBA::ValueFactory

This describes the mapping of the IDL native type `CORBA::ValueFactory`. For native IDL types, each language mapping specifies how repository IDs are used to find the appropriate factory for an instance of a value type so that it may be created as it is unmarshaled off the wire.

```
// IDL in module CORBA
native ValueFactory;
```

Recall that value types allow objects to be passed by value which implies that the ORB must be able to create instances of your value type classes during unmarshaling. However, because the ORB cannot know about all potential value type classes, you must implement factory classes for those types and register them with the ORB so the ORB can create value instances when necessary.

The C++ mapping for the IDL `CORBA::ValueFactory` native type includes the following:

- The `ValueFactory` type which is a pointer to a `ValueFactoryBase` class.
- The `ValueFactoryBase` class which is is the base class for all value type factory classes.

Just as your applications must provide concrete value type classes (see `CORBA::ValueBase`), your applications must also provide factory classes for those concrete classes.

If the ORB is unable to locate and use the appropriate factory, then a `MARSHAL` exception with a minor code is raised.

CORBA::ValueFactory Type

```
// C++ in namespace CORBA
typedef ValueFactoryBase* ValueFactory;
```

The `ValueFactory` is a pointer to a `ValueFactoryBase` class. Applications derive concrete factory classes from `ValueFactoryBase`, and register instances of those factory classes with the ORB via `ORB::register_value_factory()`.

See Also

CORBA::ValueFactoryBase
CORBA::ORB::lookup_value_factory()
CORBA::ORB::register_value_factory()
CORBA::ORB::unregister_value_factory()

CORBA::ValueFactoryBase Class

When unmarshaling value instances, the ORB needs to be able to call up to the application to ask it to create those instances. Value instances are normally created via their type-specific value factories so as to preserve any invariants they might have for their state. However, creation for unmarshaling is different because the ORB has no knowledge of application-specific factories, and in fact in most cases may not even have the necessary arguments to provide to the type-specific factories.

To allow the ORB to create value instances required during unmarshaling, the `ValueFactoryBase` class provides the private `create_for_unmarshal()` pure virtual function. The function is private so that only the ORB, can invoke it. Your applications do not invoke `create_for_unmarshal()`, however, your derived classes must override `create_for_unmarshal()` and implement it such that it creates a new value instance and returns a pointer to the instance. The caller (in this case the ORB) assumes ownership of the returned instance. Once the ORB has created a value instance via the `create_for_unmarshal()` function, it uses the value data member modifier functions to set the state of the new value instance from the unmarshaled data.

```
// C++ in namespace CORBA
    class ValueFactoryBase {
    public:
        virtual ~ValueFactoryBase();
        virtual void _add_ref();
        virtual void _remove_ref();
        static ValueFactory _downcast(ValueFactory vf);
    protected:
        ValueFactoryBase();
    private:
        virtual ValueBase* create_for_unmarshal() = 0;
        ...
    };
```

See Also CORBA::ValueBase

ValueFactoryBase::_add_ref()

```
virtual void _add_ref();
```

Increases this object factory's reference count by one. The ValueFactoryBase uses reference counting to prevent itself from being destroyed while still in use by the application. A ValueFactoryBase object initially has a reference count of one.

See Also CORBA::ValueFactoryBase::_remove_ref()

ValueFactoryBase::_downcast()

```
static ValueFactory _downcast(
    ValueFactory vf
);
```

Returns a pointer to the type-specific factory object.

Parameters

vf	The original value factory object.

You can use _downcast() on the return type of the function ORB::lookup_value_factory() to obtain a pointer to a type-specific factory object. Memory management of the return value from _downcast() is *not* the responsibility of the caller, and thus you should not call _remove_ref() on it.

See Also CORBA::ORB::lookup_value_factory()
CORBA::ValueFactoryBase::_remove_ref()

ValueFactoryBase::_remove_ref()

```
virtual void _remove_ref();
```

Decreases this object factory's reference count by one, and if the resulting reference count equals zero, the object factory is destroyed.

See Also CORBA::ValueFactoryBase::_add_ref()

ValueFactoryBase::~ValueFactoryBase() Destructor

```
virtual ~ValueFactoryBase();
```

The default destructor.

See Also CORBA::ValueFactoryBase::ValueFactoryBase()

ValueFactoryBase::ValueFactoryBase() Constructor

```
protected:
    ValueFactoryBase();
```

The default constructor.

See Also CORBA::ValueFactoryBase::~ValueFactoryBase()

CORBA::ValueMemberDef Interface

The `ValueMemberDef` interface provides the definition of a value type member in the interface repository.

```
// IDL in module CORBA.
interface ValueMemberDef : Contained {
    readonly attribute TypeCode type;
    attribute IDLType type_def;
    attribute Visibility access;
};
```

ValueMemberDef::access Attribute

```
// IDL
attribute Visibility access;
```

Contains an indicator of the visibility of an IDL value type state member. IDL value types can have state members that are either public or private.

ValueMemberDef::type Attribute

```
// IDL
readonly attribute TypeCode type;
```

Describes the type of this `ValueMemberDef`.

See Also `CORBA::ValueMemberDef::type_def`

ValueMemberDef::type_def Attribute

```
// IDL
attribute IDLType type_def;
```

Identifies the object that defines the IDL type of this `ValueMemberDef`. The same information is contained in the `type` attribute.

You can change the type of a `ValueMemberDef` by changing its `type_def` attribute. This also changes its `type` attribute.

See Also `CORBA::ValueMemberDef::type`

CORBA::WstringDef Interface

Interface `WstringDef` represents a bounded IDL wide string type in the interface repository. A `WstringDef` object is anonymous, which means it is unnamed. Use `Repository::create_wstring()` to obtain a new `WstringDef` object.

Unbounded strings are primitive types represented with the `PrimitiveDef` interface. Use `Repository::get_primitive()` to obtain unbounded wide strings.

```
// IDL in module CORBA.
interface WstringDef : IDLType {
    attribute unsigned long bound;
};
```

The inherited `type` attribute is also described.

See Also

`CORBA::IDLType`
`CORBA::Repository::create_wstring()`
`CORBA::PrimitiveDef`
`CORBA::StringDef`

WstringDef::bound Attribute

```
// IDL
attribute unsigned long bound;
```

Specifies the maximum number of characters in the wide string. This cannot be zero.

WstringDef::type Attribute

```
// IDL
readonly attribute TypeCode type;
```

The `type` attribute is inherited from interface `IDLType`. This attribute is a `tk_wstring` `TypeCode` that describes the wide string.

See Also CORBA::IDLType::type

CORBA::WString_var Class

The class `WString_var` implements the `_var` type for IDL wide strings required by the standard C++ mapping. The `WString_var` class contains a `char*` value and ensures that this is properly freed when a `WString_var` object is deallocated, for example when exectution goes out of scope.

```
class WString_var {
public:
    WString_var();
    WString_var(char *p);
    WString_var(const char *p);
    WString_var(const WString_var &s);
    ~WString_var();
    WString_var & operator=(char *p);
    WString_var & operator=(const char *p);
    WString_var & operator=(const WString_var &s);
    operator char*();
    operator const char*() const;
    const char* in() const;
    char*& inout();
    char*& out();
    char* _retn();
    char & operator[](ULong index);
    char operator[](ULong index) const;
};
```

WString_var::char*()

```
operator char*();
operator const char*() const;
```

Converts a `WString_var` object to a `char*`.

See Also CORBA::WString_var::operator=()

WString_var::in()

```
const char* in() const;
```

Returns the proper string for use as an input parameter.

See Also

CORBA::WString_var::out()
CORBA::WString_var::inout()
CORBA::WString_var::_retn()

WString_var::inout()

```
char*& inout();
```

Returns the proper string for use as an inout parameter.

See Also

CORBA::WString_var::in()
CORBA::WString_var::out()
CORBA::WString_var::_retn()

WString_var::operator=() Assignment Operators

```
WString_var &operator=(
    char *p
);

WString_var &operator=(
    const char *p
);

WString_var &operator=(
    const WString_var &s
);
```

Assignment operators allow you to assign values to a `WString_var` from a `char*` or from another `WString_var` type.

Parameters

`p`	A character string to assign to the `WString_var`.
`s`	A `WString_var` to assign to the `WString_var`.

See Also

CORBA::WString_var::char*()

WString_var::operator[]() Subscript Operators

```
char &operator[](
    ULong index
);

char operator[](
    ULong index
) const;
```

Return the character at the given location of the string. Subscript operators allow access to the individual characters in the string.

Parameters

index	The index location in the string.

WString_var::out()

```
char*& out();
```

Returns the proper string for use as an output parameter.

See Also

CORBA::WString_var::in()
CORBA::WString_var::inout()
CORBA::WString_var::_retn()

WString_var::WString_var() Constructors

```
WString_var();
```

The default constructor.

```
WString_var(
    char *p
);

WString_var(
    const char *p
);
```

Constructors that convert from a char* to a WString_var.

```
WString_var(
    const WString_var &s
);
```

The copy constructor.

Parameters

p	The character string to convert to a `WString_var`. The `WString_var` assumes ownership of the parameter.
s	The original `WString_var` that is copied.

See Also CORBA::WString_var::~WString_var()

WString_var::~WString_var() Destructor

```
~WString_var();
```

The destructor.

See Also CORBA::WString_var::WString_var()

WString_var::_retn()

```
char* _retn();
```

Returns the proper string for use as a method's return value.

See Also
CORBA::WString_var::inout()
CORBA::WString_var::in()
CORBA::WString_var::out()

IT_CORBA Module

IT_CORBA Overview

This module contains Orbix enhancements to the CORBA module. The key additional feature is the policy WellKnownAddressingPolicy. The classes include:

- RefCountedLocalObject
- RefCountedLocalObjectNC
- WellKnownAddressingPolicy

The IDL code is as follows:

IT_CORBA::WELL_KNOWN_ADDRESSING_POLICY_ID Constant

```
// IDL in module IT_CORBA
const CORBA::PolicyType WELL_KNOWN_ADDRESSING_POLICY_ID =
    IT_PolicyBase::IONA_POLICY_ID + 2;

// C++ in namespace IT_CORBA
IT_ART_API IT_NAMESPACE_STATIC
    const CORBA::ULong WELL_KNOWN_ADDRESSING_POLICY_ID;
```

Defines a policy ID for well-known addressing.

Enhancement This is an Orbix enhancement to CORBA.

See Also CORBA::PolicyType

IT_CORBA::RefCountedLocalObject Class

RefCountedLocalObject is an implementation of a CORBA local object that automatically handles reference counting in a thread safe manner.

```
// in namespace IT_CORBA
...
class IT_ART_API RefCountedLocalObject :
public CORBA::LocalObject {
  public:

    RefCountedLocalObject();

    void _add_ref();

    void _remove_ref();

  protected:
    virtual void _destroy_this();

  private:
    ...
};
```

See Also IT_CORBA::RefCountedLocalObjectNC

RefCountedLocalObject::_add_ref()

```
void _add_ref();
```

Increments the reference count.

Enhancement This is an Orbix enhancement to CORBA.

RefCountedLocalObject::_destroy_this()

```
virtual void _destroy_this();
```

Destroys the local object.

Enhancement This is an Orbix enhancement to CORBA.

RefCountedLocalObject::RefCountedLocalObject() Constructor

```
RefCountedLocalObject();
```

The constructor.

Enhancement This is an Orbix enhancement to CORBA.

RefCountedLocalObject::_remove_ref()

```
void _remove_ref()
```

Decrements the reference count.

Enhancement This is an Orbix enhancement to CORBA.

IT_CORBA::
RefCountedLocalObjectNC Class

`RefCountedLocalObjectNC` is an implementation of a `CORBA` local object that automatically handles reference counting but not in a thread-safe manner as the `RefCountedLocalObject` class does. A `RefCountedLocalObjectNC` object does not protect its reference count with a mutex, making it suitable for lightweight objects such as `CORBA::Request`.

```
// in namespace IT_CORBA
...
class IT_ART_API RefCountedLocalObjectNC :
public CORBA::LocalObject {
  public:
    RefCountedLocalObjectNC();

    void _add_ref();

    void _remove_ref();

  protected:
    virtual void _destroy_this();

  private:
    ...
}
```

See Also `IT_CORBA::RefCountedLocalObject`

RefCountedLocalObjectNC::_add_ref()

```
void _add_ref();
```

Increments the reference count.

Enhancement This is an Orbix enhancement to CORBA.

RefCountedLocalObjectNC::_destroy_this()

```
virtual void _destroy_this();
```

Destroys the local object.

Enhancement This is an Orbix enhancement to CORBA.

RefCountedLocalObjectNC::RefCountedLocalObjectNC() Constructor

```
RefCountedLocalObjectNC();
```

The constructor.

Enhancement This is an Orbix enhancement to CORBA.

RefCountedLocalObjectNC::_remove_ref()

```
void _remove_ref();
```

Decrements the reference count.

Enhancement This is an Orbix enhancement to CORBA.

IT_CORBA::
WellKnownAddressingPolicy Class

This is an interface for a local policy object derived from CORBA::Policy. You create instances of WellKnownAddressingPolicy by calling CORBA::ORB::create_policy().

```
// in namespace IT_CORBA
...
class IT_ART_API WellKnownAddressingPolicy :
public virtual ::CORBA::Policy {
  public:

    typedef IT_CORBA::WellKnownAddressingPolicy_ptr _ptr_type;
    typedef IT_CORBA::WellKnownAddressingPolicy_var _var_type;

    virtual ~WellKnownAddressingPolicy();

    static WellKnownAddressingPolicy_ptr _narrow(
        CORBA::Object_ptr obj
    );

    static WellKnownAddressingPolicy_ptr _unchecked_narrow(
        CORBA::Object_ptr obj
    );

    inline static WellKnownAddressingPolicy_ptr _duplicate(
        WellKnownAddressingPolicy_ptr p
    );

    inline static WellKnownAddressingPolicy_ptr _nil();

    virtual char* config_scope() = 0;

    static const IT_FWString _it_fw_type_id;
};
```

See page 5 for descriptions of the standard helper functions:

- `_duplicate()`
- `_narrow()`
- `_nil()`
- `_unchecked_narrow()`

WellKnownAddressingPolicy::config_scope()

```
// C++
virtual char* config_scope() = 0;
```

Returns the configuration scope.

Enhancement This is an Orbix enhancement to CORBA.

WellKnownAddressingPolicy::~WellKnownAddressingPolicy() Destructor

```
virtual ~WellKnownAddressingPolicy();
```

The destructor for this policy object.

Enhancement This is an Orbix enhancement to CORBA.

DynamicAny Module

DynamicAny Overview

The DynamicAny namespace implements the IDL DynamicAny module which includes the following classes:

DynAny
DynAnyFactory
DynArray
DynEnum
DynFixed
DynSequence
DynStruct
DynUnion
DynValue

The common data types in the scope of the DynamicAny module include the following:

AnySeq
DynAnySeq
FieldName
NameDynAnyPair
NameDynAnyPairSeq
NameValuePair
NameValuePairSeq

For most IDL data types there is a straight-forward language mapping that an object implementation uses to interpret data. However, an any data type can be passed to a program that may not have any static information about how to interpret the type of data in the any value. The DynamicAny module provides a runtime mechanism for constructing any values, traversing them, and extracting the data from any values. This mechanism is especially helpful for writing generic clients and servers such as bridges, browsers, debuggers, and user interface tools.

Applications dynamically construct and interpret any values using DynAny objects. For complex any types a DynAny object is an ordered collection of other component DynAny objects.

A DynAny object can be created as follows:

- Invoking a method on a DynAnyFactory object.
- Invoking a method on an existing DynAny object.

  A constructed DynAny object supports methods that enable the creation of new DynAny objects that encapsulate access to the value of some constituent of the DynAny object. DynAny objects also support a copy method for creating new DynAny objects.

There is a different interface associated with each kind of constructed IDL type that inherits from the DynAny interface. The interfaces that inherit the DynAny interface include:

DynArray
DynEnum
DynFixed
DynSequence
DynStruct
DynUnion
DynValue

Exceptions are represented by the DynStruct interface and value types are represented by the DynValue interface.

DynamicAny::AnySeq Sequence

```
class AnySeq: private ITCxxUSeq< CORBA::Any > {
public:
    typedef AnySeq_var _var_type;

    AnySeq(
        CORBA::ULong max,
        CORBA::ULong length,
        CORBA::Any* buf,
        CORBA::Boolean release = 0
    ) : ITCxxUSeq< CORBA::Any >(max, length, buf, release) {}
    AnySeq() : ITCxxUSeq< CORBA::Any >() {}

    AnySeq(
        CORBA::ULong max
    ) : ITCxxUSeq< CORBA::Any >(max) {}

    AnySeq(
```

```
        const AnySeq& seq
    ) : ITCxxUSeq< CORBA::Any >(seq) {}

    AnySeq& operator=(
        const AnySeq& seq
    )
    {
        ITCxxUSeq< CORBA::Any >::operator=(seq);
        return *this;
    }

    ITCxxUSeq< CORBA::Any >::maximum;
    ITCxxUSeq< CORBA::Any >::length;
    ITCxxUSeq< CORBA::Any >::operator[];
    ITCxxUSeq< CORBA::Any >::replace;
    ITCxxUSeq< CORBA::Any >::get_buffer;
    ITCxxUSeq< CORBA::Any >::allocbuf;
    ITCxxUSeq< CORBA::Any >::freebuf;

    ITCxxUSeq< CORBA::Any >::operator new;
    ITCxxUSeq< CORBA::Any >::operator delete;
};
```

A sequence of CORBA::Any values.

See Also

DynamicAny::DynSequence
DynamicAny::DynArray

"About Sequences"

DynamicAny::DynAnySeq Sequence

```
class DynAnySeq:
    private ITCxxUIntfSeq< DynAny_ptr,
    DynAny, ITCxxIntfAlloc< DynAny_ptr, DynAny > > {
public:
    typedef DynAnySeq_var _var_type;

    DynAnySeq(
        CORBA::ULong max,
        CORBA::ULong length,
        DynAny_ptr* buf,
        CORBA::Boolean release = 0
```

```
 ) : ITCxxUIntfSeq< DynAny_ptr, DynAny, ITCxxIntfAlloc<
DynAny_ptr, DynAny > >(max, length, buf, release) {}

 DynAnySeq() : ITCxxUIntfSeq< DynAny_ptr, DynAny,
ITCxxIntfAlloc< DynAny_ptr, DynAny > >() {}

 DynAnySeq(
     CORBA::ULong max
 ) : ITCxxUIntfSeq< DynAny_ptr, DynAny, ITCxxIntfAlloc<
DynAny_ptr, DynAny > >(max) {}

 DynAnySeq(
     const DynAnySeq& seq
 ) : ITCxxUIntfSeq< DynAny_ptr, DynAny, ITCxxIntfAlloc<
DynAny_ptr, DynAny > >(seq) {}

 DynAnySeq& operator=(
     const DynAnySeq& seq
 )
 {
     ITCxxUIntfSeq< DynAny_ptr, DynAny, ITCxxIntfAlloc<
DynAny_ptr, DynAny > >::operator=(seq);
     return *this;
 }

 ITCxxUIntfSeq< DynAny_ptr, DynAny, ITCxxIntfAlloc< DynAny_ptr,
DynAny > >::maximum;
 ITCxxUIntfSeq< DynAny_ptr, DynAny, ITCxxIntfAlloc< DynAny_ptr,
DynAny > >::length;
 ITCxxUIntfSeq< DynAny_ptr, DynAny, ITCxxIntfAlloc< DynAny_ptr,
DynAny > >::operator[];
 ITCxxUIntfSeq< DynAny_ptr, DynAny, ITCxxIntfAlloc< DynAny_ptr,
DynAny > >::replace;
 ITCxxUIntfSeq< DynAny_ptr, DynAny, ITCxxIntfAlloc< DynAny_ptr,
DynAny > >::get_buffer;
 ITCxxUIntfSeq< DynAny_ptr, DynAny, ITCxxIntfAlloc< DynAny_ptr,
DynAny > >::allocbuf;
 ITCxxUIntfSeq< DynAny_ptr, DynAny, ITCxxIntfAlloc< DynAny_ptr,
DynAny > >::freebuf;

 ITCxxUIntfSeq< DynAny_ptr, DynAny, ITCxxIntfAlloc< DynAny_ptr,
DynAny > >::operator new;
```

```
    ITCxxUIntfSeq< DynAny_ptr, DynAny, ITCxxIntfAlloc< DynAny_ptr,
   DynAny > >::operator delete;
};
```

A sequence of DynAny values.

See Also DynamicAny::DynSequence
DynamicAny::DynArray

"About Sequences"

DynamicAny::FieldName Type

```
// IDL
typedef string FieldName;
```

```
// C++
typedef char* FieldName;
```

A string representing the name of a member in a structure, union, or value type.

See Also DynamicAny::DynStruct
DynamicAny::DynUnion
DynamicAny::DynValue

DynamicAny::NameDynAnyPair Structure

```
// IDL
struct NameDynAnyPair {
    FieldName id;
    DynAny value;
};
```

```
struct NameDynAnyPair {
    typedef NameDynAnyPair_var _var_type;
    ITGenFieldName_mgr id;
    ITGenDynAny_mgr value;
};
```

A structure containing the name and value of a field or member.

See Also DynamicAny::NameDynAnyPairSeq

DynamicAny::NameDynAnyPairSeq Sequence

```
class NameDynAnyPairSeq: private ITCxxUSeq< NameDynAnyPair > {
public:
    typedef NameDynAnyPairSeq_var _var_type;

    NameDynAnyPairSeq(
        CORBA::ULong max,
        CORBA::ULong length,
        NameDynAnyPair* buf,
        CORBA::Boolean release = 0
    ) : ITCxxUSeq< NameDynAnyPair >(max, length, buf, release) {}
    NameDynAnyPairSeq() : ITCxxUSeq< NameDynAnyPair >() {}

    NameDynAnyPairSeq(
        CORBA::ULong max
    ) : ITCxxUSeq< NameDynAnyPair >(max) {}

    NameDynAnyPairSeq(
        const NameDynAnyPairSeq& seq
    ) : ITCxxUSeq< NameDynAnyPair >(seq) {}

    NameDynAnyPairSeq& operator=(
        const NameDynAnyPairSeq& seq
    )
    {
        ITCxxUSeq< NameDynAnyPair >::operator=(seq);
        return *this;
    }

    ITCxxUSeq< NameDynAnyPair >::maximum;
    ITCxxUSeq< NameDynAnyPair >::length;
    ITCxxUSeq< NameDynAnyPair >::operator[];
    ITCxxUSeq< NameDynAnyPair >::replace;
    ITCxxUSeq< NameDynAnyPair >::get_buffer;
    ITCxxUSeq< NameDynAnyPair >::allocbuf;
    ITCxxUSeq< NameDynAnyPair >::freebuf;

    ITCxxUSeq< NameDynAnyPair >::operator new;
    ITCxxUSeq< NameDynAnyPair >::operator delete;
};
```

A sequence of NameDynAnyPair structures.

See Also DynamicAny::DynStruct
DynamicAny::DynValue

"About Sequences"

DynamicAny::NameValuePair Structure

```
struct NameValuePair {
    typedef NameValuePair_var _var_type;
    ITGenFieldName_mgr id;
    CORBA::Any value;
};
```

A structure containing the name and value of a field or member.

See Also DynamicAny::NameValuePairSeq

DynamicAny::NameValuePairSeq Sequence

```
class NameValuePairSeq: private ITCxxUSeq< NameValuePair > {
public:
    typedef NameValuePairSeq_var _var_type;

    NameValuePairSeq(
        CORBA::ULong max,
        CORBA::ULong length,
        NameValuePair* buf,
        CORBA::Boolean release = 0
    ) : ITCxxUSeq< NameValuePair >(max, length, buf, release) {}
    NameValuePairSeq() : ITCxxUSeq< NameValuePair >() {}

    NameValuePairSeq(
        CORBA::ULong max
    ) : ITCxxUSeq< NameValuePair >(max) {}

    NameValuePairSeq(
        const NameValuePairSeq& seq
    ) : ITCxxUSeq< NameValuePair >(seq) {}

    NameValuePairSeq& operator=(
        const NameValuePairSeq& seq
    )
```

```
    {
        ITCxxUSeq< NameValuePair >::operator=(seq);
        return *this;
    }

    ITCxxUSeq< NameValuePair >::maximum;
    ITCxxUSeq< NameValuePair >::length;
    ITCxxUSeq< NameValuePair >::operator[];
    ITCxxUSeq< NameValuePair >::replace;
    ITCxxUSeq< NameValuePair >::get_buffer;
    ITCxxUSeq< NameValuePair >::allocbuf;
    ITCxxUSeq< NameValuePair >::freebuf;

    ITCxxUSeq< NameValuePair >::operator new;
    ITCxxUSeq< NameValuePair >::operator delete;
};
```

A sequence of NameValuePair structures.

See Also

DynamicAny::DynStruct
DynamicAny::DynValue

"About Sequences"

DynamicAny::DynAny Class

Your application can dynamically construct and interpreted Any values using DynAny objects. A DynAny object is associated with a data value which corresponds to a copy of the value inserted into an any. Portable programs should use the DynAny interface to access and modify the contents of an Any in those cases where basic insertion and extraction operators are not sufficient.

DynAny methods can be organized as follows:

Table 9: *DynAny Methods*

General Methods	Insert Methods	Get Methods
assign()	insert_any()	get_any()
component_count()	insert_boolean()	get_boolean()
copy()	insert_char()	get_char()
current_component()	insert_double()	get_double()
destroy()	insert_dyn_any()	get_dyn_any()
~DynAny()	insert_float()	get_float()
equal()	insert_long()	get_long()
from_any()	insert_longdouble()	get_longdouble()
next()	insert_longlong()	get_longlong()
rewind()	insert_octet()	get_octet()
seek()	insert_reference()	get_reference()
to_any()	insert_short()	get_short()
type()	insert_string()	get_string()
	insert_typecode()	get_typecode()
	insert_ulong()	get_ulong()
	insert_ulonglong()	get_ulonglong()
	insert_ushort()	get_ushort()
	insert_val()	get_val()
	insert_wchar()	get_wchar()
	insert_wstring()	get_wstring()

The following exceptions are also defined in the DynAny class:

InvalidValue

TypeMismatch

The DynAny class is the base for the following classes:

DynArray
DynEnum
DynFixed
DynSequence
DynStruct
DynUnion
DynValue

Because the values of Any types can be quite complex, it is helpful to think of a DynAny object as an ordered collection of other *component* DynAny objects. For simpler DynAny objects that represent a basic type, the ordered collection of components is empty. For example, a long or a type without components (such as an empty exception) has empty components.

The DynAny interface allows a client to iterate through the components of the values pointed to by these objects. Each DynAny object maintains the notion of a *current position* into its collection of component DynAny objects. The current position is identified by an index value that runs from 0 to *n*-1, where *n* is the number of components. Methods are available that allow you to recursively examine DynAny contents. For example, you can determine the current position using current_component(), and component_count() returns the number of components in the DynAny object. You can also use rewind(), seek(), and next() to change the current position. If a DynAny is initialized with a value that has components, the index is initialized to 0. The special index value of -1 indicates a current position that points nowhere. For example, some values (such as an empty exception) cannot have a current position. In these cases the index value is fixed at -1.

You can use the iteration operations, together with current_component(), to dynamically compose an Any value. After creating a dynamic any, such as a DynStruct, you can use current_component() and next() to initialize all the components of the value. Once the dynamic value is completely initialized, to_any() creates the corresponding Any value.

You use the insert_*type*() and get_*type*() methods to not only handle basic DynAny objects but they are also helpful in handling constructed DynAny objects. when you insert a basic data type value into a constructed DynAny object, it initializes the current component of the constructed data value associated with the DynAny object.

For example, invoking insert_boolean() on a DynStruct object implies inserting a boolean data value at the current position of the associated structure data value. In addition, you can use the insert_*type*() and get_*type*() methods to traverse Any values associated with sequences of basic data types without the need to generate a DynAny object for each element in the sequence.

The DynAny object has a destroy() method that you can use to destroy a top-level DynAny object and any component DynAny objects obtained from it.

Exceptions

TypeMismatch is raised if you call methods insert_type() or get_type() on a DynAny whose current component itself has components.

MARSHAL is raised if you attempt to export DynAny objects to other processes or externalize one with CORBA::ORB::object_to_string(). This is because DynAny objects are intended to be local to the process in which they are created and used.

NO_IMPLEMENT might be raised if you attempt the following:

- Invoke operations exported through the CORBA::Object interface even though DynAny objects export operations defined in this standard interface.
- Use a DynAny object with the DII.

The following code is the complete class:

```
// class is in namespace DynamicAny
class IT_DYNANY_API DynAny : public virtual CORBA::Object {
public:
    typedef DynamicAny::DynAny_ptr _ptr_type;
    typedef DynamicAny::DynAny_var _var_type;

    virtual ~DynAny();

    static DynAny_ptr _narrow(
        CORBA::Object_ptr obj
    );
    static DynAny_ptr _unchecked_narrow(
        CORBA::Object_ptr obj
    );
    inline static DynAny_ptr _duplicate(
        DynAny_ptr p
    );
```

```
inline static DynAny_ptr _nil();

class IT_DYNANY_API InvalidValue: public CORBA::UserException
{ ... };

class IT_DYNANY_API TypeMismatch: public CORBA::UserException
{ ... };

virtual ::CORBA::TypeCode_ptr type() = 0;
virtual void assign(
    DynAny_ptr  dyn_any
) = 0;
virtual void from_any(
    const CORBA::Any&  value
) = 0;
virtual CORBA::Any* to_any() = 0;
virtual CORBA::Boolean equal(
    DynAny_ptr  dyn_any
) = 0;
virtual void destroy() = 0;
virtual DynAny_ptr copy() = 0;
virtual void insert_boolean(
    CORBA::Boolean value
) = 0;
virtual void insert_octet(
    CORBA::Octet value
) = 0;
virtual void insert_char(
    CORBA::Char value
) = 0;
virtual void insert_short(
    CORBA::Short value
) = 0;
virtual void insert_ushort(
    CORBA::UShort value
) = 0;
virtual void insert_long(
    CORBA::Long value
) = 0;
virtual void insert_ulong(
    CORBA::ULong value
) = 0;
virtual void insert_float(
```

```
    CORBA::Float value
) = 0;
virtual void insert_double(
    CORBA::Double value
) = 0;
virtual void insert_string(
    const char*  value
) = 0;
virtual void insert_reference(
    CORBA::Object_ptr  value
) = 0;
virtual void insert_typecode(
    ::CORBA::TypeCode_ptr  value
) = 0;
virtual void insert_longlong(
    CORBA::LongLong value
) = 0;
virtual void insert_ulonglong(
    CORBA::ULongLong value
) = 0;
virtual void insert_longdouble(
    CORBA::LongDouble value
) = 0;
virtual void insert_wchar(
    CORBA::WChar value
) = 0;
virtual void insert_wstring(
    const CORBA::WChar*  value
) = 0;
virtual void insert_any(
    const CORBA::Any&  value
) = 0;
virtual void insert_dyn_any(
    DynAny_ptr  value
) = 0;
virtual void insert_val(
    CORBA::ValueBase*  value
) = 0;
virtual CORBA::Boolean get_boolean() = 0;
virtual CORBA::Octet get_octet() = 0;
virtual CORBA::Char get_char() = 0;
virtual CORBA::Short get_short() = 0;
virtual CORBA::UShort get_ushort() = 0;
```

```
    virtual CORBA::Long get_long() = 0;
    virtual CORBA::ULong get_ulong() = 0;
    virtual CORBA::Float get_float() = 0;
    virtual CORBA::Double get_double() = 0;
    virtual char* get_string() = 0;
    virtual CORBA::Object_ptr get_reference() = 0;
    virtual ::CORBA::TypeCode_ptr get_typecode() = 0;
    virtual CORBA::LongLong get_longlong() = 0;
    virtual CORBA::ULongLong get_ulonglong() = 0;
    virtual CORBA::LongDouble get_longdouble() = 0;
    virtual CORBA::WChar get_wchar() = 0;
    virtual CORBA::WChar* get_wstring() = 0;
    virtual CORBA::Any* get_any() = 0;
    virtual DynAny_ptr get_dyn_any() = 0;
    virtual CORBA::ValueBase* get_val() = 0;

    virtual CORBA::Boolean seek(
        CORBA::Long index
    ) = 0;
    virtual void rewind() = 0;
    virtual CORBA::Boolean next() = 0;
    virtual CORBA::ULong component_count() = 0;
    virtual DynAny_ptr current_component() = 0;

    static const IT_FWString _it_fw_type_id;
};
```

See page 4 for descriptions of the standard helper functions:

- `_duplicate()`
- `_narrow()`
- `_nil()`
- `_unchecked_narrow()`

DynAny::assign()

```
// C++
virtual void assign(
    DynAny_ptr  dyn_any
) = 0;
```

Initializes the value associated with a DynAny object with the value associated with another DynAny object.

Parameters

dyn_any	The DynAny object to initialize to.

The current position of the target DynAny is set to zero for values that have components and to -1 for values that do not have components.

Exceptions

TypeMismatch	The type of the passed DynAny is not equivalent to the type of the target DynAny.

DynAny::component_count()

```
// C++
virtual CORBA::ULong component_count() = 0;
```

Returns the number of components of a DynAny. For a DynAny without components, it returns zero.

The operation only counts the components at the top level. For example, if you invoke component_count() on a DynStruct with a single member, the return value is 1, irrespective of the type of the member.

Table 10: *Return Values for DynAny::component_count()*

Type	Return Value
DynSequence	The current number of elements.
DynStruct DynValue	The number of members.

Table 10: *Return Values for DynAny::component_count()*

Type	Return Value
DynArray	The number of elements.
DynUnion	2 if the discriminator indicates that a named member is active. 1 Otherwise.
DynFixed DynEnum	zero

Exceptions

`TypeMismatch` The method is called on a `DynAny` that cannot have components, such as a `DynEnum` or an empty exception.

See Also

`DynamicAny::DynAny::current_component()`
`DynamicAny::DynAny::seek()`
`DynamicAny::DynAny::rewind()`
`DynamicAny::DynAny::next()`

DynAny::copy()

```
// C++
virtual DynAny_ptr copy() = 0;
```

Returns a new `DynAny` object whose value is a deep copy of the `DynAny` on which it is invoked.

The operation is polymorphic, that is, invoking it on one of the types derived from `DynAny`, such as `DynStruct`, creates the derived type but returns its reference as the `DynAny` base type.

DynAny::current_component()

```
// C++
virtual DynAny_ptr current_component() = 0;
```

Returns the DynAny for the component at the current position. It does not advance the current position, so repeated calls without an intervening call to `rewind()`, `next()`, or `seek()` return the same component. If the current position current position is `-1`, the method returns a nil reference.

The returned DynAny object reference can be used to get or set the value of the current component. If the current component represents a complex type, the returned reference can be narrowed based on the TypeCode to get the interface corresponding to the complex type.

Exceptions

`TypeMismatch`	The method is called on a DynAny that cannot have components, such as a `DynEnum` or an empty exception.

See Also

`DynamicAny::DynAny::component_count()`
`DynamicAny::DynAny::seek()`
`DynamicAny::DynAny::rewind()`
`DynamicAny::DynAny::next()`

DynAny::destroy()

```
// C++
virtual void destroy() = 0;
```

Destroys a DynAny object. This operation frees any resources used to represent the data value associated with a DynAny object.

Destroying a top-level DynAny object (one that was not obtained as a component of another DynAny) also destroys any component DynAny objects obtained from it. Destroying a non-top level (component) DynAny object does nothing.

You can manipulate a component of a DynAny object beyond the life time of its top-level DynAny by making a copy of the component with `copy()` before destroying the top-level DynAny object.

Enhancement Orbix guarantees to always destroy all local objects it creates when the last reference to them is released so you do not have to call `destroy()`. However, code that relies on this feature is not strictly CORBA compliant and may leak resources with other ORBs. (According to the CORBA specification, simply calling `CORBA::release()` on all references to a DynAny object does not delete the object or its components so each DynAny object created must be explicitly destroyed to avoid memory leaks.)

Exceptions

OBJECT_NOT_EXIST	A destroyed DynAny object or any of its components is referenced.

See Also

DynamicAny::DynAny::copy()
CORBA::release()
IT_CORBA::RefCountedLocalObject

DynAny::~DynAny() Destructor

```
// C++
virtual ~DynAny();
```

The destructor for a DynAny object.

DynAny::equal()

```
// C++
virtual CORBA::Boolean equal(
    DynAny_ptr  dyn_any
) = 0;
```

Compares two DynAny values for equality and returns true of the values are equal, false otherwise. Two DynAny values are equal if their type codes are equivalent and, recursively, all respective component DynAny values are equal. The current position of the two DynAny values being compared has no effect on the result of equal().

Parameters

dyn_any	The DynAny value to compare.

DynAny::from_any()

```
// C++
virtual void from_any(
    const CORBA::Any&  value
) = 0;
```

Initializes the value associated with a DynAny object with the value contained in an Any type.

The current position of the target DynAny is set to zero for values that have components and to -1 for values that do not have components.

Parameters

value	An Any value to initialize the DynAny object to.

Exceptions

TypeMismatch	The type of the passed Any is not equivalent to the type of the target DynAny.
InvalidValue	The passed Any does not contain a legal value (such as a null string).

See Also DynamicAny::DynAny::to_any()

DynAny::get_any()

```
// C++
virtual CORBA::Any* get_any() = 0;
```

Returns an Any value from the DynAny object.

It is valid for you to use this method if the TypeCode contained in the DynAny is equivalent to _tc_any (an Any TypeCode), or, if the TypeCode at the current position (a DynAny objects with components) is equivalent to _tc_any. The current position is unchanged after the call.

Exceptions

TypeMismatch	The accessed component in the DynAny is of a type that is not equivalent to the requested type.
InvalidValue	The DynAny has components and the current position is -1.

See Also DynamicAny::DynAny::insert_any()

DynAny::get_boolean()

```
// C++
virtual CORBA::Boolean get_boolean() = 0;
```

Returns a Boolean value from the DynAny object.

It is valid for you to use this method if the TypeCode contained in the DynAny is equivalent to _tc_boolean (a boolean TypeCode), or, if the TypeCode at the current position (a DynAny objects with components) is equivalent to _tc_boolean. The current position is unchanged after the call.

Exceptions

TypeMismatch	The accessed component in the DynAny is of a type that is not equivalent to the requested type.
InvalidValue	The DynAny has components and the current position is -1.

See Also DynamicAny::DynAny::insert_boolean()

DynAny::get_char()

```
// C++
virtual CORBA::Char get_char() = 0;
```

Returns a Char value from the DynAny object.

It is valid for you to use this method if the TypeCode contained in the DynAny is equivalent to _tc_char (a char TypeCode), or, if the TypeCode at the current position (a DynAny objects with components) is equivalent to _tc_char. The current position is unchanged after the call.

Exceptions

TypeMismatch	The accessed component in the DynAny is of a type that is not equivalent to the requested type.
InvalidValue	The DynAny has components and the current position is -1.

See Also DynamicAny::DynAny::insert_char()

DynAny::get_double()

```
// C++
virtual CORBA::Double get_double() = 0;
```

Returns a Double value from the DynAny object.

It is valid for you to use this method if the TypeCode contained in the DynAny is equivalent to _tc_double (a double TypeCode), or, if the TypeCode at the current position (a DynAny objects with components) is equivalent to _tc_double. The current position is unchanged after the call.

Exceptions

TypeMismatch	The accessed component in the DynAny is of a type that is not equivalent to the requested type.
InvalidValue	The DynAny has components and the current position is -1.

See Also DynamicAny::DynAny::insert_double()

DynAny::get_dyn_any()

```
// C++
virtual DynAny_ptr get_dyn_any() = 0;
```

Returns a DynAny reference value from the DynAny object. get_dyn_any() is provided to deal with Any values that contain another any.

It is valid for you to use this method if the TypeCode contained in the DynAny is equivalent to the TypeCode of a DynAny or, if the TypeCode at the current position (a DynAny objects with components) is equivalent the TypeCode of a DynAny. The current position is unchanged after the call.

Exceptions

TypeMismatch	The accessed component in the DynAny is of a type that is not equivalent to the requested type.
InvalidValue	The DynAny has components and the current position is -1.

See Also DynamicAny::DynAny::insert_dyn_any()

DynAny::get_float()

```
// C++
virtual CORBA::Float get_float() = 0;
```

Returns a Float value from the DynAny object.

It is valid for you to use this method if the TypeCode contained in the DynAny is equivalent to _tc_float (a float TypeCode), or, if the TypeCode at the current position (a DynAny objects with components) is equivalent to _tc_float. The current position is unchanged after the call.

Exceptions

TypeMismatch	The accessed component in the DynAny is of a type that is not equivalent to the requested type.
InvalidValue	The DynAny has components and the current position is -1.

See Also DynamicAny::DynAny::insert_float()

DynAny::get_long()

```
// C++
virtual CORBA::Long get_long() = 0;
Returns a Long value from the DynAny object.
```

It is valid for you to use this method if the TypeCode contained in the DynAny is equivalent to _tc_long (a long TypeCode), or, if the TypeCode at the current position (a DynAny objects with components) is equivalent to _tc_long. The current position is unchanged after the call.

Exceptions

TypeMismatch	The accessed component in the DynAny is of a type that is not equivalent to the requested type.
InvalidValue	The DynAny has components and the current position is -1.

See Also DynamicAny::DynAny::insert_long()

DynAny::get_longdouble()

```
// C++
virtual CORBA::LongDouble get_longdouble() = 0;
```

Returns a LongDouble value from the DynAny object.

It is valid for you to use this method if the TypeCode contained in the DynAny is equivalent to _tc_longdouble (a long double TypeCode), or, if the TypeCode at the current position (a DynAny objects with components) is equivalent to _tc_longdouble. The current position is unchanged after the call.

Exceptions

TypeMismatch	The accessed component in the DynAny is of a type that is not equivalent to the requested type.
InvalidValue	The DynAny has components and the current position is -1.

See Also DynamicAny::DynAny::insert_longdouble()

DynAny::get_longlong()

```
// C++
virtual CORBA::LongLong get_longlong() = 0;
```

Returns a LongLong value from the DynAny object.

It is valid for you to use this method if the TypeCode contained in the DynAny is equivalent to _tc_longlong (a long long TypeCode), or, if the TypeCode at the current position (a DynAny objects with components) is equivalent to _tc_longlong. The current position is unchanged after the call.

Exceptions

TypeMismatch	The accessed component in the DynAny is of a type that is not equivalent to the requested type.
InvalidValue	The DynAny has components and the current position is -1.

See Also DynamicAny::DynAny::insert_longlong()

DynAny::get_octet()

```
// C++
virtual CORBA::Octet get_octet() = 0;
```

Returns an Octet value from the DynAny object.

It is valid for you to use this method if the TypeCode contained in the DynAny is equivalent to _tc_octet (an octet TypeCode), or, if the TypeCode at the current position (a DynAny objects with components) is equivalent to _tc_octet. The current position is unchanged after the call.

Exceptions

TypeMismatch	The accessed component in the DynAny is of a type that is not equivalent to the requested type.
InvalidValue	The DynAny has components and the current position is -1.

See Also DynamicAny::DynAny::insert_octet()

DynAny::get_reference()

```
// C++
virtual CORBA::Object_ptr get_reference() = 0;
```

Returns an Object reference from the DynAny object.

It is valid for you to use this method if the TypeCode contained in the DynAny is equivalent to _tc_Object (an object reference TypeCode), or, if the TypeCode at the current position (a DynAny objects with components) is equivalent to _tc_Object. The current position is unchanged after the call.

Exceptions

TypeMismatch	The accessed component in the DynAny is of a type that is not equivalent to the requested type.
InvalidValue	The DynAny has components and the current position is -1.

See Also DynamicAny::DynAny::insert_reference()

DynAny::get_short()

```
// C++
virtual CORBA::Short get_short() = 0;
```

Returns a Short value from the DynAny object.

It is valid for you to use this method if the TypeCode contained in the DynAny is equivalent to _tc_short_tc_short (a short TypeCode), or, if the TypeCode at the current position (a DynAny objects with components) is equivalent to _tc_short. The current position is unchanged after the call.

Exceptions

TypeMismatch	The accessed component in the DynAny is of a type that is not equivalent to the requested type.
InvalidValue	The DynAny has components and the current position is -1.

See Also DynamicAny::DynAny::insert_short()

DynAny::get_string()

```
// C++
virtual char* get_string() = 0;
```

Returns a string value from the DynAny object.

It is valid for you to use this method if the TypeCode contained in the DynAny is equivalent to _tc_string (a string TypeCode), or, if the TypeCode at the current position (a DynAny objects with components) is equivalent to _tc_string. The current position is unchanged after the call.

Exceptions

TypeMismatch	The accessed component in the DynAny is of a type that is not equivalent to the requested type.
InvalidValue	The DynAny has components and the current position is -1.

See Also DynamicAny::DynAny::insert_string()

DynAny::get_typecode()

```
// C++
virtual CORBA::TypeCode_ptr get_typecode() = 0;
```

Returns a TypeCode value from the DynAny object.

It is valid for you to use this method if the TypeCode contained in the DynAny is equivalent to _tc_TypeCode (a TypeCode), or, if the TypeCode at the current position (a DynAny objects with components) is equivalent to _tc_TypeCode. The current position is unchanged after the call.

Exceptions

TypeMismatch	The accessed component in the DynAny is of a type that is not equivalent to the requested type.
InvalidValue	The DynAny has components and the current position is -1.

See Also DynamicAny::DynAny::insert_typecode()

DynAny::get_ulong()

```
// C++
virtual CORBA::ULong get_ulong() = 0;
```

Returns a ULong value from the DynAny object.

It is valid for you to use this method if the TypeCode contained in the DynAny is equivalent to _tc_ulong (an unsigned long TypeCode), or, if the TypeCode at the current position (a DynAny objects with components) is equivalent to _tc_ulong. The current position is unchanged after the call.

Exceptions

TypeMismatch	The accessed component in the DynAny is of a type that is not equivalent to the requested type.
InvalidValue	The DynAny has components and the current position is -1.

See Also DynamicAny::DynAny::insert_ulong()

DynAny::get_ulonglong()

```
// C++
virtual CORBA::ULongLong get_ulonglong() = 0;
```

Returns a ULongLong value from the DynAny object.

It is valid for you to use this method if the TypeCode contained in the DynAny is equivalent to _tc_ulonglong (an unsigned long long TypeCode), or, if the TypeCode at the current position (a DynAny objects with components) is equivalent to _tc_ulonglong. The current position is unchanged after the call.

Exceptions

TypeMismatch	The accessed component in the DynAny is of a type that is not equivalent to the requested type.
InvalidValue	The DynAny has components and the current position is -1.

See Also DynamicAny::DynAny::insert_ulonglong()

DynAny::get_ushort()

```
// C++
virtual CORBA::UShort get_ushort() = 0;
```

Returns a UShortshort value from the DynAny object.

It is valid for you to use this method if the TypeCode contained in the DynAny is equivalent to _tc_ushort (an unsigned short TypeCode), or, if the TypeCode at the current position (a DynAny objects with components) is equivalent to _tc_ushort. The current position is unchanged after the call.

Exceptions

TypeMismatch	The accessed component in the DynAny is of a type that is not equivalent to the requested type.
InvalidValue	The DynAny has components and the current position is -1.

See Also DynamicAny::DynAny::insert_ushort()

DynAny::get_val()

```
// C++
virtual CORBA::ValueBase* get_val() = 0;
```

Returns a value type value from the DynAny object.

It is valid for you to use this method if the TypeCode contained in the DynAny is equivalent to a value type TypeCode, or, if the TypeCode at the current position (a DynAny objects with components) is equivalent to a value type TypeCode. The current position is unchanged after the call.

Exceptions

TypeMismatch	The accessed component in the DynAny is of a type that is not equivalent to the requested type.
InvalidValue	The DynAny has components and the current position is -1.

See Also DynamicAny::DynAny::insert_val()

DynAny::get_wchar()

```
// C++
virtual CORBA::WChar get_wchar() = 0;
```

Returns a WChar value from the DynAny object.

It is valid for you to use this method if the TypeCode contained in the DynAny is equivalent to _tc_wchar (a wchar TypeCode), or, if the TypeCode at the current position (a DynAny objects with components) is equivalent to _tc_wchar. The current position is unchanged after the call.

Exceptions

TypeMismatch	The accessed component in the DynAny is of a type that is not equivalent to the requested type.
InvalidValue	The DynAny has components and the current position is -1.

See Also DynamicAny::DynAny::insert_wchar()

DynAny::get_wstring()

```
// C++
virtual CORBA::WChar* get_wstring() = 0;
```

Returns a wide string value from the DynAny object.

It is valid for you to use this method if the TypeCode contained in the DynAny is equivalent to _tc_wstring (a wide string TypeCode), or, if the TypeCode at the current position (a DynAny objects with components) is equivalent to _tc_wstring. The current position is unchanged after the call.

Exceptions

TypeMismatch	The accessed component in the DynAny is of a type that is not equivalent to the requested type.
InvalidValue	The DynAny has components and the current position is -1.

See Also DynamicAny::DynAny::insert_wstring()

DynAny::insert_any()

```
// C++
virtual void insert_any(
    const CORBA::Any&  value
) = 0;
```

Inserts an Any value into the DynAny object.

Parameters

value	The value to insert into the DynAny object.

It is valid for you to use this method if the TypeCode contained in the DynAny is equivalent to _tc_any (an Any TypeCode), or, if the TypeCode at the current position (a DynAny objects with components) is equivalent to _tc_any. The current position is unchanged after the call.

Exceptions

InvalidValue	The DynAny has components and the current position is -1.
TypeMismatch	The accessed component in the DynAny is of a type that is not equivalent to the inserted type.

See Also DynamicAny::DynAny::get_any()

DynAny::insert_boolean()

```
// C++
virtual void insert_boolean(
    CORBA::Boolean value
) = 0;
```

Inserts a Boolean value into the DynAny object.

Parameters

value	The value to insert into the DynAny object.

It is valid for you to use this method if the TypeCode contained in the DynAny is equivalent to _tc_boolean (a boolean TypeCode), or, if the TypeCode at the current position (a DynAny objects with components) is equivalent to _tc_boolean. The current position is unchanged after the call.

Exceptions

InvalidValue	The DynAny has components and the current position is -1.
TypeMismatch	The accessed component in the DynAny is of a type that is not equivalent to the inserted type.

See Also DynamicAny::DynAny::get_boolean()

DynAny::insert_char()

```
// C++
virtual void insert_char(
    CORBA::Char value
) = 0;
```

Inserts a Char value into the DynAny object.

Parameters

value	The value to insert into the DynAny object.

It is valid for you to use this method if the TypeCode contained in the DynAny is equivalent to _tc_char (a char TypeCode), or, if the TypeCode at the current position (a DynAny objects with components) is equivalent to _tc_char. The current position is unchanged after the call.

Exceptions

InvalidValue	The DynAny has components and the current position is -1.
TypeMismatch	The accessed component in the DynAny is of a type that is not equivalent to the inserted type.

See Also DynamicAny::DynAny::get_char()

DynAny::insert_double()

```
// C++
virtual void insert_double(
    CORBA::Double value
) = 0;
```

Inserts a Double value into the DynAny object.

Parameters

value	The value to insert into the DynAny object.

It is valid for you to use this method if the TypeCode contained in the DynAny is equivalent to _tc_double (a double TypeCode), or, if the TypeCode at the current position (a DynAny objects with components) is equivalent to _tc_double. The current position is unchanged after the call.

Exceptions

InvalidValue	The DynAny has components and the current position is -1.
TypeMismatch	The accessed component in the DynAny is of a type that is not equivalent to the inserted type.

See Also DynamicAny::DynAny::get_double()

DynAny::insert_dyn_any()

```
// C++
virtual void insert_dyn_any(
    DynAny_ptr  value
) = 0;
```

Inserts a DynAny value into the DynAny object. `insert_dyn_any()` is provided to deal with Any values that contain another any.

Parameters

`value`	The value to insert into the DynAny object.

It is valid for you to use this method if the TypeCode contained in the DynAny is equivalent to the TypeCode of a DynAny or, if the TypeCode at the current position (a DynAny objects with components) is equivalent the TypeCode of a DynAny. The current position is unchanged after the call.

Exceptions

InvalidValue	The DynAny has components and the current position is -1.
TypeMismatch	The accessed component in the DynAny is of a type that is not equivalent to the inserted type.

See Also DynamicAny::DynAny::get_dyn_any()

DynAny::insert_float()

```
// C++
virtual void insert_float(
    CORBA::Float value
) = 0;
```

Inserts a Float value into the DynAny object.

Parameters

`value`	The value to insert into the DynAny object.

It is valid for you to use this method if the TypeCode contained in the DynAny is equivalent to _tc_float (a `float` TypeCode), or, if the TypeCode at the current position (a DynAny objects with components) is equivalent to _tc_float. The current position is unchanged after the call.

Exceptions

InvalidValue	The DynAny has components and the current position is -1.
TypeMismatch	The accessed component in the DynAny is of a type that is not equivalent to the inserted type.

See Also DynamicAny::DynAny::get_float()

DynAny::insert_long()

```
// C++
virtual void insert_long(
    CORBA::Long value
) = 0;
```

Inserts a Long value into the DynAny object.

Parameters

value	The value to insert into the DynAny object.

It is valid for you to use this method if the TypeCode contained in the DynAny is equivalent to _tc_long (a long TypeCode), or, if the TypeCode at the current position (a DynAny objects with components) is equivalent to _tc_long. The current position is unchanged after the call.

Exceptions

TypeMismatch	The accessed component in the DynAny is of a type that is not equivalent to the requested type.
InvalidValue	The DynAny has components and the current position is -1.

See Also DynamicAny::DynAny::get_long()

DynAny::insert_longdouble()

```
// C++
virtual void insert_longdouble(
    CORBA::LongDouble value
) = 0;
```

Inserts a LongDouble value into the DynAny object.

Parameters

value	The value to insert into the DynAny object.

It is valid for you to use this function if the TypeCode contained in the DynAny is equivalent to _tc_longdouble (a long double TypeCode), or, if the TypeCode at the current position (a DynAny objects with components) is equivalent to _tc_longdouble. The current position is unchanged after the call.

Exceptions

TypeMismatch	The accessed component in the DynAny is of a type that is not equivalent to the requested type.
InvalidValue	The DynAny has components and the current position is -1.

See Also DynamicAny::DynAny::get_longdouble()

DynAny::insert_long long()

```
// C++
virtual void insert_longlong(
    CORBA::LongLong value
) = 0;
```

Inserts a LongLong value into the DynAny object.

Parameters

value	The value to insert into the DynAny object.

It is valid for you to use this method if the TypeCode contained in the DynAny is equivalent to _tc_longlong (a long long TypeCode), or, if the TypeCode at the current position (a DynAny objects with components) is equivalent to _tc_longlong. The current position is unchanged after the call.

Exceptions

InvalidValue	The DynAny has components and the current position is -1.
TypeMismatch	The accessed component in the DynAny is of a type that is not equivalent to the inserted type.

See Also DynamicAny::DynAny::get_longlong()

DynAny::insert_octet()

```
// C++
virtual void insert_octet(
    CORBA::Octet value
) = 0;
```

Inserts an Octet value into the DynAny object.

Parameters

value	The value to insert into the DynAny object.

It is valid for you to use this method if the TypeCode contained in the DynAny is equivalent to _tc_octet (an octet TypeCode), or, if the TypeCode at the current position (a DynAny objects with components) is equivalent to _tc_octet. The current position is unchanged after the call.

Exceptions

TypeMismatch	The accessed component in the DynAny is of a type that is not equivalent to the requested type.
InvalidValue	The DynAny has components and the current position is -1.

See Also

DynamicAny::DynAny::get_octet()

DynAny::insert_reference()

```
// C++
virtual void insert_reference(
    CORBA::Object_ptr  value
) = 0;
```

Inserts an Object reference into the DynAny object.

Parameters

value	The value to insert into the DynAny object.

It is valid for you to use this method if the TypeCode contained in the DynAny is equivalent to _tc_Object (an object reference TypeCode), or, if the TypeCode at the current position (a DynAny objects with components) is equivalent to _tc_Object. The current position is unchanged after the call.

Exceptions

TypeMismatch	The accessed component in the DynAny is of a type that is not equivalent to the requested type.
InvalidValue	The DynAny has components and the current position is -1.

See Also DynamicAny::DynAny::get_reference()

DynAny::insert_short()

```
// C++
virtual void insert_short(
    CORBA::Short value
) = 0;
```

Inserts a Short value into the DynAny object.

Parameters

value	The value to insert into the DynAny object.

It is valid for you to use this method if the TypeCode contained in the DynAny is equivalent to _tc_short (a short TypeCode), or, if the TypeCode at the current position (a DynAny objects with components) is equivalent to _tc_short. The current position is unchanged after the call.

Exceptions

TypeMismatch	The accessed component in the DynAny is of a type that is not equivalent to the requested type.
InvalidValue	The DynAny has components and the current position is -1.

See Also DynamicAny::DynAny::get_short()

DynAny::insert_string()

```
// C++
virtual void insert_string(
    const char*  value
) = 0;
```

Inserts a string into the DynAny object.

Parameters

`value`	The value to insert into the DynAny object.

You can insert both bounded and unbounded strings using `insert_string()`.

It is valid for you to use this method if the TypeCode contained in the DynAny is equivalent to `_tc_string` (a `string` TypeCode), or, if the TypeCode at the current position (a DynAny objects with components) is equivalent to `_tc_string`. The current position is unchanged after the call.

Exceptions

`TypeMismatch`	The accessed component in the DynAny is of a type that is not equivalent to the inserted type.
`InvalidValue`	• The DynAny has components and the current position is -1. • The string inserted is longer than the bound of a bounded string.

See Also `DynamicAny::DynAny::get_string()`

DynAny::insert_typecode()

```
// C++
virtual void insert_typecode(
    ::CORBA::TypeCode_ptr  value
) = 0;
```

Inserts a TypeCode value into the DynAny object.

Parameters

`value`	The value to insert into the DynAny object.

It is valid for you to use this method if the TypeCode contained in the DynAny is equivalent to `_tc_TypeCode` (a TypeCode), or, if the TypeCode at the current position (a DynAny objects with components) is equivalent to `_tc_TypeCode`. The current position is unchanged after the call.

Exceptions

TypeMismatch	The accessed component in the DynAny is of a type that is not equivalent to the requested type.
InvalidValue	The DynAny has components and the current position is -1.

See Also DynamicAny::DynAny::get_typecode()

DynAny::insert_ulong()

```
// C++
virtual void insert_ulong(
    CORBA::ULong value
) = 0;
```

Inserts a ULong value into the DynAny object.

Parameters

value	The value to insert into the DynAny object.

It is valid for you to use this method if the TypeCode contained in the DynAny is equivalent to _tc_ulong (an unsigned long TypeCode), or, if the TypeCode at the current position (a DynAny objects with components) is equivalent to _tc_ulong. The current position is unchanged after the call.

Exceptions

TypeMismatch	The accessed component in the DynAny is of a type that is not equivalent to the requested type.
InvalidValue	The DynAny has components and the current position is -1.

See Also DynamicAny::DynAny::get_ulong()

DynAny::insert_ulonglong()

```
// C++
virtual void insert_ulonglong(
    CORBA::ULongLong value
) = 0;
```

Inserts a ULongLong value into the DynAny object.

Parameters

value	The value to insert into the DynAny object.

It is valid for you to use this method if the TypeCode contained in the DynAny is equivalent to _tc_ulonglong (an unsigned long long TypeCode), or, if the TypeCode at the current position (a DynAny objects with components) is equivalent to _tc_ulonglong. The current position is unchanged after the call.

Exceptions

TypeMismatch	The accessed component in the DynAny is of a type that is not equivalent to the requested type.
InvalidValue	The DynAny has components and the current position is -1.

See Also DynamicAny::DynAny::get_ulonglong()

DynAny::insert_ushort()

```
// C++
virtual void insert_ushort(
    CORBA::UShort value
) = 0;
```

Inserts a UShort value into the DynAny object.

Parameters

value	The value to insert into the DynAny object.

It is valid for you to use this method if the TypeCode contained in the DynAny is equivalent to _tc_ushort (an unsigned short TypeCode), or, if the TypeCode at the current position (a DynAny objects with components) is equivalent to _tc_ushort. The current position is unchanged after the call.

Exceptions

TypeMismatch	The accessed component in the DynAny is of a type that is not equivalent to the requested type.
InvalidValue	The DynAny has components and the current position is -1.

See Also DynamicAny::DynAny::get_ushort()

DynAny::insert_val()

```
// C++
virtual void insert_val(
    CORBA::ValueBase*  value
) = 0;
```

Inserts a value type value into the DynAny object.

Parameters

value	The value to insert into the DynAny object.

It is valid for you to use this method if the TypeCode contained in the DynAny is equivalent to a value type TypeCode, or, if the TypeCode at the current position (a DynAny objects with components) is equivalent to a value type TypeCode. The current position is unchanged after the call.

Exceptions

TypeMismatch	The accessed component in the DynAny is of a type that is not equivalent to the requested type.
InvalidValue	The DynAny has components and the current position is -1.

See Also DynamicAny::DynAny::get_val()

DynAny::insert_wchar()

```
// C++
virtual void insert_wchar(
    CORBA::WChar value
) = 0;
```

Inserts a WChar value into the DynAny object.

Parameters

value	The value to insert into the DynAny object.

It is valid for you to use this method if the TypeCode contained in the DynAny is equivalent to _tc_wchar (a wide character TypeCode), or, if the TypeCode at the current position (a DynAny objects with components) is equivalent to _tc_wchar. The current position is unchanged after the call.

Exceptions

TypeMismatch	The accessed component in the DynAny is of a type that is not equivalent to the requested type.
InvalidValue	The DynAny has components and the current position is -1.

See Also DynamicAny::DynAny::get_wchar()

DynAny::insert_wstring()

```
// C++
virtual void insert_wstring(
    const CORBA::WChar*  value
) = 0;
```

Inserts a wide string into the DynAny object.

Parameters

value	The value to insert into the DynAny object.

You can insert both bounded and unbounded strings using `insert_wstring()`.

It is valid for you to use this method if the TypeCode contained in the DynAny is equivalent to _tc_wstring (a wide `string` TypeCode), or, if the TypeCode at the current position (a DynAny objects with components) is equivalent to _tc_wstring. The current position is unchanged after the call.

Exceptions

TypeMismatch	The accessed component in the DynAny is of a type that is not equivalent to the inserted type.
InvalidValue	• The DynAny has components and the current position is -1. • The string inserted is longer than the bound of a bounded string.

See Also DynamicAny::DynAny::get_wstring()

DynAny::InvalidValue User Exception

```
class IT_DYNANY_API InvalidValue: public CORBA::UserException {
public:
    InvalidValue();
    void operator=(
        const InvalidValue&
    );
    static InvalidValue* _downcast(
        CORBA::Exception* exc
    );
    static const InvalidValue* _downcast(
        const CORBA::Exception* exc
    );
    static InvalidValue* _narrow(
        CORBA::Exception* exc
    );
    static const InvalidValue* _narrow(
        const CORBA::Exception* exc
    );
    virtual void _raise() const;
    virtual CORBA::TypeCode_ptr _it_get_typecode() const;
    virtual CORBA::Exception* _it_copy() const;
    virtual void _it_insert(
        CORBA::Any& any,
        CORBA::Boolean consume
    );
    virtual ~InvalidValue();
};
static CORBA::TypeCode_ptr _tc_InvalidValue;
```

A user exception meaning that an invalid value has been used as a parameter.

See Also DynamicAny::DynAny::TypeMismatch

DynAny::next()

```
// C++
virtual CORBA::Boolean next() = 0;
```

Advances the current position to the next component of the DynAny object. Returns true if the resulting current position indicates a component, false

otherwise. Invoking `next()` on a DynAny that has no components returns false. A false return value always sets the current position to -1.

See Also

DynamicAny::DynAny::component_count()
DynamicAny::DynAny::current_component()
DynamicAny::DynAny::seek()
DynamicAny::DynAny::rewind()

DynAny::rewind()

```
// C++
virtual void rewind() = 0;
```

Sets the current position to the first component of the DynAny object. This is equivalent to calling `seek(0)`.

See Also

DynamicAny::DynAny::seek()

DynAny::seek()

```
// C++
virtual CORBA::Boolean seek(
    CORBA::Long index
) = 0;
```

Sets the current position to a component of the DynAny object. The method returns true if the resulting current position indicates a component of the DynAny object and false if the position does not correspond to a component.

Parameters

`index` The new index to set the current position to. An index can range from 0 to *n*-1. An index of zero corresponds to the first component.

Calling seek with a negative index is legal and sets the current position to -1 to indicate no component. The method returns false in this case.

Passing a non-negative index value for a DynAny that does not have a component at the corresponding position sets the current position to - 1 and returns false.

See Also

DynamicAny::DynAny::component_count()
DynamicAny::DynAny::current_component()
DynamicAny::DynAny::rewind()
DynamicAny::DynAny::next()

DynAny::to_any()

```
// C++
virtual CORBA::Any* to_any() = 0;
```

Returns an Any value created from a DynAny object. A copy of the TypeCode associated with the DynAny object is assigned to the resulting any. The value associated with the DynAny object is copied into the Any value.

See Also

DynamicAny::DynAny::from_any()

DynAny::type()

```
// C++
virtual CORBA::TypeCode_ptr type() = 0;
```

Returns the TypeCode associated with a DynAny object.

A DynAny object is created with a TypeCode value assigned to it. This value determines the type of the value handled through the DynAny object. `type()` returns the TypeCode associated with a DynAny object.

Note that the TypeCode associated with a DynAny object is initialized at the time the DynAny is created and cannot be changed during the lifetime of the DynAny object.

DynAny::TypeMismatch User Exception

```
class IT_DYNANY_API TypeMismatch: public CORBA::UserException {
public:
    TypeMismatch();
    void operator=(
        const TypeMismatch&
    );
    static TypeMismatch* _downcast(
        CORBA::Exception* exc
```

```
    );
    static const TypeMismatch* _downcast(
        const CORBA::Exception* exc
    );
    static TypeMismatch* _narrow(
        CORBA::Exception* exc
    );
    static const TypeMismatch* _narrow(
        const CORBA::Exception* exc
    );
    virtual void _raise() const;
    virtual CORBA::TypeCode_ptr _it_get_typecode() const;
    virtual CORBA::Exception* _it_copy() const;
    virtual void _it_insert(
        CORBA::Any& any,
        CORBA::Boolean consume
    );
    virtual ~TypeMismatch();
};
static CORBA::TypeCode_ptr _tc_TypeMismatch;
```

A user exception meaning that the type of a parameter does not match the type of the target.

This exception is also raised when attempts are made to access DynAny components illegally. For example:

- If an attempt is made to access an object's component but the type of object does not have components.
- If an attempt is made to call an `insert_`*`type`*`()` or `get_`*`type`*`()` method on a DynAny object whose current component itself has components.

See Also DynamicAny::DynAny::InvalidValue

DynamicAny::DynAnyFactory Class

You can create DynAny objects by invoking operations on the DynAnyFactory object. You obtain a reference to the DynAnyFactory object by calling CORBA::ORB::resolve_initial_references() with the identifier parameter set to "DynAnyFactory".

A typical first step in dynamic interpretation of an Any involves creating a DynAny object using create_dyn_any() or create_dyn_any_from_type_code(). Then, depending on the type of the Any, you narrow the resulting DynAny object reference to one of the following complex types of object references:

DynFixed
DynStruct
DynSequence
DynArray
DynUnion
DynEnum
DynValue

Finally, you can use DynAny::to_any() (which each of these classes inherits from the DynAny class) to create an Any value from the constructed DynAny.

Exceptions

MARSHAL: an attempt is made to exported references to DynAnyFactory objects to other processes or if an attempt is made to externalized them with ORB::object_to_string(). DynAnyFactory objects are intended to be local to the process in which they are created and used.

```
// class is in namespace DynamicAny
class IT_DYNANY_API DynAnyFactory : public virtual CORBA::Object {
public:
    typedef DynamicAny::DynAnyFactory_ptr _ptr_type;
    typedef DynamicAny::DynAnyFactory_var _var_type;

    virtual ~DynAnyFactory();
    static DynAnyFactory_ptr _narrow(
        CORBA::Object_ptr obj
    );
    static DynAnyFactory_ptr _unchecked_narrow(
        CORBA::Object_ptr obj
```

```
    );
    inline static DynAnyFactory_ptr _duplicate(
        DynAnyFactory_ptr p
    );
    inline static DynAnyFactory_ptr _nil();

    class IT_DYNANY_API InconsistentTypeCode:
        public CORBA::UserException
    { ... }
    static CORBA::TypeCode_ptr _tc_InconsistentTypeCode;

    virtual DynAny_ptr create_dyn_any(
        const CORBA::Any&  value
    ) = 0;
    virtual DynAny_ptr create_dyn_any_from_type_code(
        ::CORBA::TypeCode_ptr  type
    ) = 0;

    static const IT_FWString  _it_fw_type_id;
};
```

See page 4 for descriptions of the standard helper functions:

- `_duplicate()`
- `_narrow()`
- `_nil()`
- `_unchecked_narrow()`

DynAnyFactory::create_dyn_any()

```
// C++
virtual DynAny_ptr create_dyn_any(
    const CORBA::Any&  value
) = 0;
```

Returns a new DynAny object from an Any value.

Parameters

`value`	An Any value to use to set the DynAny object.

A copy of the TypeCode associated with the any value is assigned to the resulting DynAny object. The value associated with the DynAny object is a copy of the value in the original Any. The current position of the created DynAny object is set to zero if the passed value has components; otherwise, the current position is set to -1.

Exceptions InconsistentTypeCode: the value has a TypeCode with a TCKind of tk_Principal, tk_native, or tk_abstract_interface.

See Also DynamicAny::DynAnyFactory::create_dyn_any_from_type_code()

DynAnyFactory::create_dyn_any_from_type_code()

```
// C++
virtual DynAny_ptr create_dyn_any_from_type_code(
    ::CORBA::TypeCode_ptr  type
) = 0;
```

Returns a new DynAny object from a TypeCode value. Depending on the TypeCode, the created object may be of type DynAny, or one of its derived types, such as DynStruct. The returned reference can be narrowed to the derived type.

Parameters

type A TypeCode value to use to set the DynAny object.

Table 11 shows the initial default values set depending on the type created:

Table 11: *Default Values When Using create_dyn_any_from_type_code()*

Type	Default Value
Any values	An Any containing a TypeCode with a TCKind value of tk_null and no value.
Boolean	FALSE
char	zero
DynArray	The operation sets the current position to zero and recursively initializes elements to their default value.

Table 11: *Default Values When Using create_dyn_any_from_type_code()*

Type	Default Value
DynEnum	The operation sets the current position to -1 and sets the value of the enumerator to the first enumerator value indicated by the TypeCode.
DynFixed	Operations set the current position to -1 and sets the value to zero.
DynSequence	The operation sets the current position to -1 and creates an empty sequence.
DynStruct	The operation sets the current position to -1 for empty exceptions and to zero for all other TypeCode values. The members (if any) are recursively initialized to their default values.
DynUnion	The operation sets the current position to zero. The discriminator value is set to a value consistent with the first named member of the union. That member is activated and recursively initialized to its default value.
DynValue	The members are initialized as for a DynStruct.
numeric types	zero
object references	nil
octet	zero
string	the empty string
TypeCode	A TypeCode with a TCKind value of tk_null
wchar	zero
wstring	the empty string

Exceptions InconsistentTypeCode: the TypeCode has a TCKind of tk_Principal, tk_native, or tk_abstract_interface.

See Also DynamicAny::DynAnyFactory::create_dyn_any()

DynAnyFactory::~DynAnyFactory() Destructor

```
// C++
virtual ~DynAnyFactory();
```

Destroys the DynAnyFactory object.

See Also CORBA::ORB::resolve_initial_references()
CORBA::ORB::list_initial_services()

DynAnyFactory::InconsistentTypeCode User Exception Class

```
// C++
class IT_DYNANY_API InconsistentTypeCode:
    public CORBA::UserException
{
  public:

    InconsistentTypeCode();
    void operator=(
        const InconsistentTypeCode&
    );

    static InconsistentTypeCode* _downcast(
        CORBA::Exception* exc
    );
    static const InconsistentTypeCode* _downcast(
        const CORBA::Exception* exc
    );
    static InconsistentTypeCode* _narrow(
        CORBA::Exception* exc
    );
    static const InconsistentTypeCode* _narrow(
        const CORBA::Exception* exc
    );
    virtual void _raise() const;

    virtual CORBA::TypeCode_ptr _it_get_typecode() const;
```

```
    virtual CORBA::Exception* _it_copy() const;

    virtual void _it_insert(
        CORBA::Any& any,
        CORBA::Boolean consume
    );

    virtual ~InconsistentTypeCode();
};
static CORBA::TypeCode_ptr _tc_InconsistentTypeCode;
```

A user exception meaning that a parameter has an inconsistent TypeCode compared to the object.

DynamicAny::DynArray Class

DynArray objects let you dynamically manipulate Any values as arrays. The following methods let you get and set array elements:

get_elements()
set_elements()
get_elements_as_dyn_any()
set_elements_as_dyn_any()

This class inherits from the DynAny class. Use component_count() to get the dimension of the array. Use the iteration methods such as seek() to access portions of the array.

```
// C++ class is in namespace DynamicAny
class IT_DYNANY_API DynArray : public virtual DynAny {
public:
    typedef DynamicAny::DynArray_ptr _ptr_type;
    typedef DynamicAny::DynArray_var _var_type;

    virtual ~DynArray();
    static DynArray_ptr _narrow(
        CORBA::Object_ptr obj
    );
    static DynArray_ptr _unchecked_narrow(
        CORBA::Object_ptr obj
    );
    inline static DynArray_ptr _duplicate(
        DynArray_ptr p
    );
    inline static DynArray_ptr _nil();

    virtual AnySeq* get_elements() = 0;
    virtual void set_elements(
        const AnySeq &  value
    ) = 0;
    virtual DynAnySeq* get_elements_as_dyn_any() = 0;
    virtual void set_elements_as_dyn_any(
        const DynAnySeq &  value
    ) = 0;
```

```
    static const IT_FWString _it_fw_type_id;
};
```

See Also DynamicAny::DynAny

See page 4 for descriptions of the standard helper functions:

- _duplicate()
- _narrow()
- _nil()
- _unchecked_narrow()

DynArray::~DynArray() Destructor

```
// C++
virtual ~DynArray();
```

The destructor for a DynArray object.

DynArray::get_elements()

```
// C++
virtual AnySeq* get_elements() = 0;
```

Returns a sequence of Any values containing the elements of the array.

See Also DynamicAny::DynArray::set_elements()
DynamicAny::DynArray::get_elements_as_dyn_any()
DynamicAny::DynAny::component_count()

DynArray::get_elements_as_dyn_any()

```
// C++
virtual DynAnySeq* get_elements_as_dyn_any() = 0;
```

Returns a sequence of DynAny objects that describes each member in the array.

Use this method instead of get_elements() if you want to avoid converting DynAny objects to Any objects when your application needs to handle DynArray objects extensively.

See Also

DynamicAny::DynArray::get_elements()
DynamicAny::DynArray::set_elements_as_dyn_any()
DynamicAny::DynAny::component_count()

DynArray::set_elements()

```
// C++
virtual void set_elements(
    const AnySeq &  value
) = 0;
```

Sets the array values with a sequence of Any values.

Parameters

value	A sequence of Any values containing the elements for the array.

This method sets the current position to -1 if the sequence has a zero length and it sets it to 0 otherwise.

Exceptions

TypeMismatch is raised if an inconsistent value is passed in the sequence.

InvalidValue is raised if the sequence length does not match the array length.

See Also

DynamicAny::DynArray::get_elements()
DynamicAny::DynArray::set_elements_as_dyn_any()
DynamicAny::DynAny::component_count()

DynArray::set_elements_as_dyn_any()

```
// C++
virtual void set_elements_as_dyn_any(
    const DynAnySeq &  value
) = 0;
```

Initializes the array data associated with a DynArray object from a sequence of DynAny objects. Use this method instead of set_elements() if you want to avoid converting DynAny objects to Any objects when your application needs to handle DynArray objects extensively.

Parameters

`value`	A sequence of `DynAny` objects representing the array elements.

This method sets the current position to -1 if the sequence has a zero length and it sets it to 0 otherwise.

Exceptions

`TypeMismatch` is raised if an inconsistent value is passed in the sequence.

`InvalidValue` is raised if the sequence length does not match the array length.

See Also

`DynamicAny::DynArray::get_elements_as_dyn_any()`
`DynamicAny::DynArray::set_elements()`
`DynamicAny::DynAny::component_count()`

DynamicAny::DynEnum Class

A DynEnum object lets you dynamically manipulate an Any value as an enumerated value. The key methods allow you to get and set a value as an IDL identifier string or you can manipulate the number that the enumerated value represents:

get_as_string()
set_as_string()
get_as_ulong()
set_as_ulong()

This class inherits from the DynAny class. The current position of a DynEnum is always -1 because it can only be one value at a given time.

```
// C++ class is in namespace DynamicAny
class IT_DYNANY_API DynEnum : public virtual DynAny {
public:

    typedef DynamicAny::DynEnum_ptr _ptr_type;
    typedef DynamicAny::DynEnum_var _var_type;

    virtual ~DynEnum();
    static DynEnum_ptr _narrow(
        CORBA::Object_ptr obj
    );
    static DynEnum_ptr _unchecked_narrow(
        CORBA::Object_ptr obj
    );
    inline static DynEnum_ptr _duplicate(
        DynEnum_ptr p
    );
    inline static DynEnum_ptr _nil();

    virtual char* get_as_string() = 0;
    virtual void set_as_string(
        const char*  value
    ) = 0;
    virtual CORBA::ULong get_as_ulong() = 0;
    virtual void set_as_ulong(
```

```
        CORBA::ULong value
    ) = 0;

    static const IT_FWString _it_fw_type_id;
};
```

See Also DynamicAny::DynAny

See page 4 for descriptions of the standard helper functions:

- _duplicate()
- _narrow()
- _nil()
- _unchecked_narrow()

DynEnum::~DynEnum() Destructor

```
virtual ~DynEnum();
```

The destructor for a DynEnum object.

DynEnum::get_as_string()

```
// C++
virtual char* get_as_string() = 0;
```

Returns a string for the DynEnum that represents the IDL enumeration identifier.

See Also DynamicAny::DynEnum::set_as_string()
DynamicAny::DynEnum::get_as_ulong()

DynEnum::get_as_ulong()

```
// C++
virtual CORBA::ULong get_as_ulong() = 0;
```

Returns a number for the DynEnum that represents the enumerated ordinal value. Enumerators have ordinal values of 0 to *n*-1, as they appear from left to right in the corresponding IDL definition.

See Also DynamicAny::DynEnum::set_as_ulong()
DynamicAny::DynEnum::get_as_string()

DynEnum::set_as_string()

```
// C++
virtual void set_as_string(
    const char*  value
) = 0;
```

Sets the enumerated identifier string value for the DynEnum.

Parameters

value	The identifier string to set the enumerated value to.

Exceptions

InvalidValue	The value string is not a valid IDL identifier for the corresponding IDL enumerated type.

See Also

DynamicAny::DynEnum::get_as_string()
DynamicAny::DynEnum::set_as_ulong()

DynEnum::set_as_ulong()

```
void set_as_ulong(
        int value
    ) throws org.omg.DynamicAny.DynAnyPackage.InvalidValue;

virtual void set_as_ulong(
    CORBA::ULong value
) = 0;
```

Sets the numerical value for the DynEnum that represents the enumerated ordinal value.

Parameters

value	The number to set the enumerated value to.

Exceptions

InvalidValue	The value is outside the range of ordinal values for the corresponding IDL enumerated type.

See Also

DynamicAny::DynEnum::get_as_ulong()
DynamicAny::DynEnum::set_as_string()

DynamicAny::DynFixed Class

A DynFixed object lets you dynamically manipulate an Any value as a fixed point value. This class inherits from the DynAny class. The key methods include get_value() and set_value().

These methods use strings to represent fixed-point values. A fixed-point format consists of an integer part of digits, a decimal point, a fraction part of digits, and a d or D. Examples include:

```
1.2d
35.98D
456.32
.467
```

Either the integer part or the fraction part (but not both) may be missing. The decimal point is not required for whole numbers. The d or D are optional. leading or trailing white space is allowed.

```
// C++ class is in namespace DynamicAny
class IT_DYNANY_API DynFixed : public virtual DynAny {
public:
    typedef DynamicAny::DynFixed_ptr _ptr_type;
    typedef DynamicAny::DynFixed_var _var_type;

    virtual ~DynFixed();
    static DynFixed_ptr _narrow(
        CORBA::Object_ptr obj
    );
    static DynFixed_ptr _unchecked_narrow(
        CORBA::Object_ptr obj
    );
    inline static DynFixed_ptr _duplicate(
        DynFixed_ptr p
    );
    inline static DynFixed_ptr _nil();

    virtual char* get_value() = 0;
    virtual CORBA::Boolean set_value(
        const char*  val
```

```
    ) = 0;

    static const IT_FWString _it_fw_type_id;
};
```

See Also DynamicAny::DynAny

See page 4 for descriptions of the standard helper functions:

- _duplicate()
- _narrow()
- _nil()
- _unchecked_narrow()

DynFixed::~DynFixed() Destructor

```
// C++
virtual ~DynFixed();
```

The destructor for a DynFixed object.

DynFixed::get_value()

```
// C++
virtual char* get_value() = 0;
```

Returns a string representing the fixed value of the DynFixed object.

See Also DynamicAny::DynFixed::set_value()

DynFixed::set_value()

```
// C++
virtual CORBA::Boolean set_value(
    const char*  val
) = 0;
```

Sets the value of the DynFixed. The method returns true if val can be represented as the DynFixed without loss of precision. If val has more fractional

digits than can be represented in the DynFixed, the fractional digits are truncated and the method returns false.

Parameters

val	A string containing the fixed point value to be set in the DynFixed. The string must contain a fixed string constant in the same format as would be used for IDL fixed-point literals. However, the trailing d or D is optional.

Exceptions

InvalidValue	val contains a value whose scale exceeds that of the DynFixed or is not initialized.
TypeMismatch	val does not contain a valid fixed-point literal or contains extraneous characters other than leading or trailing white space.

See Also DynamicAny::DynFixed::get_value()

DynamicAny::DynSequence Class

DynSequence objects let you dynamically manipulate Any values as sequences. The key methods allow you to manage the sequence length and get and set sequence elements:

get_length()
set_length()
get_elements()
set_elements()
get_elements_as_dyn_any()
set_elements_as_dyn_any()

This class inherits from the DynAny class.

```
// C++ class is in namespace DynamicAny
class IT_DYNANY_API DynSequence : public virtual DynAny {
public:
    typedef DynamicAny::DynSequence_ptr _ptr_type;
    typedef DynamicAny::DynSequence_var _var_type;

    virtual ~DynSequence();
    static DynSequence_ptr _narrow(
        CORBA::Object_ptr obj
    );
    static DynSequence_ptr _unchecked_narrow(
        CORBA::Object_ptr obj
    );
    inline static DynSequence_ptr _duplicate(
        DynSequence_ptr p
    );
    inline static DynSequence_ptr _nil();

    virtual CORBA::ULong get_length() = 0;
    virtual void set_length(
        CORBA::ULong len
    ) = 0;
    virtual AnySeq* get_elements() = 0;
    virtual void set_elements(
        const AnySeq &  value
```

```
    ) = 0;
    virtual DynAnySeq* get_elements_as_dyn_any() = 0;
    virtual void set_elements_as_dyn_any(
        const DynAnySeq &  value
    ) = 0;

    static const IT_FWString _it_fw_type_id;
};
```

See Also DynamicAny::DynAny

See page 4 for descriptions of the standard helper functions:

- _duplicate()
- _narrow()
- _nil()
- _unchecked_narrow()

DynSequence::~DynSequence()

```
virtual ~DynSequence();
```

The destructor for a DynSequence object.

DynSequence::get_elements()

```
virtual AnySeq* get_elements() = 0;
```

Returns a sequence of Any values containing the elements of the sequence.

See Also DynamicAny::DynSequence::set_elements()
DynamicAny::DynSequence::get_elements_as_dyn_any()

DynSequence::get_elements_as_dyn_any()

```
virtual DynAnySeq* get_elements_as_dyn_any() = 0;
```

Returns a sequence of DynAny objects that describes each member in the sequence.

Use this method instead of get_elements() if you want to avoid converting DynAny objects to Any objects when your application needs to handle DynSequence objects extensively.

See Also

DynamicAny::DynSequence::get_elements()
DynamicAny::DynSequence::get_elements_as_dyn_any()

DynSequence::get_length()

```
virtual CORBA::ULong get_length() = 0;
```

Returns the number of elements in the sequence.

See Also

DynamicAny::DynSequence::set_length()
DynamicAny::DynSequence::get_elements()

DynSequence::set_elements()

```
virtual void set_elements(
    const AnySeq &  value
) = 0;
```

Sets the sequence values.

Parameters

value	A sequence of Any values containing the elements for the sequence.

This method sets the current position to -1 if the sequence has a zero length and it sets it to 0 otherwise.

Exceptions

`Invalidvalue`	The parameter's length is greater than the DynSequence length.
`TypeMismatch`	an inconsistent value is passed in. This can happen if: • The element type codes between the DynSequence and the parameter do not agree. • The DynSequence is a bounded sequence and the number of elements in the parameter are greater than the bound allows.

See Also

`DynamicAny::DynSequence::get_elements()`
`DynamicAny::DynSequence::set_elements_as_dyn_any()`
`DynamicAny::DynSequence::get_length()`
`DynamicAny::DynSequence::set_length()`

DynSequence::set_elements_as_dyn_any()

```
virtual void set_elements_as_dyn_any(
    const DynAnySeq &  value
) = 0;
```

Initializes the sequence data associated with a DynSequence object from a sequence of DynAny objects. Use this method instead of `set_elements()` if you want to avoid converting DynAny objects to Any objects when your application needs to handle DynSequence objects extensively.

Parameters

`value`	A sequence of DynAny objects to represent the elements of the DynSequence.

This method sets the current position to -1 if the sequence has a zero length and it sets it to 0 otherwise.

Exceptions

`Invalidvalue`	The parameter's length is greater than the `DynSequence` length.
`TypeMismatch`	An inconsistent value is passed in. This can happen if: • The element type codes between the `DynSequence` and the parameter do not agree. • The `DynSequence` is a bounded sequence and the number of elements in the parameter are greater than the bound allows.

See Also

`DynamicAny::DynSequence::get_elements_as_dyn_any()`
`DynamicAny::DynSequence::set_elements()`
`DynamicAny::DynSequence::get_length()`
`DynamicAny::DynSequence::set_length()`

DynSequence::set_length()

```
void set_length(
        int len
    ) throws org.omg.DynamicAny.DynAnyPackage.InvalidValue;

virtual void set_length(
    CORBA::ULong len
) = 0;
```

Sets the length of the sequence.

Parameters

`len`	The length desired for the sequence.

Increasing the length adds new (default-initialized) elements to the end of the sequence without affecting existing elements in the sequence. The new current position is set to the first new element if the previous current position was -1. The new current position remains the same as the old one if the previous current position indicates a valid element (was anything but -1).

Decreasing the length removes elements from the end of the sequence without affecting the rest of the elements. The new current position is as follows:

- If the previous current position indicates a valid element and that element is not removed, the new current position remains the same.
- If the previous current position indicates a valid element and that element is removed, the new current position is set to -1.
- If the sequence length is set to 0, the new current position is set to -1.
- If the previous current position was -1, the new current position remains -1.

Exceptions

`InvalidValue`	An attempt is made to increase the length of a bounded sequence to a value greater than the bound.

See Also

`DynamicAny::DynSequence::get_length()`
`DynamicAny::DynSequence::set_elements()`

DynamicAny::DynStruct Class

You use DynStruct objects for dynamically handling structures and exceptions in Any values. This class inherits from the DynAny class. Key methods allow you to set and get the structure (or exception) as a sequence of name-value pairs:

get_members()
set_members()
get_members_as_dyn_any()
set_members_as_dyn_any()

Use the DynAny iteration methods such as seek() to set the current position to a member of the structure. You can also obtain the name and kind of TypeCode for a member at the current position:

current_member_name()
current_member_kind()

```
// C++ class is in namespace DynamicAny
class IT_DYNANY_API DynStruct : public virtual DynAny {
public:
    typedef DynamicAny::DynStruct_ptr _ptr_type;
    typedef DynamicAny::DynStruct_var _var_type;

    virtual ~DynStruct();
    static DynStruct_ptr _narrow(
        CORBA::Object_ptr obj
    );
    static DynStruct_ptr _unchecked_narrow(
        CORBA::Object_ptr obj
    );
    inline static DynStruct_ptr _duplicate(
        DynStruct_ptr p
    );
    inline static DynStruct_ptr _nil();

    virtual FieldName current_member_name() = 0;
    virtual ::CORBA::TCKind current_member_kind() = 0;
```

```
    virtual NameValuePairSeq* get_members() = 0;
    virtual void set_members(
        const NameValuePairSeq &  value
    ) = 0;
    virtual NameDynAnyPairSeq* get_members_as_dyn_any() = 0;
    virtual void set_members_as_dyn_any(
        const NameDynAnyPairSeq &  value
    ) = 0;

    static const IT_FWString _it_fw_type_id;
};
```

See Also DynamicAny::DynAny

See page 4 for descriptions of the standard helper functions:

- _duplicate()
- _narrow()
- _nil()
- _unchecked_narrow()

DynStruct::current_member_kind()

```
virtual ::CORBA::TCKind current_member_kind() = 0;
```

Returns the kind of TypeCode associated with the current position.

Exceptions

TypeMismatch	The DynStruct object represents an empty exception.
InvalidValue	The current position does not indicate a member.

See Also DynamicAny::DynAny::seek()
DynamicAny::DynStruct::current_member_name()

DynStruct::current_member_name()

```
virtual FieldName current_member_name() = 0;
```

Returns the name of the member at the current position. This method can return an empty value since the TypeCode of the value being manipulated may not contain the names of members.

Exceptions

TypeMismatch	DynStruct object represents an empty exception.
InvalidValue	The current position does not indicate a member.

See Also

DynamicAny::DynAny::seek()
DynamicAny::DynStruct::current_member_kind()

DynStruct::~DynStruct()

```
virtual ~DynStruct();
```

The destructor of a DynStruct object.

DynStruct::get_members()

```
virtual NameValuePairSeq* get_members() = 0;
```

Returns a sequence of members that describes the name and the value of each member in the structure (or exception) associated with a DynStruct object.

The sequence order is the same as the declaration order of members as indicated by the TypeCode of the DynStruct. The current position is not affected. The member names in the returned sequence will be empty strings if the TypeCode of the DynStruct does not contain member names.

See Also

DynamicAny::DynStruct::set_members()
DynamicAny::DynStruct::get_members_as_dyn_any()

DynStruct::get_members_as_dyn_any()

```
virtual NameDynAnyPairSeq* get_members_as_dyn_any() = 0;
```

Returns a sequence of name-DynAny pairs that describes each member in the structure (or exception) associated with a DynStruct object. Use this method instead of get_members() if you want to avoid converting DynAny objects to any objects when your application needs to handle DynStruct objects extensively.

The sequence order is the same as the declaration order of members as indicated by the TypeCode of the DynStruct. The current position is not affected. The member names in the returned sequence will be empty strings if the TypeCode of the DynStruct does not contain member names.

See Also

DynamicAny::DynStruct::set_members_as_dyn_any()
DynamicAny::DynStruct::get_members()

DynStruct::set_members()

```
virtual void set_members(
    const NameValuePairSeq &  value
) = 0;
```

Initializes the structure data associated with a DynStruct object from a sequence of name-value pairs.

Parameters

value	A sequence of name-value pairs representing member names and the values of the members.

The current position is set to zero if the sequence passed in has a non-zero length. The current position is set to -1 if an empty sequence is passed in.

Members in the sequence must follow these rules:

- Members must be in the order in which they appear in the IDL specification of the structure.
- If member names are supplied in the sequence, they must either match the corresponding member name in the TypeCode of the DynStruct or they must be empty strings.

- Members must be supplied in the same order as indicated by the TypeCode of the DynStruct. The method does not reassign member values based on member names.

Exceptions

InvalidValue	The sequence has a number of elements that disagrees with the number of members as indicated by the TypeCode of the DynStruct.
TypeMismatch	Raised if: • One or more sequence elements have a type that is not equivalent to the TypeCode of the corresponding member. • The member names do not match the corresponding member name in the TypeCode of the DynStruct.

See Also

DynamicAny::DynStruct::get_members()
DynamicAny::DynStruct::set_members_as_dyn_any()
DynamicAny::NameValuePairSeq

DynStruct::set_members_as_dyn_any()

```
virtual void set_members_as_dyn_any(
    const NameDynAnyPairSeq &  value
) = 0;
```

Initializes the structure data associated with a DynStruct object from a sequence of name-DynAny pairs. Use this method instead of set_members() if you want to avoid converting DynAny objects to any objects when your application needs to handle DynStruct objects extensively.

Parameters

value	A sequence of name-DynAny pairs representing member names and the values of the members as DynAny objects.

The current position is set to zero if the sequence passed in has a non-zero length. The current position is set to -1 if an empty sequence is passed in.

Members in the sequence must follow these rules:

- Members must be in the order in which they appear in the IDL specification of the structure.
- If member names are supplied in the sequence, they must either match the corresponding member name in the TypeCode of the DynStruct or they must be empty strings.
- Members must be supplied in the same order as indicated by the TypeCode of the DynStruct. The method does not reassign DynAny values based on member names.

Exceptions

InvalidValue	The sequence has a number of elements that disagrees with the number of members as indicated by the TypeCode of the DynStruct.
TypeMismatch	Raised if: • One or more sequence elements have a type that is not equivalent to the TypeCode of the corresponding member. • The member names do not match the corresponding member name in the TypeCode of the DynStruct.

See Also

DynamicAny::DynStruct::get_members_as_dyn_any()
DynamicAny::DynStruct::set_members()
DynamicAny::NameDynAnyPairSeq

DynamicAny::DynUnion Class

The DynUnion class lets you dynamically manage an Any value as a union value. This class inherits from the DynAny class. Key methods to manipulate a union include:

has_no_active_member()
member()
member_kind()
member_name()

Other methods are available to manipulate a union's discriminator:

discriminator_kind()
get_discriminator()
set_discriminator()
set_to_default_member()
set_to_no_active_member()

A union can have only two valid current positions: Zero denotes the discriminator and 1 denotes the active member.

The value returned by DynAny::component_count() for a union depends on the current discriminator: it is 2 for a union whose discriminator indicates a named member, and 1 otherwise.

```
class IT_DYNANY_API DynUnion : public virtual DynAny {
public:

    typedef DynamicAny::DynUnion_ptr _ptr_type;
    typedef DynamicAny::DynUnion_var _var_type;

    virtual ~DynUnion();
    static DynUnion_ptr _narrow(
        CORBA::Object_ptr obj
    );
    static DynUnion_ptr _unchecked_narrow(
        CORBA::Object_ptr obj
    );
    inline static DynUnion_ptr _duplicate(
        DynUnion_ptr p
```

```
    );
    inline static DynUnion_ptr _nil();

    virtual DynAny_ptr get_discriminator() = 0;
    virtual void set_discriminator(
        DynAny_ptr  d
    ) = 0;
    virtual void set_to_default_member() = 0;
    virtual void set_to_no_active_member() = 0;
    virtual CORBA::Boolean has_no_active_member() = 0;
    virtual ::CORBA::TCKind discriminator_kind() = 0;
    virtual DynAny_ptr member() = 0;
    virtual FieldName member_name() = 0;
    virtual ::CORBA::TCKind member_kind() = 0;

    static const IT_FWString _it_fw_type_id;
};
```

See Also DynamicAny::DynAny

See page 4 for descriptions of the standard helper functions:

- _duplicate()
- _narrow()
- _nil()
- _unchecked_narrow()

DynUnion::discriminator_kind()

```
virtual ::CORBA::TCKind discriminator_kind() = 0;
```

Returns the kind of TypeCode of the union's discriminator.

See Also DynamicAny::DynUnion::get_discriminator()
DynamicAny::DynUnion::set_discriminator()

DynUnion::~DynUnion()

```
virtual ~DynUnion();
```

The destructor for a DynUnion object.

DynUnion::get_discriminator()

```
virtual DynAny_ptr get_discriminator() = 0;
```

Returns the current discriminator value of the DynUnion.

See Also

DynamicAny::DynUnion::set_discriminator()
DynamicAny::DynUnion::discriminator_kind()

DynUnion::has_no_active_member()

```
virtual CORBA::Boolean has_no_active_member() = 0;
```

Returns true if the union has no active member (that is, the union's value consists solely of its discriminator because the discriminator has a value that is not listed as an explicit case label). The method returns false if:

- The IDL union has a default case.
- The IDL union's explicit case labels use the entire range of discriminator values.

See Also

DynamicAny::DynUnion::member()
DynamicAny::DynUnion::set_to_default_member()
DynamicAny::DynUnion::set_to_no_active_member()

DynUnion::member()

```
virtual DynAny_ptr member() = 0;
```

Returns the currently active member. Note that the returned reference remains valid only for as long as the currently active member does not change.

Parameters

InvalidValue	The union has no active member.
OBJECT_NOT_EXIST	The returned reference is used beyond the life time of the currently active member.

See Also

DynamicAny::DynUnion::member_kind()
DynamicAny::DynUnion::member_name()
DynamicAny::DynUnion::has_no_active_member()

DynUnion::member_kind()

```
virtual ::CORBA::TCKind member_kind() = 0;
```

Returns the kind of TypeCode of the currently active member.

Exceptions

InvalidValue The method is called on a union without an active member.

See Also

DynamicAny::DynUnion::member()
DynamicAny::DynUnion::member_name()

DynUnion::member_name()

```
virtual FieldName member_name() = 0;
```

Returns the name of the currently active member. The method returns an empty string if the union's TypeCode does not contain a member name for the currently active member.

Exceptions

InvalidValue The method is called on a union without an active member.

See Also

DynamicAny::DynUnion::member()
DynamicAny::DynUnion::member_kind()

DynUnion::set_discriminator()

```
virtual void set_discriminator(
    DynAny_ptr  d
) = 0;
```

Sets the discriminator of the DynUnion.

Parameters

d	The value to set the discriminator to. Setting the discriminator to a value that is consistent with the currently active union member does not affect the currently active member. Setting the discriminator to a value that is inconsistent with the currently active member deactivates the member and activates the member that is consistent with the new discriminator value (if there is a member for that value) by initializing the member to its default value.

Setting the discriminator of a union sets the current position to 0 if the discriminator value indicates a non-existent union member (The method has_no_active_member() would return true in this case). Otherwise, if the discriminator value indicates a named union member, the current position is set to 1, has_no_active_member() would return false, and component_count() would return 2 in this case.

Exceptions

TypeMismatch	The TypeCode of the parameter is not equivalent to the TypeCode of the union's discriminator.

See Also

DynamicAny::DynUnion::get_discriminator()
DynamicAny::DynUnion::has_no_active_member()
DynamicAny::DynUnion::set_to_default_member()
DynamicAny::DynUnion::set_to_no_active_member()

DynUnion::set_to_default_member()

```
virtual void set_to_default_member() = 0;
```

Sets the discriminator to a value that is consistent with the value of the default case of a union.

This method sets the current position to zero and causes component_count() to return 2.

Exceptions

TypeMismatch	The method is called on a union without an explicit default case.

See Also

DynamicAny::DynUnion::has_no_active_member()

DynamicAny::DynUnion::set_discriminator()
DynamicAny::DynUnion::set_to_no_active_member()
DynamicAny::DynUnion::set_to_no_active_member()

DynUnion::set_to_no_active_member()

```
virtual void set_to_no_active_member() = 0;
```

Sets the discriminator to a value that does not correspond to any of the union's case labels.

This method sets the current position to zero and causes DynAny::component_count() to return 1.

Exceptions

TypeMismatch	Raised if this method is called on a union that: • Does not have an explicit default case. • Uses the entire range of discriminator values for explicit case labels.

See Also

DynamicAny::DynUnion::has_no_active_member()
DynamicAny::DynUnion::set_discriminator()
DynamicAny::DynUnion::set_to_default_member()

DynamicAny::DynValue Class

You use DynValue objects for dynamically handling value types in Any values. Value types are used for objects-by-value. This class inherits from the DynAny class. Key methods allow you to set and get the value type as a sequence of name-value pairs:

get_members()
set_members()
get_members_as_dyn_any()
set_members_as_dyn_any()

Use the DynAny iteration methods such as seek() to set the current position to a member of the value type. You can also obtain the name and kind of TypeCode for a member at the current position:

current_member_name()
current_member_kind()

The class is as follows:

```
// class is in namespace DynamicAny
class IT_DYNANY_API DynValue : public virtual DynAny {
public:
    typedef DynamicAny::DynValue_ptr _ptr_type;
    typedef DynamicAny::DynValue_var _var_type;

    virtual ~DynValue();
    static DynValue_ptr _narrow(
        CORBA::Object_ptr obj
    );
    static DynValue_ptr _unchecked_narrow(
        CORBA::Object_ptr obj
    );
    inline static DynValue_ptr _duplicate(
        DynValue_ptr p
    );
    inline static DynValue_ptr _nil();

    virtual FieldName current_member_name() = 0;
    virtual ::CORBA::TCKind current_member_kind() = 0;
```

```
    virtual NameValuePairSeq* get_members() = 0;
    virtual void set_members(
        const NameValuePairSeq &  values
    ) = 0;
    virtual NameDynAnyPairSeq* get_members_as_dyn_any() = 0;
    virtual void set_members_as_dyn_any(
        const NameDynAnyPairSeq &  value
    ) = 0;

    static const IT_FWString _it_fw_type_id;
};
```

See Also DynamicAny::DynAny

See page 4 for descriptions of the standard helper functions:

- _duplicate()
- _narrow()
- _nil()
- _unchecked_narrow()

DynValue::current_member_kind()

```
virtual ::CORBA::TCKind current_member_kind() = 0;
```

Returns the kind of TypeCode associated with the current position.

Exceptions

TypeMismatch	The DynValue object represents an empty value type.
InvalidValue	The current position does not indicate a member.

See Also DynamicAny::DynAny::seek()
DynamicAny::DynValue::current_member_name()

DynValue::current_member_name()

```
virtual FieldName current_member_name() = 0;
```

Returns the name of the member at the current position. This method can return an empty value since the TypeCode of the value being manipulated may not contain the names of members.

Exceptions

TypeMismatch	The DynValue object represents an empty value type.
InvalidValue	The current position does not indicate a member.

See Also

DynamicAny::DynAny::seek()
DynamicAny::DynValue::current_member_kind()

DynValue:: ~DynValue()

```
virtual ~DynValue();
```

The destructor for a DynValue object.

DynValue::get_members()

```
virtual NameValuePairSeq* get_members() = 0;
```

Returns a sequence of members that describes the name and the value of each member in the DynValue object.

The sequence order is the same as the declaration order of members as indicated by the TypeCode of the DynValue. The current position is not affected. The member names in the returned sequence will be empty strings if the TypeCode of the DynValue does not contain member names.

See Also

DynamicAny::DynValue::set_members()
DynamicAny::DynValue::get_members_as_dyn_any()

DynValue::get_members_as_dyn_any()

```
virtual NameDynAnyPairSeq* get_members_as_dyn_any() = 0;
```

Returns a sequence of name-DynAny pairs that describes each member in the value type associated with a DynValue object. Use this method instead of get_members() if you want to avoid converting DynAny objects to Any objects when your application needs to handle DynValue objects extensively.

The sequence order is the same as the declaration order of members as indicated by the TypeCode of the DynValue. The current position is not affected. The member names in the returned sequence will be empty strings if the TypeCode of the DynValue does not contain member names.

See Also

DynamicAny::DynValue::set_members_as_dyn_any()
DynamicAny::DynValue::get_members()

DynValue::set_members()

```
virtual void set_members(
    const NameValuePairSeq &  values
) = 0;
```

Initializes the data value associated with a DynValue object from a sequence of name-value pairs.

Parameters

values	A sequence of name-value pairs representing member names and the values of the members.

The current position is set to zero if the sequence passed in has a non-zero length. The current position is set to -1 if an empty sequence is passed in.

Members in the sequence must follow these rules:

- Members must be in the order in which they appear in the IDL specification.
- If member names are supplied in the sequence, they must either match the corresponding member name in the TypeCode of the DynValue or they must be empty strings.
- Members must be supplied in the same order as indicated by the TypeCode of the DynValue. The method does not reassign member values based on member names.

Exceptions

InvalidValue	The sequence has a number of elements that disagrees with the number of members as indicated by the TypeCode of the DynValue.
TypeMismatch	Raised if: • One or more sequence elements have a type that is not equivalent to the TypeCode of the corresponding member. • The member names do not match the corresponding member name in the TypeCode of the DynValue.

See Also

DynamicAny::DynValue::get_members()
DynamicAny::DynValue::set_members_as_dyn_any()
DynamicAny::NameValuePairSeq

DynValue::set_members_as_dyn_any()

```
virtual void set_members_as_dyn_any(
    const NameDynAnyPairSeq &  value
) = 0;
```

Initializes the data value associated with a DynValue object from a sequence of name-DynAny pairs. Use this method instead of set_members() if you want to avoid converting DynAny objects to any objects when your application needs to handle DynValue objects extensively.

Parameters

value	A sequence of name-DynAny pairs representing member names and the values of the members as DynAny objects.

The current position is set to zero if the sequence passed in has a non-zero length. The current position is set to -1 if an empty sequence is passed in.

Members in the sequence must follow these rules:

- Members must be in the order in which they appear in the IDL specification of the structure.
- If member names are supplied in the sequence, they must either match the corresponding member name in the TypeCode of the DynValue or they must be empty strings.

- Members must be supplied in the same order as indicated by the TypeCode of the DynValue. The method does not reassign DynAny values based on member names.

Exceptions

InvalidValue	The sequence has a number of elements that disagrees with the number of members as indicated by the TypeCode of the DynValue.
TypeMismatch	Raised if: • One or more sequence elements have a type that is not equivalent to the TypeCode of the corresponding member. • The member names do not match the corresponding member name in the TypeCode of the DynValue.

See Also

DynamicAny::DynValue::get_members_as_dyn_any()
DynamicAny::DynValue::set_members()
DynamicAny::NameDynAnyPairSeq

IT_Logging Module

IT_Logging Overview

The IT_Logging module is the centralized point for controlling all logging methods.

- The EventLog interface controls the reporting of log events.
- The LogStream interface controls how and where events are received.

The IT_Logging module also uses the following common data types, static method, and macros.

Table 12: *IT_Logging Common Data Types, Methods, and Macros*

Common Data Types	Methods and Macros
ApplicationId EventId EventParameters EventPriority SubsystemId Timestamp	format_message() IT_LOG_MESSAGE() IT_LOG_MESSAGE_1() IT_LOG_MESSAGE_2() IT_LOG_MESSAGE_3() IT_LOG_MESSAGE_4() IT_LOG_MESSAGE_5()

IT_Logging::ApplicationId Data Type

```
//IDL
typedef string ApplicationId;
```

An identifying string representing the application that logged the event.

For example, a Unix and Windows ApplicationId contains the host name and process ID (PID) of the reporting process. Because this value can differ from platform to platform, streams should only use it as informational text, and should not attempt to interpret it.

Enhancement Orbix enhancement to CORBA.

IT_Logging::EventId Data Type

```
//IDL
typedef unsigned long EventId;
```

An identifier for the particular event.

Enhancement Orbix enhancement to CORBA.

IT_Logging::EventParameters Data Type

```
//IDL
typedef CORBA::AnySeq  EventParameters;
```

A sequence of locale-independent parameters encoded as a sequence of `Any` values.

Enhancement Orbix enhancement to CORBA.

See Also `IT_Logging::format_message()`

IT_Logging::EventPriority Data Type

```
//IDL
typedef unsigned short EventPriority;
```

Specifies the priority of a logged event. These can be divided into the following categories of priority.

Information	A significant non-error event has occurred. Examples include server startup/shutdown, object creation/deletion, and information about administrative actions. Informational messages provide a history of events that can be invaluable in diagnosing problems.
Warning	The subsystem has encountered an anomalous condition, but can ignore it and continue functioning. Examples include encountering an invalid parameter, but ignoring it in favor of a default value.

Error	An error has occurred. The subsystem will attempt to recover, but may abandon the task at hand. Examples include finding a resource (such as memory) temporarily unavailable, or being unable to process a particular request due to errors in the request.
Fatal Error	An unrecoverable error has occurred. The subsystem or process will terminate.

The possible values for an `EventPriority` consist of the following:

```
LOG_NO_EVENTS
LOG_ALL_EVENTS
LOG_INFO_LOW
LOG_INFO_MED
LOG_INFO_HIGH
LOG_INFO  (LOG_INFO_LOW)
LOG_ALL_INFO

LOG_WARNING
LOG_ERROR
LOG_FATAL_ERROR
```

A single value is used for `EventLog` operations that report events or `LogStream` operations that receive events. In filtering operations such as `set_filter()`, these values can be combined as a filter mask to control which events are logged at runtime.

Enhancement Orbix enhancement to CORBA.

IT_Logging::format_message()

```
// C++
static char* format_message(
    const char* description,
    const IT_Logging::EventParameters& params
);
```

Returns a formatted message based on a format description and a sequence of parameters.

Parameters Messages are reported in two pieces for internationalization:

`description`	A locale-dependent string that describes of how to use the sequence of parameters in `params`.
`params`	A sequence of locale-dependent parameters.

`format_message()` copies the `description` into an output string, interprets each event parameter, and inserts the event parameters into the output string where appropriate. Event parameters that are primitive and `SystemException` parameters are converted to strings before insertion. For all other types, question marks (?) are inserted.

Enhancement Orbix enhancement to CORBA.

IT_Logging::SubsystemId Data Type

```
//IDL
typedef string SubsystemId;
```

An identifying string representing the subsystem from which the event originated. The constant `_DEFAULT` may be used to enable all subsystems.

Enhancement Orbix enhancement to CORBA.

IT_Logging::Timestamp Data Type

```
//IDL
typedef unsigned long  Timestamp;
```

The time of the logged event in seconds since January 1, 1970.

Enhancement Orbix enhancement to CORBA.

IT_LOG_MESSAGE() Macro

```
// C++
#define IT_LOG_MESSAGE( \
    event_log,  \
    subsystem, \
    id, \
```

```
    severity, \
    desc \
) ...
```

A macro to use for reporting a log message.

Parameters

`event_log`	The log (`EventLog`) where the message is to be reported.
`subsystem`	The `SubsystemId`.
`id`	The `EventId`.
`severity`	The `EventPriority`.
`desc`	A string description of the event.

Enhancement Orbix enhancement to CORBA.

Examples Here is a simple example of usage:

```
...
IT_LOG_MESSAGE(
    event_log,
    IT_IIOP_Logging::SUBSYSTEM,
    IT_IIOP_Logging::SOCKET_CREATE_FAILED,
    IT_Logging::LOG_ERROR,
    SOCKET_CREATE_FAILED_MSG
);
```

IT_LOG_MESSAGE_1() Macro

```
// C++
#define IT_LOG_MESSAGE_1( \
    event_log, \
    subsystem, \
    id, \
    severity,  \
    desc, \
    param0 \
) ...
```

A macro to use for reporting a log message with one event parameter.

Parameters

`event_log`	The log (`EventLog`) where the message is to be reported.
`subsystem`	The `SubsystemId`.
`id`	The `EventId`.
`severity`	The `EventPriority`.
`desc`	A string description of the event.
`param0`	A single parameter for an `EventParameters` sequence.

Enhancement Orbix enhancement to CORBA.

See Also `IT_Logging::IT_LOG_MESSAGE()`

IT_LOG_MESSAGE_2() Macro

```
// C++
#define IT_LOG_MESSAGE_2( \
    event_log, \
    subsystem, \
    id, \
    severity, \
    desc, \
    param0, \
    param1 \
) ...
```

A macro to use for reporting a log message with two event parameters.

Parameters

`event_log`	The log (`EventLog`) where the message is to be reported.
`subsystem`	The `SubsystemId`.
`id`	The `EventId`.
`severity`	The `EventPriority`.
`desc`	A string description of the event.
`param0`	The first parameter for an `EventParameters` sequence.
`param1`	The second parameter for an `EventParameters` sequence.

Enhancement Orbix enhancement to CORBA.

See Also IT_Logging::IT_LOG_MESSAGE()

IT_LOG_MESSAGE_3() Macro

```
// C++
#define IT_LOG_MESSAGE_3( \
    event_log, \
    subsystem, \
    id, \
    severity, \
    desc, \
    param0, \
    param1, \
    param2 \
) ...
```

A macro to use for reporting a log message with three event parameters.

Parameters

event_log	The log (EventLog) where the message is to be reported.
subsystem	The SubsystemId.
id	The EventId.
severity	The EventPriority.
desc	A string description of the event.
param0	The first parameter for an EventParameters sequence.
param1	The second parameter for an EventParameters sequence.
param2	The third parameter for an EventParameters sequence.

Enhancement Orbix enhancement to CORBA.

See Also IT_Logging::IT_LOG_MESSAGE()

IT_LOG_MESSAGE_4() Macro

```
// C++
#define IT_LOG_MESSAGE_4( \
    event_log, \
    subsystem, \
```

```
    id, \
    severity, \
    desc, \
    param0, \
    param1, \
    param2, \
    param3 \
) ...
```

A macro to use for reporting a log message with four event parameters.

Parameters

`event_log`	The log (`EventLog`) where the message is to be reported.
`subsystem`	The `SubsystemId`.
`id`	The `EventId`.
`severity`	The `EventPriority`.
`desc`	A string description of the event.
`param0`	The first parameter for an `EventParameters` sequence.
`param1`	The second parameter for an `EventParameters` sequence.
`param2`	The third parameter for an `EventParameters` sequence.
`param3`	The forth parameter for an `EventParameters` sequence.

Enhancement Orbix enhancement to CORBA.

See Also `IT_Logging::IT_LOG_MESSAGE()`

IT_LOG_MESSAGE_5() Macro

```
// C++
#define IT_LOG_MESSAGE_5( \
    event_log, \
    subsystem, \
    id, \
    severity, \
    desc, \
    param0, \
    param1, \
    param2, \
    param3, \
```

```
    param4 \
) ...
```

A macro to use for reporting a log message with five event parameters.

Parameters

`event_log`	The log (`EventLog`) where the message is to be reported.
`subsystem`	The `SubsystemId`.
`id`	The `EventId`.
`severity`	The `EventPriority`.
`desc`	A string description of the event.
`param0`	The first parameter for an `EventParameters` sequence.
`param1`	The second parameter for an `EventParameters` sequence.
`param2`	The third parameter for an `EventParameters` sequence.
`param3`	The forth parameter for an `EventParameters` sequence.
`param4`	The fifth parameter for an `EventParameters` sequence.

Enhancement Orbix enhancement to CORBA.

See Also `IT_Logging::IT_LOG_MESSAGE()`

IT_Logging::EventLog Interface

Logging is controlled with the EventLog interface, which defines operations to register interfaces for receiving notification of logged events, report logged events, and filter logged events. Each ORB maintains its own EventLog instance, which applications obtain by calling resolve_initial_references() with the string argument IT_EventLog.

The EventLog interface has the following operations:

- register_stream() registers the receivers of log events. report_event() reports log events and report_message() reports messages to receivers.
- get_filter(), set_filter(), expand_filter(), and clear_filter() set filters for which log events are reported.

An EventLog has several operations for controlling which events are logged at runtime. A filter has an EventPriority that describes the types of events that are reported. Every subsystem is associated with a filter that controls which events are allowed for that subsystem. A default filter is also associated with the entire EventLog.

The complete EventLog interface is as follows:

```
// IDL in module IT_Logging
interface EventLog {
    void register_stream(
        in LogStream the_stream
    );

    void report_event(
        in SubsystemId   subsystem,
        in EventId       event,
        in EventPriority priority,
        in any           event_data
    );

    void report_message(
        in SubsystemId     subsystem,
        in EventId         event,
```

```
            in EventPriority   priority,
            in string          description,
            in EventParameters parameters
        );

        EventPriority get_filter(
            in SubsystemId subsystem
        );

        void set_filter(
            in SubsystemId   subsystem,
            in EventPriority filter_mask
        );

        void expand_filter(
            in SubsystemId   subsystem,
            in EventPriority filter_mask
        );

        void clear_filter(
            in SubsystemId subsystem
        );
    ...
    };
```

EventLog::clear_filter()

```
// IDL
void clear_filter(
    in SubsystemId subsystem
);
```

Removes an explicitly configured subsystem filter, causing the subsystem to revert to using the default filter.

Enhancement Orbix enhancement to CORBA.

See Also IT_Logging::EventLog::get_filter()

EventLog::expand_filter()

```
// IDL
void expand_filter(
    in SubsystemId subsystem,
    in EventPriority filter_mask
);
```

Adds to a subsystem filter by combining the new filter mask with the existing subsystem filter.

Parameters

`subsystem`	The name of the subsystem for which the filter applies.
`filter_mask`	A value representing the types of events to be reported.

Enhancement Orbix enhancement to CORBA.

See Also `IT_Logging::EventLog::set_filter()`
`IT_Logging::EventLog::clear_filter()`

EventLog::get_filter()

```
// IDL
EventPriority get_filter(
    in SubsystemId subsystem
);
```

Returns a sub-system's filter priorities.

Parameters

`subsystem`	The name of the subsystem for which the filter applies.

Enhancement Orbix enhancement to CORBA.

See Also `IT_Logging::EventLog::get_filter()`

EventLog::register_stream()

```
// IDL
void register_stream(
```

```
    in LogStream the_stream
);
```

Explicitly registers a LogStream.

Parameters

the_stream	The stream to register.

Log events "flow" to receivers on streams, thus streams must be registered with the EventLog. Once registered, the stream will receive notification of logged events.

An EventLog can have multiple streams registered at one time, and it can have a single stream registered more than once.

Enhancement Orbix enhancement to CORBA.

See Also IT_Logging::LogStream

EventLog::report_event()

```
// IDL
void report_event(
    in SubsystemId   subsystem,
    in EventId       event,
    in EventPriority priority,
    in any           event_data
);
```

Reports an event and its event-specific data.

Parameters

subsystem	The name of the subsystem reporting the event.
event	The unique ID defining the event.
priority	The event priority.
event_data	Event-specific data.

Enhancement Orbix enhancement to CORBA.

See Also IT_Logging::EventLog::report_message()

EventLog::report_message()

```
// IDL
void report_message(
    in SubsystemId     subsystem,
    in EventId         event,
    in EventPriority   priority,
    in string          description,
    in EventParameters parameters
);
```

Reports an event and message.

Parameters

`subsystem`	The name of the subsystem reporting the event.
`event`	The unique ID defining the event.
`priority`	The event priority.
`description`	A string describing the format of `parameters`.
`parameters`	A sequence of parameters for the log.

Enhancement Orbix enhancement to CORBA.

See Also `IT_Logging::EventLog::report_event()`

EventLog::set_filter()

```
// IDL
void set_filter(
    in SubsystemId subsystem,
    in EventPriority  filter_mask
);
```

Sets a filter for a given subsystem. This operation overrides the subsystem's existing filter.

Parameters

`subsystem`	The name of the subsystem for which the filter applies.
`filter_mask`	A value representing the types of events to be reported.

A subsystem will use the default filter if its filter has not been explicitly configured by a call to `set_filter()`.

Enhancement Orbix enhancement to CORBA.

See Also `IT_Logging::EventLog::get_filter()`

IT_Logging::LogStream Interface

The LogStream interface allows an application to intercept events and write them to some concrete location via a stream. IT_Logging::EventLog objects maintain a list of LogStream objects. You register a LogStream object from an EventLog using register_stream(). The complete LogStream interface is as follows:

```
// IDL in module IT_Logging
interface LogStream {
    void report_event(
        in ApplicationId   application,
        in SubsystemId     subsystem,
        in EventId         event,
        in EventPriority   priority,
        in Timestamp       event_time,
        in any             event_data
    );

    void report_message(
        in ApplicationId   application,
        in SubsystemId     subsystem,
        in EventId         event,
        in EventPriority   priority,
        in Timestamp       event_time,
        in string          description,
        in EventParameters parameters
    );
};
```

These operations are described in detail as follows:

LogStream::report_event()

```
// IDL
void report_event(
    in ApplicationId   application,
    in SubsystemId     subsystem,
```

```
    in EventId          event,
    in EventPriority    priority,
    in Timestamp        event_time,
    in any              event_data
);
```

Reports an event and its event-specific data to the log stream.

Parameters

`application`	An ID representing the reporting application.
`subsystem`	The name of the subsystem reporting the event.
`event`	A unique ID defining the event.
`priority`	The event priority.
`event_time`	The time when the event occurred.
`event_data`	Event-specific data.

Enhancement Orbix enhancement to CORBA.

See Also `IT_Logging::EventLog::report_event()`
`IT_Logging::LogStream::report_message()`

LogStream::report_message()

```
// IDL
void report_message(
    in ApplicationId    application,
    in SubsystemId      subsystem,
    in EventId          event,
    in EventPriority    priority,
    in Timestamp        event_time,
    in string           description,
    in EventParameters  parameters
);
```

Reports an event and message to the log stream.

Parameters

`application`	An ID representing the reporting application.
`subsystem`	The name of the subsystem reporting the event.

`event`	The unique ID defining the event.
`priority`	The event priority.
`event_time`	The time when the event occurred.
`description`	A string describing the format of `parameters`.
`parameters`	A sequence of parameters for the log.

Enhancement Orbix enhancement to CORBA.

See Also `IT_Logging::EventLog::report_message()`
`IT_Logging::LogStream::report_event()`

Messaging Module

Messaging Overview

CORBA provides synchronous and deferred synchronous modes of invocations. The Messaging module provides the additional asynchronous mode, also known here as *Asynchronous Method Invocation* (*AMI*). The Messaging module includes the following base classes, value types, policy classes, common data structures, and constants:

Table 13: *The Messaging Module*

Base Classes and Value Types	Common Structures and Constants	QoS Policy Classes
ExceptionHolder ReplyHandler	INVOCATION_POLICIES RebindMode RoutingType RoutingTypeRange SyncScope TAG_POLICIES	RebindPolicy RoutingPolicy SyncScopePolicy

With synchronous invocations, the client program, or thread, blocks when a remote invocation is made and waits until the results arrive. With deferred synchronous invocations, the client thread continues processing, subsequently polling to see if results are available. Within the CORBA module, the deferred synchronous model is only available when using the Dynamic Invocation Interface.

Many applications require some way of managing remote requests within an asynchronous, event-driven environment in which callbacks are invoked to handle events. Sophisticated applications often need to manage several activities simultaneously, making overlapping remote requests to many objects. This can be achieved using a separate thread for each invocation, but the use of threads considerably raises the application's complexity and the probability of programming errors. The use of threads also creates a resource and synchronization problem in addition to the memory management problem inherent in asynchronous communications.

Messaging provides the *callback model* in which the client passed a callback object reference as part of the invocation. When the reply is available, that callback object is invoked with the data of the reply. The callback model uses a ReplyHandler, which is a CORBA object, implemented by the client application. The ReplyHandler is passed to an asynchronous method invocation. The ReplyHandler is invoked when the reply to that request is available.

The Messaging module also provides a QoS property to help obtain asynchronous behavior. The Messaging QoS includes some CORBA::Policy derived interfaces for client-side policies to control the behavior of requests and replies. Note however that QoS for method invocations applies to both asynchronous and synchronous invocations. See also the discussion "Quality of Service Framework".

The following constants and types are available for messaging.

Messaging::INVOCATION_POLICIES Constant

```
IT_ART_API IT_NAMESPACE_STATIC
    const CORBA::ULong INVOCATION_POLICIES;
```

A service context containing a sequence of quality of service policies in effect for the invocation. The quality of service framework abstract model includes this mechanism for transporting Policy values as part of interoperable object references and within requests.

Messaging::RebindMode Type

```
typedef     CORBA::Short RebindMode;
typedef     CORBA::Short_out RebindMode_out;
IT_ART_API IT_NAMESPACE_STATIC CORBA::TypeCode_ptr _tc_RebindMode;
```

This describes the level of transparent rebinding that may occur during the course of an invocation on an object. Values of type RebindMode are used in conjunction with a RebindPolicy. All non-negative values are reserved for use in OMG specifications and include the following constants:

TRANSPARENT	Allows the ORB to silently handle object-forwarding and necessary reconnection during the course of making a remote request.

NO_REBIND	Allows the ORB to silently handle reopening of closed connections while making a remote request, but prevents any transparent object-forwarding that would cause a change in client-visible effective QoS policies. When the RebindPolicy has this mode in effect, only explicit rebinding is allowed by calling CORBA::Object::_validate_connection().
NO_RECONNECT	Prevents the ORB from silently handling object-forwards or the reopening of closed connections. When the RebindPolicy has this mode in effect, only explicit rebinding is allowed by calling CORBA::Object::_validate_connection().

Any negative value for a RebindMode is considered a vendor extension.

See Also Messaging::RebindPolicy

Messaging::RoutingType Type

```
typedef     CORBA::Short RoutingType;
typedef     CORBA::Short_out RoutingType_out;
IT_ART_API IT_NAMESPACE_STATIC CORBA::TypeCode_ptr
   _tc_RoutingType;
```

Describes the type of routing to be used for invocations on an object reference. RoutingType values are used in conjunction with a RoutingPolicy. All non-negative values are reserved for use in OMG specifications and include the following constants:

ROUTE_NONE	Synchronous or deferred synchronous delivery is used. No routers will be used to aid in the delivery of the request.
ROUTE_FORWARD	Asynchronous delivery is used. The request is made through the use of a router and not delivered directly to the target by the client ORB.
ROUTE_STORE_AND_FORWARD	Asynchronous TII is used. The request is made through the use of a router that persistently stores the request before attempting delivery.

Any negative value for a RoutingType is considered a vendor extension.

See Also Messaging::RoutingTypeRange

Messaging::RoutingTypeRange Structure

```
struct RoutingTypeRange;
typedef ITCxxFixLenConstr_var< RoutingTypeRange>
   RoutingTypeRange_var;
typedef RoutingTypeRange& RoutingTypeRange_out;

struct RoutingTypeRange {
    typedef RoutingTypeRange_var _var_type;
    ::Messaging::RoutingType min;
    ::Messaging::RoutingType max;
};
IT_ART_API IT_NAMESPACE_STATIC CORBA::TypeCode_ptr
   _tc_RoutingTypeRange;
```

This structure describes a range of routing types. It is invalid for the minimum RoutingType to be greater than the maximum RoutingType.

Messaging::SyncScope Type

```
typedef     CORBA::Short SyncScope;
typedef     CORBA::Short_out SyncScope_out;
IT_ART_API IT_NAMESPACE_STATIC CORBA::TypeCode_ptr _tc_SyncScope;
```

Describes the level of synchronization for a request with respect to the target. Values of type SyncScope are used in conjunction with a SyncScopePolicy to control the behavior of one way operations. All non-negative values are reserved

for use in OMG specifications. Any negative value of SyncScope is considered a vendor extension. Valid values include:

SYNC_NONE	This is equivalent to one allowable interpretation of CORBA 2.2 oneway operations. The ORB returns control to the client (that is, returns from the method invocation) before passing the request message to the transport protocol. The client is guaranteed not to block. You cannot do location-forwarding with this level of synchronization because no reply is returned from the server.
SYNC_WITH_TRANSPORT	This is equivalent to one allowable interpretation of CORBA 2.2 oneway operations. The ORB returns control to the client only after the transport has accepted the request message. This gives no guarantee that the request will be delivered, but in conjunction with knowledge of the transport it may provide the client with enough assurance. For example, for a direct message over TCP, SYNC_WITH_TRANSPORT is not a stronger guarantee than SYNC_NONE. However, for a store and forward transport, this QoS provides a high level of reliability. You cannot do location-forwarding with this level of synchronization because no reply is returned from the server.
SYNC_WITH_SERVER	The server-side ORB shall send a reply before invoking the target implementation. If a reply of NO_EXCEPTION is sent, any necessary location-forwarding has already occurred. Upon receipt of this reply, the client-side ORB returns control to the client application. This form of guarantee is useful where the reliability of the network is substantially lower than that of the server. The client blocks until all location-forwarding has been completed. For a server using a POA, the reply would be sent after invoking any ServantManager, but before delivering the request to the target Servant.

SYNC_WITH_TARGET	Equivalent to a synchronous, non-oneway operation in CORBA 2.2. The server-side ORB shall only send the reply message after the target has completed the invoked operation. Note that any LOCATION_FORWARD reply will already have been sent prior to invoking the target and that a SYSTEM_EXCEPTION reply may be sent at anytime (depending on the semantics of the exception). Even though it was declared oneway, the operation actually has the behavior of a synchronous operation. This form of synchronization guarantees that the client knows that the target has seen and acted upon a request. the OTS can only be used with this highest level of synchronization. Any operations invoked with lesser synchronization precludes the target from participating in the client's current transaction.

See Also Messaging::SyncScopePolicy

Messaging::TAG_POLICIES Constant

```
IT_ART_API IT_NAMESPACE_STATIC const CORBA::ULong TAG_POLICIES;
```

A profile component containing the sequence of quality of service policies exported with the object reference by an object adapter. The quality of service framework abstract model includes this mechanism for transporting policy values as part of interoperable object references and within requests.

See Also Messaging::RoutingPolicy

Messaging::ExceptionHolder Value Type

The messaging callback model uses an `ExceptionHolder` to deliver exceptions. Because the `ReplyHandler` implements an IDL interface, all arguments passed to its operations must be defined in IDL also. However, exceptions cannot be passed as arguments to operations, but are only raised as part of a reply. An `ExceptionHolder` value is created to encapsulate the identity and contents of the exception that might be raised. An instance of this `ExceptionHolder` is passed as the argument to the `ReplyHandler` operation that indicates an exception was raised by the target. In addition to its exception state, the `ExceptionHolder` also has operations that raise the returned exception, so the `ReplyHandler` implementation can have the returned exception re-raised within its own context.

AMI operations do not raise user exceptions. Rather, user exceptions are passed to the implemented type specific `ReplyHandler`. If an AMI operation raises a system exception with a completion status of `COMPLETED_NO`, the request has not been made. This clearly distinguishes exceptions raised by the server (which are returned via the `ReplyHandler`) from the local exceptions that caused the AMI to fail.

The `ExceptionHolder` value class implementation is provided by the ORB. For each interface, a type specific `ExceptionHolder` value is generated by the IDL compiler. This `ExceptionHolder` is implemented by the ORB and passed to an application using the callback model when exception replies are returned from the target. See the *CORBA Programmer's Guide* for more on the generated value types and operations.

The code is as follows:

```
...
class IT_ART_API ExceptionHolder : public virtual CORBA::ValueBase
{
public:
    virtual CORBA::Any* get_exception() = 0;
```

```
        virtual CORBA::Any* get_exception_with_list(
            ::CORBA::ExceptionList_ptr  exc_list
        ) = 0;

        typedef ITCxxUFixedSeq< CORBA::Octet >
       _marshaled_exception_seq;

        ...

        static ExceptionHolder* _downcast(
            CORBA::ValueBase* _val
        );

        ...

    protected:
        ExceptionHolder();
        ExceptionHolder(
            CORBA::Boolean _itfld_is_system_exception,
            CORBA::Boolean _itfld_byte_order,
            const ITCxxUFixedSeq< CORBA::Octet > &
       _itfld_marshaled_exception
        );

        virtual ~ExceptionHolder();

        virtual CORBA::Boolean is_system_exception() const = 0;
        virtual void is_system_exception(
            CORBA::Boolean
        ) = 0;

        virtual CORBA::Boolean byte_order() const = 0;
        virtual void byte_order(
            CORBA::Boolean
        ) = 0;

        virtual void marshaled_exception(
            const  _marshaled_exception_seq&
        ) = 0;
        virtual const  _marshaled_exception_seq &
            marshaled_exception() const = 0;
        virtual  _marshaled_exception_seq & marshaled_exception() = 0;
    private:
```

```
    ...
};
```

Enhancement The `ExceptionHolder` class is not compliant with the CORBA Messaging specification.

ExceptionHolder::byte_order()

```
virtual CORBA::Boolean byte_order() const = 0;
```

Returns the byte order for the exception.

```
virtual void byte_order(
    CORBA::Boolean
) = 0;
```

Sets the byte order for the exception.

ExceptionHolder::_downcast()

```
static ExceptionHolder* _downcast(
    CORBA::ValueBase* _val
);
```

Returns a pointer to the `ExceptionHolder` type for a derived class. Each value type class provides `_downcast()` as a portable way for applications to cast down the C++ inheritance hierarchy.

Parameters

`_val` Pointer to the value type class to be downcast.

- If the value type instance pointed to by the argument is an instance of the value type class being downcast to, a pointer to the downcast-to class type is returned.
- If the value type instance pointed to by the argument is not an instance of the value type class being downcast to, a null pointer is returned.
- If a null pointer is passed to `_downcast()`, it returns a null pointer.

This is especially required after an invocation of `_copy_value()`.

Enhancement Orbix enhancement.

See Also CORBA::ValueBase::_copy_value()

ExceptionHolder::ExceptionHolder() Constructors

```
ExceptionHolder();

ExceptionHolder(
    CORBA::Boolean _itfld_is_system_exception,
    CORBA::Boolean _itfld_byte_order,
    const ITCxxUFixedSeq< CORBA::Octet > &
   _itfld_marshaled_exception
);
```

Constructors for the `ExceptionHolder`.

Enhancement Orbix enhancement.

ExceptionHolder::~ExceptionHolder() Destructor

```
virtual ~ExceptionHolder();
```

The destructor for the `ExceptionHolder`.

Enhancement Orbix enhancement.

ExceptionHolder::get_exception()

```
virtual CORBA::Any* get_exception() = 0;
```

Returns the exception.

See Also Messaging::ExceptionHolder::get_exception_with_list()

Enhancement Orbix enhancement.

ExceptionHolder::get_exception_with_list()

```
virtual CORBA::Any* get_exception_with_list(
    ::CORBA::ExceptionList_ptr  exc_list
) = 0;
```

Returns a list of exceptions.

Enhancement Orbix enhancement.

See Also Messaging::ExceptionHolder::get_exception()

ExceptionHolder::is_system_exception()

```
virtual CORBA::Boolean is_system_exception() const = 0;
virtual void is_system_exception(
    CORBA::Boolean
) = 0;
```

ExceptionHolder::_it_demarshal_value()

```
virtual void _it_demarshal_value(
    CORBA::IT_InStream_ptr _is,
    CORBA::ORB_ptr _orb
);
```

Note: For internal use only.

ExceptionHolder::_it_get_fw_type_id()

```
static const IT_FWString& _it_get_fw_type_id();
```

Note: For internal use only.

ExceptionHolder::_it_get_safe_bases()

```
const char** _it_get_safe_bases() const;
```

Note: For internal use only.

ExceptionHolder::_it_marshal_value()

```
virtual void _it_marshal_value(
    CORBA::IT_OutStream_ptr _os,
    CORBA::ORB_ptr _orb
);
```

Note: For internal use only.

ExceptionHolder::_it_type()

```
virtual CORBA::TypeCode_ptr _it_type() const;
```

Note: For internal use only.

ExceptionHolder::_local_narrow()

```
virtual void* _local_narrow(
    const char* tag
);
```

Note: For internal use only.

ExceptionHolder::marshaled_exception()

Enhancement Orbix enhancement.

ExceptionHolder::marshaled_exception_seq Sequence

```
typedef ITCxxUFixedSeq< CORBA::Octet > _marshaled_exception_seq;
```

Enhancement Orbix enhancement.

Messaging::RebindPolicy Class

The `RebindPolicy` is a client-side QoS policy that specifies whether or not the ORB is allowed to transparently relocate the target corresponding to an object reference. The default `RebindPolicy` supports this transparent rebind.

Rebinding means changing the client-visible QoS as a result of replacing the IOR profile used by a client's object reference with a new IOR profile. *Transparent rebinding* is when this happens without notice to the client application.

If your application has rigorous QoS requirements, transparent rebinding can cause problems. For instance, unexpected errors may occur if your application sets its QoS policies appropriately for an object reference, and then the ORB transparently changes the application's assumptions about that reference by obtaining a new IOR. Your applications can prevent the ORB from silently changing the IOR Profile and therefore the server-side QoS that you have assumed. A more rigorous value of this policy even precludes the ORB from silently closing and opening connections such as when IIOP is being used.

`RebindPolicy` is a local object derived from `CORBA::Policy`.

```
class RebindPolicy;
typedef RebindPolicy* RebindPolicy_ptr;
typedef ITCxxObjRef_var< RebindPolicy_ptr, RebindPolicy,
   ITCxxIntfAlloc< RebindPolicy_ptr, RebindPolicy> >
   RebindPolicy_var;
typedef ITCxxObjRef_out< RebindPolicy_ptr, RebindPolicy,
   ITCxxIntfAlloc< RebindPolicy_ptr, RebindPolicy> >
   RebindPolicy_out;
   ...

IT_ART_API IT_NAMESPACE_STATIC CORBA::TypeCode_ptr
   _tc_RebindPolicy;

class IT_ART_API RebindPolicy : public virtual ::CORBA::Policy {
public:
    typedef Messaging::RebindPolicy_ptr _ptr_type;
    typedef Messaging::RebindPolicy_var _var_type;
```

```
    virtual ~RebindPolicy();

    ...
    static RebindPolicy_ptr _narrow(
        CORBA::Object_ptr obj
    );
    static RebindPolicy_ptr _unchecked_narrow(
        CORBA::Object_ptr obj
    );
    inline static RebindPolicy_ptr _duplicate(
        RebindPolicy_ptr p
    );
    inline static RebindPolicy_ptr _nil();

    virtual ::Messaging::RebindMode rebind_mode() = 0;
    ...
};
```

See page 5 for descriptions of the standard helper methods:

- `_duplicate()`
- `_narrow()`
- `_nil()`
- `_unchecked_narrow()`

RebindPolicy::_local_narrow()

```
virtual void* _local_narrow(
    const char* tag
);
```

Note: For internal use only.

RebindPolicy::rebind_mode()

```
virtual ::Messaging::RebindMode rebind_mode() = 0;
```

Returns the effective rebind policy mode. The effective policies of other types for this object reference may change from invocation to invocation.

For GIOP-based protocols an object reference is considered bound once it is in a state where a locate-request message would result in a locate-reply message with status indicating where the object is. If `rebind_mode()` returns an effective policy value of `TRANSPARENT`, the ORB will silently forward any subsequent messages.

Regardless of the rebind policy in effect, you can always explicitly requested rebind or reconnection by calling `Object::_validate_connection()`. When instances of `RebindPolicy` are created, a value of type `RebindMode` is passed to `ORB::create_policy()`.

Exceptions

`REBIND` Raised if:

- The effective policy value is `NO_REBIND` and if any rebind handling would cause a client-visible change in policies.
- The effective policy value is `NO_RECONNECT` and if any rebind handling would cause a client-visible change in policies, or if a new connection must be opened.

See Also

`Messaging::RebindMode`
`CORBA::ORB::create_policy()`
`CORBA::Object::_validate_connection()`

RebindPolicy::~RebindPolicy() Destructor

```
virtual ~RebindPolicy();
```

The destructor for the object.

Messaging::ReplyHandler Base Class

This is the base class for the messaging callback model. A `ReplyHandler` is a CORBA object, implemented by the client application, which encapsulates the functionality for handling an asynchronous reply. The `ReplyHandler` is used with an asynchronous method invocation (AMI). The `ReplyHandler` is passed to an AMI and it is invoked when the reply to that request is available.

In the callback model, the client passes a reference to a reply handler (a client side CORBA object implementation that handles the reply for a client request), in addition to the normal parameters needed by the request. The reply handler interface defines operations to receive the results of that request (including inout and out values and possible exceptions). The `ReplyHandler` is a normal CORBA object that is implemented by the programmer as with any object implementation.

You must write the implementation for a type-specific `ReplyHandler`. A client obtains an object reference for this `ReplyHandler` and passes it as part of the AMI. When the server completes the request, its reply is delivered as an invocation on the `ReplyHandler` object. This invocation is made on the `ReplyHandler` using the normal POA techniques of servant and object activation. As a result, the callback operation may be handled in a different programming context than that in which the original request was made.

Exceptions can only be raised as part of a reply in the callback model. You use an `ExceptionHolder` to handle these exception replies. You create an `ExceptionHolder` value to encapsulate the identity and contents of an exception that might be raised, and an instance of this `ExceptionHolder` is passed as the argument to the `ReplyHandler` operation to indicate if an exception was raised by the target.

For each operation in an interface, corresponding callback asynchronous method signatures are generated by the IDL compiler. See the *CORBA Programmer's Guide* for generated methods and how to write your asynchronous callback implementations.

```
class ReplyHandler;
class ITGenReplyHandlerStreamable;
typedef ReplyHandler* ReplyHandler_ptr;
typedef ITCxxObjRef_var< ReplyHandler_ptr, ReplyHandler,
   ITCxxIntfAlloc< ReplyHandler_ptr, ReplyHandler> >
   ReplyHandler_var;
typedef ITCxxObjRef_out< ReplyHandler_ptr, ReplyHandler,
   ITCxxIntfAlloc< ReplyHandler_ptr, ReplyHandler> >
   ReplyHandler_out;
...
IT_ART_API IT_NAMESPACE_STATIC CORBA::TypeCode_ptr
   _tc_ReplyHandler;

class IT_ART_API ReplyHandler : public virtual CORBA::Object {
public:
    typedef Messaging::ReplyHandler_ptr _ptr_type;
    typedef Messaging::ReplyHandler_var _var_type;

    virtual ~ReplyHandler();

    ...
    static ReplyHandler_ptr _narrow(
        CORBA::Object_ptr obj
    );

    static ReplyHandler_ptr _unchecked_narrow(
        CORBA::Object_ptr obj
    );

    inline static ReplyHandler_ptr _duplicate(
        ReplyHandler_ptr p
    );

    inline static ReplyHandler_ptr _nil();
    ...
};
```

See page 5 for descriptions of the standard helper methods:

- `_duplicate()`
- `_narrow()`
- `_nil()`

- `_unchecked_narrow()`

ReplyHandler::_local_narrow()

```
virtual void* _local_narrow(
    const char* tag
);
```

Note: For internal use only.

ReplyHandler::~ReplyHandler() Destructor

```
virtual ~ReplyHandler();
```

The destructor for the object.

Messaging::RoutingPolicy Class

The RoutingPolicy is a QoS policy that specifies whether or not the ORB must ensure delivery of a request through the use of queueing. This interface is a local object derived from CORBA::Policy.

When you create instances of RoutingPolicy, you pass a value of type RoutingTypeRange to CORBA::ORB::create_policy(). An instance of RoutingPolicy may be specified when creating a POA and therefore may be represented in object references.

In addition, a POA's RoutingPolicy is visible to clients through the object references it creates, and reconciled with the client's override. If set on both the client and server, reconciliation is performed by intersecting the server-specified RoutingPolicy range with the range of the client's effective override.

```
    class RoutingPolicy;
typedef RoutingPolicy* RoutingPolicy_ptr;
typedef ITCxxObjRef_var< RoutingPolicy_ptr, RoutingPolicy,
   ITCxxIntfAlloc< RoutingPolicy_ptr, RoutingPolicy> >
   RoutingPolicy_var;
typedef ITCxxObjRef_out< RoutingPolicy_ptr, RoutingPolicy,
   ITCxxIntfAlloc< RoutingPolicy_ptr, RoutingPolicy> >
   RoutingPolicy_out;
...
IT_ART_API IT_NAMESPACE_STATIC CORBA::TypeCode_ptr
   _tc_RoutingPolicy;

class IT_ART_API RoutingPolicy : public virtual ::CORBA::Policy {
public:
    typedef Messaging::RoutingPolicy_ptr _ptr_type;
    typedef Messaging::RoutingPolicy_var _var_type;

    virtual ~RoutingPolicy();
    ...
    static RoutingPolicy_ptr _narrow(
        CORBA::Object_ptr obj
    );
```

```
    static RoutingPolicy_ptr _unchecked_narrow(
        CORBA::Object_ptr obj
    );

    inline static RoutingPolicy_ptr _duplicate(
        RoutingPolicy_ptr p
    );

    inline static RoutingPolicy_ptr _nil();

    virtual ::Messaging::RoutingTypeRange routing_range() = 0;
    ...
};
```

See page 5 for descriptions of the standard helper methods:

- `_duplicate()`
- `_narrow()`
- `_nil()`
- `_unchecked_narrow()`

RoutingPolicy::_local_narrow()

```
virtual void* _local_narrow(
    const char* tag
);
```

Note: For internal use only.

RoutingPolicy::~RoutingPolicy() Destructor

```
virtual ~RoutingPolicy();
```

The destructor for the object.

RoutingPolicy::routing_range()

```
virtual ::Messaging::RoutingTypeRange routing_range() = 0;
```

Returns the routing type range.

Messaging::SyncScopePolicy Class

The SyncScopePolicy is an ORB-level QoS policy that modifies the behavior of oneway operations. (Operations are specified in IDL with the oneway keyword.) This policy is only applicable as a client-side override. It is applied to oneway operations to indicate the synchronization scope with respect to the target of that operation request. It is ignored when any non-oneway operation is invoked. This policy is also applied when the DII is used with a flag of INV_NO_RESPONSE because the DII is not required to consult an interface definition to determine if an operation is declared oneway. The default value of this policy is not defined.

SyncScopePolicy is a local object derived from CORBA::Policy. You create instances of SyncScopePolicy by passing a value of type Messaging::SyncScope to CORBA::ORB::create_policy(). The client's SyncScopePolicy is propagated within a request in the request header's response flags. Your applications must explicitly set a SyncScopePolicy to ensure portability across ORB implementations.

```
class SyncScopePolicy;
typedef SyncScopePolicy* SyncScopePolicy_ptr;
typedef ITCxxObjRef_var< SyncScopePolicy_ptr, SyncScopePolicy,
   ITCxxIntfAlloc< SyncScopePolicy_ptr, SyncScopePolicy> >
   SyncScopePolicy_var;
typedef ITCxxObjRef_out< SyncScopePolicy_ptr, SyncScopePolicy,
   ITCxxIntfAlloc< SyncScopePolicy_ptr, SyncScopePolicy> >
   SyncScopePolicy_out;
...
IT_ART_API IT_NAMESPACE_STATIC CORBA::TypeCode_ptr
   _tc_SyncScopePolicy;

class IT_ART_API SyncScopePolicy :
    public virtual ::CORBA::Policy
{
public:
    typedef Messaging::SyncScopePolicy_ptr _ptr_type;
    typedef Messaging::SyncScopePolicy_var _var_type;

    virtual ~SyncScopePolicy();
```

```
    ...
    static SyncScopePolicy_ptr _narrow(
        CORBA::Object_ptr obj
    );

    static SyncScopePolicy_ptr _unchecked_narrow(
        CORBA::Object_ptr obj
    );

    inline static SyncScopePolicy_ptr _duplicate(
        SyncScopePolicy_ptr p
    );

    inline static SyncScopePolicy_ptr _nil();

    virtual ::Messaging::SyncScope synchronization() = 0;
    ...
};
```

See page 5 for descriptions of the standard helper methods:

- `_duplicate()`
- `_narrow()`
- `_nil()`
- `_unchecked_narrow()`

SyncScopePolicy::_local_narrow()

```
virtual void* _local_narrow(
    const char* tag
);
```

Note: For internal use only.

SyncScopePolicy::synchronization()

```
virtual ::Messaging::SyncScope synchronization() = 0;
```

Returns the level of synchronization.

See Also `Messaging::SyncScope`

SyncScopePolicy::~SyncScopePolicy() Destructor

```
virtual ~SyncScopePolicy();
```

The destructor for the object.

IT_PolicyBase Module

IT_PolicyBase Overview

The `IT_PolicyBase` provides an IONA-specific base number for policy IDs so that IONA policies do not conflict with any other policies.

IT_PolicyBase::IONA_POLICY_ID

```
// IDL
module IT_PolicyBase
{
    const unsigned long IONA_POLICY_ID = 0x49545F00;
};

// C++ in namespace IT_PolicyBase
IT_ART_API IT_NAMESPACE_STATIC const
   CORBA::ULong IONA_POLICY_ID;
```

The base policy identifier for all IONA-specific policies. Each IONA policy is identified with a unique policy ID that is some increment of this base number.

Enhancement This is an Orbix enhancement.

See Also IT_CORBA
IT_PortableServer

PortableInterceptor Module

PortableInterceptor Module

The `PortableInterceptor` module consists of these interfaces:

`ClientRequestInfo`
`ClientRequestInterceptor`
`Current`
`Interceptor`
`IORInfo`
`IORInterceptor`
`ORBInitializer`
`ORBInitInfo`
`PolicyFactory`
`RequestInfo`
`ServerRequestInfo`
`ServerRequestInterceptor`

The `PortableInterceptor` module also has the following exceptions and data types:

- `InvalidSlot` exception
- `ForwardRequest` exception
- `ReplyStatus` type
- `SlotId` type

PortableInterceptor::ForwardRequest Exception

```
// IDL
exception ForwardRequest {
    Object forward;
    boolean permanent;
};
```

The `ForwardRequest` exception allows an Interceptor to indicate to the ORB that a retry of the request should occur with the new object given in the exception. The `permanent` flag indicates whether the `forward` object is to become permanent or used only on the forwarded request.

If an Interceptor raises a `ForwardRequest` exception, no other Interceptors are called for that interception point. The remaining Interceptors in the Flow Stack have their appropriate ending interception point called: `receive_other` on the client, or `send_other` on the server. The `reply_status` in the `receive_other` or `send_other` would be `LOCATION_FORWARD` or `LOCATION_FORWARD_PERMANENT`, depending on the value of the `permanent` element of `ForwardRequest`.

PortableInterceptor::InvalidSlot Exception

```
// IDL
exception InvalidSlot {};
```

Raised when a slot ID does not match an allocated slot.

PortableInterceptor::ReplyStatus Type

```
// IDL
typedef short ReplyStatus;
// Valid reply_status values:
const ReplyStatus SUCCESSFUL = 0;
const ReplyStatus SYSTEM_EXCEPTION = 1;
const ReplyStatus USER_EXCEPTION = 2;
const ReplyStatus LOCATION_FORWARD = 3;
const ReplyStatus LOCATION_FORWARD_PERMANENT = 4;
const ReplyStatus TRANSPORT_RETRY = 5;
```

This type is used to define an attribute describing the state of the result of an operation invocation.

See Also `RequestInfo::reply_status`

PortableInterceptor::SlotId Type

```
// IDL
typedef unsigned long SlotId;
```

This type is used to define a slot ID, identifying a slot within its table.

PortableInterceptor:: ClientRequestInfo Interface

This is a locally constrained interface.

```
// IDL
local interface ClientRequestInfo : RequestInfo {
    readonly attribute Object target;
    readonly attribute Object effective_target;
    readonly attribute IOP::TaggedProfile effective_profile;
    readonly attribute any received_exception;
    readonly attribute CORBA::RepositoryId received_exception_id;

    IOP::TaggedComponent get_effective_component(
        in IOP::ComponentId id
    );
    IOP_N::TaggedComponentSeq get_effective_components(
        in IOP::ComponentId id
    );
    CORBA::Policy get_request_policy(
        in CORBA::PolicyType type
    );
    void add_request_service_context(
        in IOP::ServiceContext service_context,
        in boolean replace
    );
};
```

`ClientRequestInfo` is an object through which the client-side Interceptor can access request information. It is passed to the client-side interception points, just as `ServerRequestInfo` is passed to server-side interception points. As there is information that is common to both, they both inherit from a common interface–`RequestInfo`.

Some attributes and operations on `ClientRequestInfo` are not valid at all interception points. Table 14 shows the validity of each attribute or operation. If it is not valid, attempting to access it will result in a `BAD_INV_ORDER` being raised with a standard minor code of 10.

Table 14: *`ClientRequestInfo` Validity*

	send_request	**send_poll**	**receive_reply**	**receive_exception**	**receive_other**
request_id	Yes	Yes	Yes	Yes	Yes
operation	Yes	Yes	Yes	Yes	Yes
arguments	Yes (note 1)	No	Yes	No	No
exceptions	Yes	No	Yes	Yes	Yes
contexts	Yes	No	Yes	Yes	Yes
operation_context	Yes	No	Yes	Yes	Yes
result	No	No	Yes	No	No
response_expected	Yes	Yes	Yes	Yes	Yes
sync_scope	Yes	No	Yes	Yes	Yes
reply_status	No	No	Yes	Yes	Yes
forward_reference	No	No	No	No	Yes (note 2)
get_slot	Yes	Yes	Yes	Yes	Yes
get_request_service_context	Yes	No	Yes	Yes	Yes
get_reply_service_context	No	No	Yes	Yes	Yes

Table 14: *ClientRequestInfo Validity*

	send_request	send_poll	receive_reply	receive_exception	receive_other
target	Yes	Yes	Yes	Yes	Yes
effective_target	Yes	Yes	Yes	Yes	Yes
effective_profile	Yes	Yes	Yes	Yes	Yes
received_exception	No	No	No	Yes	No
received_exception_id	No	No	No	Yes	No
get_effective_component	Yes	No	Yes	Yes	Yes
get_effective_components	Yes	No	Yes	Yes	Yes
get_request_policy	Yes	No	Yes	Yes	Yes
add_request_service_context	Yes	No	No	No	No

Notes

1. When `ClientRequestInfo` is passed to `send_request`, there is an entry in the list for every argument, whether in, inout, or out. But only the in and inout arguments are available.

2. If the `reply_status` attribute is not `LOCATION_FORWARD` or `LOCATION_FORWARD_PERMANENT`, accessing this attribute raises `BAD_INV_ORDER` with a standard minor code of 10.

See Also

`ServerRequestInfo : RequestInfo; RequestInfo`

ClientRequestInfo::add_request_service_context()

```
// IDL
void add_request_service_context(
    in IOP::ServiceContext service_context,
    in boolean replace
);
```

This operation allows Interceptors to add service contexts to a request for information. There is no declaration of the order of the service contexts. They may or may not appear in the order that they are added.

Parameters

`service_context`	The `IOP::ServiceContext` to be added to the request.
`replace`	Indicates the behavior of this operation when a service context already exists with the given ID: • `true`: the existing service context is replaced by the new one. • `false`: `BAD_INV_ORDER` with minor code of `11` is raised.

ClientRequestInfo::effective_profile Attribute

```
// IDL
readonly attribute IOP::TaggedProfile effective_profile;
```

This attribute is the profile that is used to send a request for information. If a location forward has occurred for this operation's object and that object's profile changed accordingly, then this profile is that located profile.

ClientRequestInfo::effective_target Attribute

```
// IDL
readonly attribute Object effective_target;
```

This attribute is the actual object on which a request for information is invoked. If the `reply_status` is `LOCATION_FORWARD`, then on subsequent requests, `effective_target` contains the forwarded IOR while `target`

remains unchanged. If the `reply_status` is `LOCATION_FORWARD_PERMANENT`, then on subsequent requests, both `effective_target` and `target` contains the forwarded IOR.

ClientRequestInfo::get_effective_component()

```
// IDL
IOP::TaggedComponent get_effective_component(
    in IOP::ComponentId id
);
```

This operation returns the `IOP::TaggedComponent` with the given ID from the profile selected for this request. If there is more than one component for a given component ID, it is undefined which component this operation returns.

If there is more than one component for a given component ID, call `get_effective_components` instead.

Parameters

`id`	The `IOP::ComponentId` of the component that is to be returned.

Exceptions

`BAD_PARAM`, minor code `25`	No component exists for the given component ID.

ClientRequestInfo::get_effective_components()

```
// IDL
IOP_N::TaggedComponentSeq get_effective_components(
    in IOP::ComponentId id
);
```

This operation returns all the tagged components with the given ID from the profile elected for this request. This sequence is in the form of an `IOP::TaggedComponentSeq`.

Parameters

`id`	The `IOP::ComponentId` of the components which are to be returned.

Exceptions

`BAD_PARAM`, minor code `25`	No component exists for the given component ID.

ClientRequestInfo::get_request_policy()

```
// IDL
CORBA::Policy get_request_policy(
    in CORBA::PolicyType type
);
```

This operation returns the given policy in effect for the current request for information.

Parameters

`type`	The `CORBA::PolicyType` that specifies the policy to be returned.

Exceptions

`INV_POLICY`, minor code `1`	The policy type is not valid either because the specified type is not supported by this ORB or because a policy object of that type is not associated with this Object.

ClientRequestInfo::received_exception Attribute

```
// IDL
readonly attribute any received_exception;
```

This attribute is an any that contains the exception to be returned to the client.

If the exception is a user exception which cannot be inserted into an any (for example, it is unknown or the bindings do not provide the `TypeCode`, this

attribute will be an any containing the system exception UNKNOWN with a standard minor code of 1.

However, the RepositoryId of the exception is available in the received_exception_id attribute.

ClientRequestInfo::received_exception_id Attribute

```
// IDL
readonly attribute CORBA::RepositoryId received_exception_id;
```

This attribute is the CORBA::RepositoryId of the exception to be returned to the client.

ClientRequestInfo::target Attribute

```
// IDL
readonly attribute Object target;
```

This attribute is the object that the client called to perform the operation.

PortableInterceptor:: ClientRequestInterceptor Interface

This is a locally constrained interface.

```
// IDL
local interface ClientRequestInterceptor : Interceptor {
    void send_request(
        in ClientRequestInfo ri
    ) raises (ForwardRequest);
    void send_poll(
        in ClientRequestInfo ri
    );
    void receive_reply(
        in ClientRequestInfo ri
    );
    void receive_exception(
        in ClientRequestInfo ri
    ) raises (ForwardRequest);
    void receive_other(
        in ClientRequestInfo ri
    ) raises (ForwardRequest);
};
```

A request Interceptor is designed to intercept the flow of a request/reply sequence through the ORB at specific points so that services can query the request information and manipulate the service contexts which are propagated between clients and servers.

The primary use of request Interceptors is to enable ORB services to transfer context information between clients and servers. `ClientRequestInterceptor` provides the client-side request interceptor.

See Also `Interceptor`

ClientRequestInterceptor::receive_exception()

```
// IDL
    void receive_exception(
        in ClientRequestInfo ri
    ) raises (ForwardRequest);
```

This interception point is called when an exception occurs. It allows an Interceptor to query the exception's information before it is raised to the client. This interception point can raise a system exception. This has the effect of changing the exception that successive Interceptors popped from the Flow Stack `receive` on their calls to `receive_exception`. The exception raised to the client is the last exception raised by an Interceptor, or the original exception if no Interceptor changes the exception.

This interception point can also raise a `ForwardRequest` exception (see "PortableInterceptor::ForwardRequest Exception" on page 529 for details on this exception). If an Interceptor raises this exception, no other Interceptors' `receive_exception` operations are called. The remaining Interceptors in the Flow Stack are popped and have their `receive_other` interception point called.

If the `completion_status` of the exception is not `COMPLETED_NO`, then it is inappropriate for this interception point to raise a `ForwardRequest` exception. The request's at-most-once semantics would be lost.

Compliant Interceptors that follow `completion_status` semantics raise a system exception from this interception point. If the original exception is a system exception, the `completion_status` of the new exception is the same as the original. If the original exception is a user exception, then the `completion_status` of the new exception is `COMPLETED_YES`.

Under some conditions, depending on what policies are in effect, an exception (such as `COMM_FAILURE`) can result in a retry of the request. While this retry is a new request with respect to Interceptors, there is one point of correlation between the original request and the retry: because control has not returned to the client, the `PortableInterceptor::Current` for both the original request and the retrying request is the same.

ClientRequestInterceptor::receive_other()

```
// IDL
    void receive_other(
        in ClientRequestInfo ri
    ) raises (ForwardRequest);
```

This interception point allows an Interceptor to query the information available when a request results in something other than a normal reply or an exception.

For example, a request could result in a retry (for example, a GIOP Reply with a `LOCATION_FORWARD` status was received); or on asynchronous calls, the reply does not immediately follow the request, but control returns to the client and an ending interception point is called.

For retries, depending on the policies in effect, a new request may or may not follow when a retry has been indicated. If a new request does follow there is one point of correlation between the original request and the retry, with respect to Interceptors, and for as long as this request is a new request. This is because control has not returned to the client, and so the request scoped `PortableInterceptor::Current` for both the original request and the retrying request is the same.

This interception point can raise a system exception. If it does, no other Interceptors' `receive_other` operations are called. The remaining Interceptors in the Flow Stack are popped and have their `receive_exception` interception point called.

This interception point can also raise a `ForwardRequest` exception (see "PortableInterceptor::ForwardRequest Exception" on page 529 for details on this exception). If an Interceptor raises this exception, successive Interceptors' `receive_other` operations are called with the new information provided by the `ForwardRequest` exception.

Compliant Interceptors properly follow `completion_status` semantics if they raise a system exception from this interception point. The `completion_status` must be `COMPLETED_NO`. If the target invocation had completed, this interception point would not be called.

ClientRequestInterceptor::receive_reply()

```
// IDL
    void receive_reply(
        in ClientRequestInfo ri
    );
```

This interception point allows an Interceptor to query the information on a reply, after it is returned from the server, and before control is returned to the client. This interception point can raise a system exception. If it does, no other Interceptors' `receive_reply` operations are called. The remaining Interceptors in the Flow Stack have their `receive_exception` interception point called.

Compliant Interceptors properly follow `completion_status` semantics if they raise a system exception from this interception point. The `completion_status` is `COMPLETED_YES`.

ClientRequestInterceptor::send_poll()

```
// IDL
    void send_poll(
        in ClientRequestInfo ri
    );
```

This interception point allows an Interceptor to query information during a Time-Independent Invocation (TII) polling get reply sequence. With TII, an application can poll for a response to a request sent previously by the polling client or some other client. This poll is reported to Interceptors through the `send_poll` interception point and the response is returned through the `receive_reply` or `receive_exception` interception points. If the response is not available before the poll time-out expires, the system exception `TIMEOUT` is raised and `receive_exception` is called with this exception.

This interception point can raise a system exception. If it does, no other Interceptors' `send_poll` operations are called. Those Interceptors on the Flow Stack are popped and their `receive_exception` interception points are called. Compliant Interceptors properly follow `completion_status` semantics if they raise a system exception from this interception point. The `completion_status` is `COMPLETED_NO`.

ClientRequestInterceptor::send_request()

```
// IDL
    void send_request(
        in ClientRequestInfo ri
    ) raises (ForwardRequest);
```

This interception point allows an Interceptor to query request information and modify the service context before the request is sent to the server. This interception point can raise a system exception. If it does, no other Interceptors' `send_request` operations are called. Those Interceptors on the Flow Stack are popped and their `receive_exception` interception points are called.

This interception point may also raise a `ForwardRequest` exception (see "PortableInterceptor::ForwardRequest Exception" on page 529 for details of this exception). If an Interceptor raises this exception, no other Interceptors' `send_request` operations are called. Those Interceptors on the Flow Stack are popped and their `receive_other` interception points are called.

Compliant Interceptors follow `completion_status` semantics if they raise a system exception from this interception point. The `completion_status` is `COMPLETED_NO`.

PortableInterceptor::Current Interface

This is a locally constrained interface.

```
// IDL
local interface Current : CORBA::Current {
    any get_slot(
        in SlotId id
    ) raises (InvalidSlot);
    void set_slot(
        in SlotId id,
        in any data
    ) raises (InvalidSlot);
};
```

The `PortableInterceptor::Current` object (referred to as `PICurrent`) is a `Current` object that is used specifically by portable Interceptors to transfer thread context information to a request context. Portable Interceptors are not required to use `PICurrent`. But if information from a client's thread context is required at an Interceptor's interception points, then `PICurrent` can be used to propagate that information. `PICurrent` allows portable service code to be written regardless of an ORB's threading model.

On the client side, this information includes, but is not limited to, thread context information that is propagated to the server through a service context.

On the server side, this information includes, but is not limited to, service context information received from the client which is propagated to the target's thread context.

Current::get_slot()

```
// IDL
any get_slot(
    in SlotId id
) raises (InvalidSlot);
```

A service can get the slot data it set in PICurrent with get_slot(). The return value is the data, in the form of an any, of the given slot identifier.

If the given slot has not been set, an any containing a type code with a TCKind value of tk_null and no value is returned.

Parameters

id	The SlotId of the slot from which the data will be returned.

Exceptions

InvalidSlot	get_slot() is called on a slot that has not been allocated.

Current::set_slot()

```
// IDL
void set_slot(
    in SlotId id,
    in any data
) raises (InvalidSlot);
```

A service sets data in a slot with set_slot(). The data is in the form of an any. If data already exists in that slot, it is overwritten.

Parameters

id	The SlotId of the slot to which the data is set.
data	The data, in the form of an any, which will be set to the identified slot.

Exceptions

InvalidSlot	set_slot() is called on a slot that has not been allocated.

PortableInterceptor::Interceptor Interface

This is a locally constrained interface.

```
// IDL
local interface Interceptor {
    readonly attribute string name;
};
```

Portable Interceptor interfaces and related type definitions reside in the module `PortableInterceptor`. All portable Interceptors inherit from the local interface `Interceptor`.

Interceptor::name Attribute

```
// IDL
readonly attribute string name;
```

Each Interceptor can have a name that is used to order the lists of Interceptors. Only one Interceptor of a given name can be registered with the ORB for each Interceptor type. An Interceptor can be anonymous, that is, have an empty string as the name attribute. Any number of anonymous Interceptors can be registered with the ORB.

PortableInterceptor::IORInfo Interface

This is a locally constrained interface.

```
// IDL
local interface IORInfo {
    CORBA::Policy get_effective_policy(
        in CORBA::PolicyType type
    );
    void add_ior_component(
        in IOP::TaggedComponent a_component
    );
    void add_ior_component_to_profile(
        in IOP::TaggedComponent a_component,
        in IOP::ProfileId profile_id
    );
};
```

In some cases, a portable ORB service implementation has to add information describing the server's or object's ORB service capabilities to object references. This permits the ORB service implementation in the client to function properly.

This is supported through the `IORInterceptor` and `IORInfo` interfaces. The IOR Interceptor is used to establish tagged components in the profiles within an IOR.

IORInfo::add_ior_component()

```
// IDL
void add_ior_component(
    in IOP::TaggedComponent a_component
);
```

A portable ORB service implementation can call `add_ior_component` from its implementation of `establish_components` to add a tagged component to the set that is included when constructing IORs. The components in this set is included in all profiles.

Any number of components can exist with the same component ID.

Parameters

a_component	The IOP::TaggedComponent to add.

IORInfo::add_ior_component_to_profile()

```
// IDL
void add_ior_component_to_profile(
    in IOP::TaggedComponent a_component,
    in IOP::ProfileId profile_id
);
```

A portable ORB service can call add_ior_component_to_profile from its implementation of establish_components to add a tagged component to the set that is included when constructing IORs. The components in this set included in the specified profile.

Any number of components can exist with the same component ID.

Exceptions

BAD_PARAM, minor code 26	The given profile ID does not define a known profile or it is impossible to add components to that profile.

Parameters

a_component	The IOP::TaggedComponent to add.
profile_id	The IOP::ProfileId of the profile to which this component is to be added.

.IORInfo::get_effective_policy()

```
// IDL
CORBA::Policy get_effective_policy(
    in CORBA::PolicyType type
);
```

An ORB service implementation can determine what server side policy of a particular type is in effect for an IOR being constructed by calling `get_effective_policy()`. The returned `CORBA::Policy` object can only be a policy whose type was registered with `ORBInitInfo::register_policy_factory` (see "ORBInitInfo::register_policy_factory()" on page 562).

The return value is the effective `CORBA::Policy` object of the requested type.

Parameters

`type`	The `CORBA::PolicyType` specifying the type of policy to return.

Exceptions

`INV_POLICY`, minor code 2	A policy for the given type was not registered with `register_policy_factory()`.

PortableInterceptor::IORInterceptor Interface

This is a locally constrained interface.

```
// IDL
local interface IORInterceptor : Interceptor {
    void establish_components(
        in IORInfo info
    );
};
```

In some cases, a portable ORB service implementation has to add information describing the server's or object's ORB service capabilities to object references. This permits the ORB service implementation in the client to function properly.

This is supported through the `IORInterceptor` and `IORInfo` interfaces. The IOR Interceptor is used to establish tagged components in the profiles within an IOR.

IORInterceptor::establish_components()

```
// IDL
void establish_components(
    in IORInfo info
);
```

A server side ORB calls `establish_components()` on all registered `IORInterceptor` instances when it is assembling the list of components that to be included in the profile or profiles of an object reference.

This operation is not necessarily called for each individual object reference. For example, the POA specifies policies at POA granularity and therefore, this operation might be called once per POA rather than once per object. In any case, `establish_components` is guaranteed to be called at least once for each distinct set of server policies.

An implementation of `establish_components` must not throw exceptions. If it does, the ORB ignores the exception and proceeds to call the next IOR Interceptor's `establish_components()` operation.

Parameters

`info`	The `IORInfo` instance used by the ORB service to query applicable policies and add components to be included in the generated IORs.

PortableInterceptor::ORBInitializer Interface

This is a locally constrained interface.

```
// IDL
local interface ORBInitializer {
    void pre_init(
        in ORBInitInfo info
    );
    void post_init(
        in ORBInitInfo info
    );
};
```

Interceptors are a means by which ORB services gain access to ORB processing, effectively becoming part of the ORB. Since Interceptors are part of the ORB, when `ORB_init` returns an ORB, the Interceptors have been registered.

Interceptors cannot be registered on an ORB after it has been returned by a call to `ORB_init`.

An Interceptor is registered by registering an associated `ORBInitializer` object that implements the `ORBInitializer` interface. When an ORB initializes, it calls each registered `ORBInitializer`, passing it an `ORBInitInfo` object that is used to register its Interceptor.

ORBInitializer::post_init()

```
// IDL
    void post_init(
        in ORBInitInfo info
    );
```

This operation is called during ORB initialization. If a service must resolve initial references as part of its initialization, it can assume that all initial references are available at this point.

Parameters

`info`	This object provides initialization attributes and operations by which Interceptors can be registered.

ORBInitializer::pre_init()

```
// IDL
    void pre_init(
        in ORBInitInfo info
    );
```

This operation is called during ORB initialization. All calls to `ORBInitInfo::register_initial_reference` must be made at this point so that the list of initial references is complete for the `post_init` point.

Parameters

`info`	This object provides initialization attributes and operations by which Interceptors can be registered.

PortableInterceptor::ORBInitInfo Interface

This is a locally constrained interface.

```
// IDL
local interface ORBInitInfo {
    typedef string ObjectId;
    exception DuplicateName {
        string name;
    };
    exception InvalidName {};
    readonly attribute CORBA::StringSeq arguments;
    readonly attribute string orb_id;
    readonly attribute IOP_N::CodecFactory codec_factory;

    void register_initial_reference(
        in ObjectId id,
        in Object obj
    ) raises (InvalidName);
    void resolve_initial_references(
        in ObjectId id
    ) raises (InvalidName);
    void add_client_request_interceptor(
        in ClientRequestInterceptor interceptor
    ) raises (DuplicateName);
    void add_server_request_interceptor(
        in ServerRequestInterceptor interceptor
    ) raises (DuplicateName);
    void add_ior_interceptor(
        in IORInterceptor interceptor
    ) raises (DuplicateName);
    SlotId allocate_slot_id();
    void register_policy_factory(
        in CORBA::PolicyType type,
        in PolicyFactory policy_factory
```

```
    );
};
```

Interceptors are a means by which ORB services gain access to ORB processing, effectively becoming part of the ORB. Since Interceptors are part of the ORB, when `ORB_init` returns an ORB, the Interceptors have been registered.

Interceptors cannot be registered on an ORB after it has been returned by a call to `ORB_init`.

An Interceptor is registered by registering an associated `ORBInitializer` object that implements the `ORBInitializer` interface. When an ORB initializes, it calls each registered `ORBInitializer`, passing it an `ORBInitInfo` object that is used to register its Interceptor.

ORBInitInfo::add_client_request_interceptor()

```
// IDL
void add_client_request_interceptor(
    in ClientRequestInterceptor interceptor
) raises (DuplicateName);
```

This operation is used to add a client-side request Interceptor to the list of client-side request Interceptors.

Parameters

`interceptor`	The `ClientRequestInterceptor` to be added.

Exceptions

`DuplicateName`	A client-side request Interceptor has already been registered with this Interceptor's name.

ORBInitInfo::add_ior_interceptor()

```
// IDL
void add_ior_interceptor(
    in IORInterceptor interceptor
) raises (DuplicateName);
```

This operation is used to add an IOR Interceptor to the list of IOR Interceptors.

Parameters

`interceptor`	The `IORInterceptor` to be added.

Exceptions

`DuplicateName`	An IOR Interceptor has already been registered with this Interceptor's name.

ORBInitInfo:add_server_request_interceptor()

```
// IDL
void add_server_request_interceptor(
    in ServerRequestInterceptor interceptor
) raises (DuplicateName);
```

This operation is used to add a server-side request Interceptor to the list of server-side request Interceptors.

If a server-side request Interceptor has already been registered with this Interceptor's name, `DuplicateName` is raised.

Parameters

`interceptor`	The `ServerRequestInterceptor` to be added.

ORBInitInfo::allocate_slot_id()

```
// IDL
SlotId allocate_slot_id();
```

A service calls `allocate_slot_id` to allocate a slot on `PortableInterceptor::Current`.

The return value is the allocated slot index.

ORBInitInfo::arguments Attribute

```
// IDL
readonly attribute CORBA::StringSeq arguments;
```

This attribute contains the arguments passed to `ORB_init`. They may or may not contain the ORB's arguments.

ORBInitInfo::codec_factory Attribute

```
// IDL
readonly attribute IOP_N::CodecFactory codec_factory;
```

This attribute is the `IOP::CodecFactory`. The `CodecFactory` is normally obtained with a call to `ORB::resolve_initial_references ("CodecFactory")`, but as the ORB is not yet available and Interceptors, particularly when processing service contexts, require a `Codec`, a means of obtaining a `Codec` is necessary during ORB initialization.

ORBInitInfo::DuplicateName Exception

```
// IDL
exception DuplicateName {
    string name;
};
```

Only one Interceptor of a given name can be registered with the ORB for each Interceptor type. If an attempt is made to register a second Interceptor with the same name, `DuplicateName` is raised.

An Interceptor can be anonymous, that is, have an empty string as the name attribute.

Any number of anonymous Interceptors may be registered with the ORB so, if the Interceptor being registered is anonymous, the registration operation will not raise `DuplicateName`.

ORBInitInfo::InvalidName Exception

```
// IDL
exception InvalidName {};
```

This exception is raised by `register_initial_reference` and `resolve_initial_references`.

`register_initial_reference` raises `InvalidName` if this operation is called with an empty string id; or this operation is called with an id that is already registered, including the default names defined by OMG.

`resolve_initial_references` raises `InvalidName` if the name to be resolved is invalid.

ORBInitInfo::ObjectId Type

```
// IDL
typedef string ObjectId;
```

See Also `ORBInitInfo::register_initial_reference()`

ORBInitInfo::orb_id Attribute

```
// IDL
readonly attribute string orb_id;
```

This attribute is the ID of the ORB being initialized.

ORBInitInfo::register_initial_reference()

```
// IDL
void register_initial_reference(
    in ObjectId id,
    in Object obj
) raises (InvalidName);
```

If this operation is called with an id, “Y”, and an object, YY, then a subsequent call to `ORB::resolve_initial_references ("Y")` will return object YY.

Parameters

`id`	The ID by which the initial reference will be known.
`obj`	The initial reference itself.

Exceptions

`BAD_PARAM`, minor code `24`	The `Object` parameter is null.
`InvalidName`	Raised if this operation is called with: • an empty string id. • an id that is already registered, including the default names defined by OMG.

Notes This method is identical to an operation is available in the ORB interface. This same functionality exists here because the ORB, not yet fully initialized, is not yet available but initial references may need to be registered as part of Interceptor registration. The only difference is that the version of this operation on the ORB uses PIDL (`CORBA::ORB::ObjectId` and `CORBA::ORB::InvalidName`) whereas the version in this interface uses IDL defined in this interface; the semantics are identical.

ORBInitInfo::register_policy_factory()

```
// IDL
void register_policy_factory(
    in CORBA::PolicyType type,
    in PolicyFactory policy_factory
);
```

Register a `PolicyFactory` for the given `PolicyType`.

Parameters

`type`	The `CORBA::PolicyType` that the given `PolicyFactory` serves.
`policy_factory`	The factory for the given `CORBA::PolicyType`.

Exceptions

`BAD_INV_ORDER` with minor code `12`	A `PolicyFactory` already exists for the given `PolicyType`.

ORBInitInfo::resolve_initial_references()

```
// IDL
void resolve_initial_references(
    in ObjectId id
) raises (InvalidName);
```

This operation is only valid during `post_init`. It is identical to `ORB::resolve_initial_references`. This same functionality exists here because the ORB, not yet fully initialized, is not yet available but initial references can be required from the ORB as part of Interceptor registration. The only difference is that the version of this operation on the ORB uses PIDL (`CORBA::ORB::ObjectId` and `CORBA::ORB::InvalidName`) whereas the version in this interface uses IDL defined in this interface; the semantics are identical.

PortableInterceptor::PolicyFactory Interface

This is a locally constrained interface.

```
// IDL
local interface PolicyFactory {
    CORBA::Policy create_policy(
        in CORBA::PolicyType type,
        in any value
    ) raises (CORBA::PolicyError);
};
```

A portable ORB service implementation registers an instance of the `PolicyFactory` interface during ORB initialization in order to enable its policy types to be constructed using `CORBA::ORB::create_policy`. The POA is required to preserve any policy which is registered with `ORBInitInfo` in this manner.

PolicyFactory::create_policy()

```
// IDL
CORBA::Policy create_policy(
    in CORBA::PolicyType type,
    in any value
) raises (CORBA::PolicyError);
```

The ORB calls `create_policy` on a registered `PolicyFactory` instance when `CORBA::ORB::create_policy` is called for the `PolicyType` under which the `PolicyFactory` has been registered.

`create_policy` returns an instance of the appropriate interface derived from `CORBA::Policy` whose value corresponds to the specified any. If it cannot, it raises an exception as described for `CORBA::ORB::create_policy`.

Parameters

type	A CORBA::PolicyType specifying the type of policy being created.
value	An any containing data with which to construct the CORBA::Policy.

PortableInterceptor::RequestInfo Interface

This is a locally constrained interface.

```
// IDL
local interface RequestInfo {
    readonly attribute unsigned long request_id;
    readonly attribute string operation;
    readonly attribute Dynamic::ParameterList arguments;
    readonly attribute Dynamic::ExceptionList exceptions;
    readonly attribute Dynamic::ContextList contexts;
    readonly attribute Dynamic::RequestContext operation_context;
    readonly attribute any result;
    readonly attribute boolean response_expected;
    readonly attribute Messaging::SyncScope sync_scope;
    readonly attribute ReplyStatus reply_status;
    readonly attribute Object forward_reference;
    any get_slot(
        in SlotId id
    ) raises (InvalidSlot);
    IOP::ServiceContext get_request_service_context(
        in IOP::ServiceId id
    );
    IOP::ServiceContext get_reply_service_context(
        in IOP::ServiceId id
    );
};
```

Each interception point is given an object through which the Interceptor can access request information. Client-side and server-side interception points are concerned with different information, so there are two information objects. `ClientRequestInfo` is passed to the client-side interception points and `ServerRequestInfo` is passed to the server-side interception points. But as there is information that is common to both, so they both inherit from a common interface: `RequestInfo`.

See Also ClientRequestInfo; ServerRequestInfo

RequestInfo::arguments Attribute

```
// IDL
readonly attribute Dynamic::ParameterList arguments;
```

This attribute is a `Dynamic::ParameterList` containing the arguments on the operation being invoked. If there are no arguments, this attribute is a zero length sequence.

Exceptions

`NO_RESOURCES`, minor code `1`	The environment does not provide access to the arguments—for example, in the case of Java portable bindings.

RequestInfo::contexts Attribute

```
// IDL
readonly attribute Dynamic::ContextList contexts;
```

This attribute is a `Dynamic::ContextList` describing the contexts that can be passed on this operation invocation. If there are no contexts, this attribute is a zero length sequence.

Exceptions

`NO_RESOURCES`, minor code `1`	The environment does not provide access to the context list—for example, in the case of Java portable bindings.

RequestInfo::exceptions Attribute

```
// IDL
readonly attribute Dynamic::ExceptionList exceptions;
```

This attribute is a `Dynamic::ExceptionList` describing the `TypeCodes` of the user exceptions that this operation invocation can raise. If there are no user exceptions, this attribute is a zero length sequence.

Exceptions

NO_RESOURCES, minor code 1	The environment does not provide access to the exception list—for example, in the case of Java portable bindings.

RequestInfo::forward_reference Attribute

```
// IDL
readonly attribute Object forward_reference;
```

If the reply_status attribute is LOCATION_FORWARD or LOCATION_FORWARD_PERMANENT, then this attribute contains the object to which the request is to be forwarded. It is indeterminate whether a forwarded request actually occurs.

RequestInfo::get_reply_service_context()

```
// IDL
IOP::ServiceContext get_reply_service_context(
    in IOP::ServiceId id
);
```

This operation returns a copy of the service context with the given ID that is associated with the reply.

The return value is the IOP::ServiceContext obtained with the given identifier.

Parameters

id	The IOP::ServiceId of the service context which is to be returned.

Exceptions

BAD_PARAM with minor code 23	The request's service context does not contain an entry for the specified ID.

RequestInfo::get_request_service_context()

```
// IDL
IOP::ServiceContext get_request_service_context(
    in IOP::ServiceId id
);
```

This operation returns a copy of the service context with the given ID that is associated with the request.

The return value is the `IOP::ServiceContext` obtained with the given identifier.

Parameters

`id`	The `IOP::ServiceId` of the service context which is to be returned.

Exceptions

`BAD_PARAM` with minor code `23`	The request's service context does not contain an entry for the specified ID.

RequestInfo::get_slot()

```
// IDL
any get_slot(
    in SlotId id
) raises (InvalidSlot);
```

This operation returns the data from the given slot of the `PortableInterceptor::Current` that is in the scope of the request. If the given slot has not been set, then an `any` containing a type code with a `TCKind` value of `tk_null` is returned.

The return value is the slot data, in the form of an any, obtained with the given identifier.

Parameters

`id`	The `SlotId` of the slot that is to be returned.

Exceptions

`InvalidSlot`	The ID does not define an allocated slot.

RequestInfo::operation Attribute

```
// IDL
readonly attribute string operation;
```

This attribute is the name of the operation being invoked.

RequestInfo::operation_context Attribute

```
// IDL
readonly attribute Dynamic::RequestContext operation_context;
```

This attribute is a `Dynamic::RequestContext` containing the contexts being sent on the request

Exceptions

`NO_RESOURCES`, minor code `1`	The environment does not provide access to the context—for example, in the case of Java portable bindings.

RequestInfo::reply_status Attribute

```
// IDL
readonly attribute ReplyStatus reply_status;
```

This attribute describes the state of the result of the operation invocation. Its value can be one of the following:

```
PortableInterceptor::SUCCESSFUL
PortableInterceptor::SYSTEM_EXCEPTION
PortableInterceptor::USER_EXCEPTION
PortableInterceptor::LOCATION_FORWARD
PortableInterceptor::LOCATION_FORWARD_PERMANENT
PortableInterceptor::TRANSPORT_RETRY
```

On the client:

- Within the `receive_reply` interception point, this attribute is only `SUCCESSFUL`.
- Within the `receive_exception` interception point, this attribute is either `SYSTEM_EXCEPTION` or `USER_EXCEPTION`.

- Within the receive_other interception point, this attribute is any of SUCCESSFUL, LOCATION_FORWARD, LOCATION_FORWARD_PERMANENT, or TRANSPORT_RETRY.

SUCCESSFUL means an asynchronous request returned successfully.

LOCATION_FORWARD and LOCATION_FORWARD_PERMANENT mean that a reply came back with one of these as its status.

TRANSPORT_RETRY means that the transport mechanism indicated a retry: a GIOP reply with a status of NEEDS_ADDRESSING_MODE, for instance.

On the server:

- Within the send_reply interception point, this attribute is only SUCCESSFUL.
- Within the send_exception interception point, this attribute is either SYSTEM_EXCEPTION or USER_EXCEPTION.
- Within the send_other interception point, this attribute is any of: SUCCESSFUL, LOCATION_FORWARD, or LOCATION_FORWARD_PERMANENT. SUCCESSFUL means an asynchronous request returned successfully. LOCATION_FORWARD and LOCATION_FORWARD_PERMANENT mean that a reply came back with one of these as its status.

RequestInfo::request_id Attribute

```
// IDL
readonly attribute unsigned long request_id;
```

This ID uniquely identifies an active request/reply sequence. Once a request/reply sequence is concluded this ID may be reused.

Note that this id is not the same as the GIOP request_id. If GIOP is the transport mechanism used, then these IDs may very well be the same, but this is not guaranteed nor required.

RequestInfo::response_expected Attribute

```
// IDL
readonly attribute boolean response_expected;
```

This boolean attribute indicates whether a response is expected. On the client:

- A reply is not returned when `response_expected` is false, so `receive_reply` cannot be called.
- If an exception occurs, `receive_exception` is called.
- `Otherwise receive_other` is called.

On the client, within send_poll, this attribute is `true`.

RequestInfo::result Attribute

```
// IDL
readonly attribute any result;
```

This attribute is an `any` containing the result of the operation invocation.

If the operation return type is `void`, this attribute is an `any` containing a type code with a `TCKind` value of `tk_void` and no value.

Exceptions

`NO_RESOURCES`, minor code `1`	The environment does not provide access to the result—for example, in the case of Java portable bindings.

RequestInfo::sync_scope Attribute

```
// IDL
readonly attribute Messaging::SyncScope sync_scope;
```

This attribute, defined in the Messaging specification, is pertinent only when `response_expected` is `false`. If `response_expected` is `true`, the value of `sync_scope` is undefined. It defines how far the request progresses before control is returned to the client. This attribute may have one of the following values:

```
Messaging::SYNC_NONE
Messaging::SYNC_WITH_TRANSPORT
Messaging::SYNC_WITH_SERVER
Messaging::SYNC_WITH_TARGET
```

On the server, for all scopes, a reply is created from the return of the target operation call, but the reply does not return to the client. Although it does not return to the client, it does occur, so the normal server-side interception points are followed (that is, `receive_request_service_contexts`, `receive_request`, `send_reply` or `send_exception`). For `SYNC_WITH_SERVER` and `SYNC_WITH_TARGET`, the server does send an empty reply back to the client before the target is invoked. This reply is not intercepted by server-side Interceptors.

PortableInterceptor:: ServerRequestInfo Interface

This is a locally constrained interface.

```
// IDL
local interface ServerRequestInfo : RequestInfo {
    readonly attribute any sending_exception;
    readonly attribute CORBA::OctetSeq object_id;
    readonly attribute CORBA::OctetSeq adapter_id;
    readonly attribute
        CORBA::RepositoryId target_most_derived_interface;
    CORBA::Policy get_server_policy(
        in CORBA::PolicyType type
    );
    void set_slot(
        in SlotId id,
        in any data
    ) raises (InvalidSlot);
    boolean target_is_a(
        in CORBA::RepositoryId id
    );
    void add_reply_service_context(
        in IOP::ServiceContext service_context,
        in boolean replace
    );
};
```

`ServerRequestInfo` is an object through which the server-side Interceptor can access request information. It is passed to the server-side interception points, just as `ClientRequestInfo` is passed to client-side interception points. As there is information that is common to both, they both inherit from a common interface–`RequestInfo`.

ServerRequestInfo::adapter_id Attribute

```
// IDL
readonly attribute CORBA::OctetSeq adapter_id;
```

This attribute is the opaque identifier for the object adapter.

ServerRequestInfo::add_reply_service_context()

```
// IDL
void add_reply_service_context(
    in IOP::ServiceContext service_context,
    in boolean replace
);
```

This operation allows Interceptors to add service contexts to the request. There is no declaration of the order of the service contexts. They may or may not appear in the order that they are added.

Parameters

`service_context`	The `IOP::ServiceContext` to add to the reply.
`replace`	Indicates the behavior of this operation when a service context already exists with the given ID. If `false`, then `BAD_INV_ORDER` with a standard minor code of 11 is raised. If true, then the existing service context is replaced by the new one.

ServerRequestInfo::get_server_policy()

```
// IDL
CORBA::Policy get_server_policy(
    in CORBA::PolicyType type
);
```

This operation returns the policy in effect for this operation for the given policy type. The returned `CORBA::Policy` object is a policy whose type was registered using `register_policy_factory`

Parameters

`type`	The `CORBA::PolicyType` which specifies the policy to be returned.

Exceptions

`INV_POLICY`, minor code 2	A policy for the given type was not registered using `register_policy_factory()`.

ServerRequestInfo::object_id Attribute

```
// IDL
readonly attribute CORBA::OctetSeq object_id;
```

This attribute is the opaque `object_id` describing the target of the operationinvocation.

ServerRequestInfo::sending_exception Attribute

```
// IDL
readonly attribute any sending_exception;
```

This attribute is an `any` that contains the exception to be returned to the client.

If the exception is a user exception which cannot be inserted into an any (that is, it is unknown or the bindings do not provide the `TypeCode`), this attribute is an any containing the system exception `UNKNOWN` with a standard minor code of `1`.

ServerRequestInfo::set_slot()

```
// IDL
void set_slot(
    in SlotId id,
    in any data
) raises (InvalidSlot);
```

This operation allows an Interceptor to set a slot in the `PortableInterceptor::Current` that is in the scope of the request. If data already exists in that slot, it is overwritten.

Parameters

`id`	The `SlotId` of the slot.
`data`	The data, in the form of an any, to store in that slot.

Exceptions

`InvalidSlot`	The ID does not define an allocated slot.

ServerRequestInfo::target_is_a()

```
// IDL
boolean target_is_a(
    in CORBA::RepositoryId id
);
```

This operation returns `true` if the servant is the given `RepositoryId`, and `false` if it is not.

Parameters

`id`	The caller wants to know if the servant is this `CORBA::RepositoryId`.

ServerRequestInfo::target_most_derived_interface Attribute

```
// IDL
readonly attribute
    CORBA::RepositoryId target_most_derived_interface;
```

This attribute is the `RepositoryID` for the most derived interface of the servant.

PortableInterceptor:: ServerRequestInterceptor Interface

This is a locally constrained interface.

```
// IDL
local interface ServerRequestInterceptor : Interceptor {
    void receive_request_service_contexts(
        in ServerRequestInfo ri
    ) raises (ForwardRequest);
    void receive_request(
        in ServerRequestInfo ri
    ) raises (ForwardRequest);
    void send_reply(
        in ServerRequestInfo ri
    );
    void send_exception(
        in ServerRequestInfo ri
    ) raises (ForwardRequest);
    void send_other(
        in ServerRequestInfo ri
    ) raises (ForwardRequest);
};
```

A request Interceptor is designed to intercept the flow of a request/reply sequence through the ORB at specific points so that services can query the request information and manipulate the service contexts which are propagated between clients and servers.

The primary use of request Interceptors is to enable ORB services to transfer context information between clients and servers. `ServerRequestInterceptor` provides the server-side request interceptor.

ServerRequestInterceptor::receive_request()

```
// IDL
void receive_request(
    in ServerRequestInfo ri
) raises (ForwardRequest);
```

This interception point allows an Interceptor to query request information after all the information, including operation parameters, is available. This interception point may or may not execute in the same thread as the target invocation.

In the DSI model, as the parameters are first available when the user code calls `arguments`, `receive_request` is called from within `arguments`. It is possible that `arguments` is not called in the DSI model. The target can call `set_exception` before calling `arguments`.

The ORB guarantees that `receive_request` is called once, either through `arguments` or through `set_exception`. If it is called through `set_exception`, requesting the arguments results in `NO_RESOURCES` being raised with a standard minor code of 1.This interception point can raise a system exception. If it does, no other Interceptors' `receive_request` operations are called. Those Interceptors on the Flow Stack are popped and their `send_exception` interception points are called.

This interception point can also raise a `ForwardRequest` exception If an Interceptor raises this exception, no other Interceptors' `receive_request` operations are called. Those Interceptors on the Flow Stack are popped and their `send_other` interception points are called.

Compliant Interceptors follow `completion_status` semantics if they raise a system exception from this interception point. The `completion_status` shall be `COMPLETED_NO`.

ServerRequestInterceptor::receive_request_service_contexts()

```
// IDL
void receive_request_service_contexts(
    in ServerRequestInfo ri
) raises (ForwardRequest);
```

At this interception point, Interceptors must get their service context information from the incoming request transfer it to `PortableInterceptor::Current` slots This interception point is called before the servant manager is called. Operation parameters are not yet available at this point. This interception point may or may not execute in the same thread as the target invocation.

This interception point can raise a system exception. If it does, no other Interceptors' `receive_request_service_contexts` operations are called. Those Interceptors on the Flow Stack are popped and their `send_exception` interception points are called.

This interception point can also raise a `ForwardRequest` exception (see "PortableInterceptor::ForwardRequest Exception" on page 529). If an Interceptor raises this exception, no other Interceptors' `receive_request_service_contexts` operations are called. Those Interceptors on the Flow Stack are popped and their `send_other` interception points are called.

Compliant Interceptors follow `completion_status` semantics if they raise a system exception from this interception point. The `completion_status` is `COMPLETED_NO`.

ServerRequestInterceptor::send_exception()

```
// IDL
void send_exception(
    in ServerRequestInfo ri
) raises (ForwardRequest);
```

This interception point is called when an exception occurs. It allows an Interceptor to query the exception information and modify the reply service context before the exception is raised to the client.

This interception point can raise a system exception. This has the effect of changing the exception that successive Interceptors popped from the Flow Stack receive on their calls to `send_exception`. The exception raised to the client is the last exception raised by an Interceptor, or the original exception if no Interceptor changes the exception.

This interception point also raises a ForwardRequest exception (see "PortableInterceptor::ForwardRequest Exception" on page 529). If an Interceptor raises this exception, no other Interceptors' send_exception operations are called. The remaining Interceptors in the Flow Stack have their send_other interception points called.

If the completion_status of the exception is not COMPLETED_NO, then it is inappropriate for this interception point to raise a ForwardRequest exception. The request's at-most-once semantics would be lost.

Compliant Interceptors follow completion_status semantics if they raise a system exception from this interception point. If the original exception is a system exception, the completion_status of the new exception is the same as on the original. If the original exception is a user exception, then the completion_status of the new exception shall be COMPLETED_YES.

ServerRequestInterceptor::send_other()

```
// IDL
void send_other(
    in ServerRequestInfo ri
) raises (ForwardRequest);
```

This interception point allows an Interceptor to query the information available when a request results in something other than a normal reply or an exception. For example, a request could result in a retry (for example, a GIOP Reply with a LOCATION_FORWARD status was received).

This interception point can raise a system exception. If it does, no other Interceptors' send_other operations are called. The remaining Interceptors in the Flow Stack have their send_exception interception points called.

This interception point can also raise a ForwardRequest exception (see "PortableInterceptor::ForwardRequest Exception" on page 529). If an Interceptor raises this exception, successive Interceptors' operations are called with the new information provided by the ForwardRequest exception.

Compliant Interceptors follow completion_status semantics if they raise a system exception from this interception point. The completion_status is COMPLETED_NO.

ServerRequestInterceptor::send_reply()

```
// IDL
void send_reply(
    in ServerRequestInfo ri
);
```

This interception point allows an Interceptor to query reply information and modify the reply service context after the target operation has been invoked and before the reply is returned to the client.

This interception point can raise a system exception. If it does, no other Interceptors' `send_reply` operations are called. The remaining Interceptors in the Flow Stack have their `send_exception` interception point called.

Compliant Interceptors follow `completion_status` semantics if they raise a system exception from this interception point. The `completion_status` is `COMPLETED_YES`.

PortableServer Module

PortableServer Overview

The `PortableServer` module includes a number of data structures and classes that are specific to a portable object adapter, or POA. This chapter describes the following:

- "PortableServer Conversion Functions"
- "PortableServer Data Types, Constants, and Exceptions"

The rest of the `PortableServer` classes and interfaces are described in subsequent chapters as follows:

- "PortableServer::AdapterActivator Interface"
- "PortableServer::Current Interface"
- "PortableServer::DynamicImplementation Class"
- "PortableServer::POA Interface"
- "PortableServer::POAManager Interface"
- "PortableServer::ServantActivator Interface"
- "PortableServer::ServantLocator Interface"
- "PortableServer::ServantManager Interface"

The `PortableServer` policy classes are described in subsequent chapters as follows:

- "PortableServer::IdAssignmentPolicy Interface"
- "PortableServer::IdUniquenessPolicy Interface"
- "PortableServer::ImplicitActivationPolicy Interface"
- "PortableServer::LifespanPolicy Interface"
- "PortableServer::RequestProcessingPolicy Interface"
- "PortableServer::ServantRetentionPolicy Interface"
- "PortableServer::ThreadPolicy Interface"

PortableServer Conversion Functions

Objects that are registered with POAs are identified by `ObjectId` types, which are sequences of octets. The PortableServer module contains several conversion functions that let you use strings as object identifiers.

```
// C++
namespace PortableServer {
    char* ObjectId_to_string(const ObjectId&);
    wchar_t* ObjectId_to_wstring(const ObjectId&);
    ObjectId* string_to_ObjectId(const char*);
    ObjectId* wstring_to_ObjectId(const wchar_t*);
}
```

ObjectId_to_string()

```
char* ObjectId_to_string(
    const ObjectId&
);
```

Converts an `ObjectId` to a string.

ObjectId_to_wstring()

```
wchar_t* ObjectId_to_wstring(
    const ObjectId&
);
```

Converts an `ObjectId` to a wide string.

string_to_ObjectId()

```
ObjectId* string_to_ObjectId(
    const char*
);
```

Converts a string to an `ObjectId`.

wstring_to_ObjectId()

```
ObjectId* wstring_to_ObjectId(
    const wchar_t*
);
```

Converts a wide string to an `ObjectId`.

PortableServer Data Types, Constants, and Exceptions

The `PortableServer` module contains the following common exception and data types:

Table 15: *PortableServer Common Types*

Common Types and Exceptions	Policy Value Enumerations	Policy ID Constants
ForwardRequest	IdAssignmentPolicyValue	ID_ASSIGNMENT_POLICY_ID
ObjectId	IdUniquenessPolicyValue	ID_UNIQUENESS_POLICY_ID
POAList	ImplicitActivationPolicyValue	IMPLICIT_ACTIVATION_POLICY_ID
Servant	LifespanPolicyValue	LIFESPAN_POLICY_ID
	RequestProcessingPolicyValue	REQUEST_PROCESSING_POLICY_ID
	ServantRetentionPolicyValue	SERVANT_RETENTION_POLICY_ID
	ThreadPolicyValue	THREAD_POLICY_ID

PortableServer::ForwardRequest Exception

```
//IDL
exception ForwardRequest {
    Object forward_reference;
};
```

In addition to standard CORBA exceptions, a servant manager is capable of raising a `ForwardRequest` exception. This exception includes an object reference.

PortableServer::ID_ASSIGNMENT_POLICY_ID Constant

```
//IDL
const CORBA::PolicyType ID_ASSIGNMENT_POLICY_ID = 19;
```

Defines an ID for the policy `IdAssignmentPolicy`.

PortableServer::IdAssignmentPolicyValue Enumeration

```
// IDL
enum IdAssignmentPolicyValue {
    USER_ID,
    SYSTEM_ID
};
```

One of the following values can be supplied when creating an `IdAssignmentPolicy` policy.

`USER_ID`	Objects created with the POA are assigned an `ObjectId` only by the application.
`SYSTEM_ID`	Objects created with the POA are assigned an `ObjectId` only by the POA. If the POA also has the `PERSISTENT` policy for its objects, the assigned `ObjectId` must be unique across all instantiations of the same POA.

PortableServer::ID_UNIQUENESS_POLICY_ID Constant

```
//IDL
const CORBA::PolicyType ID_UNIQUENESS_POLICY_ID = 18;
```

Defines an ID for the policy `IdUniquenessPolicy`.

PortableServer::IdUniquenessPolicyValue Enumeration

```
// IDL
enum IdUniquenessPolicyValue {
    UNIQUE_ID,
    MULTIPLE_ID
```

```
};
```

One of the following values can be supplied when creating an `IdUniquenessPolicy` policy.

`UNIQUE_ID`	Servants activated with the POA support exactly one `ObjectId`.
`MULTIPLE_ID`	A servant activated with the POA may support one or more `ObjectId` types.

PortableServer::IMPLICIT_ACTIVATION_POLICY_ID Constant

```
//IDL
const CORBA::PolicyType IMPLICIT_ACTIVATION_POLICY_ID = 20;
```

Defines an ID for the policy `ImplicitActivationPolicy`.

PortableServer::ImplicitActivationPolicyValue Enumeration

```
// IDL
enum ImplicitActivationPolicyValue {
    IMPLICIT_ACTIVATION,
    NO_IMPLICIT_ACTIVATION
};
```

One of the following values can be supplied when creating an `ImplicitActivationPolicy` policy.

`IMPLICIT_ACTIVATION`	The POA will support implicit activation of servants. `IMPLICIT_ACTIVATION` also requires the `SYSTEM_ID` and `RETAIN` policy values.
`NO_IMPLICIT_ACTIVATION`	The POA will not support implicit activation of servants.

PortableServer::LIFESPAN_POLICY_ID Constant

```
//IDL
const CORBA::PolicyType LIFESPAN_POLICY_ID = 17;
```

Defines an ID for the policy `LifeSpanPolicy`.

PortableServer::LifespanPolicyValue Enumeration

```
// IDL
enum LifespanPolicyValue {
    TRANSIENT,
    PERSISTENT
};
```

One of the following values can be supplied when creating a `LifespanPolicy` policy:

`TRANSIENT`	The objects implemented in the POA cannot outlive the POA instance in which they are first created. Once the POA is deactivated, use of any object references generated from it will result in an `OBJECT_NOT_EXIST` exception.
`PERSISTENT`	The objects implemented in the POA can outlive the process in which they are first created.

Persistent objects have a POA associated with them which is the POA that created them. When the ORB receives a request on a persistent object, it first searches for the matching POA, based on the names of the POA and all of its ancestors.

See Also `PortableServer::LifespanPolicy`

PortableServer::ObjectId Sequence

```
// IDL
typedef sequence<octet> ObjectId;

//C++
class ObjectId { ...
```

`ObjectId`s are strings that identify a required object reference.

See page 10 for a description of the mapping of IDL sequences.

PortableServer::POAList Sequence

```
// IDL
typedef sequence<POA> POAList;

//C++
class POAList { ...
```

A `POAList` is a sequence of child POAs.

See page 10 for a description of the mapping of IDL sequences.

PortableServer::REQUEST_PROCESSING_POLICY_ID Constant

```
//IDL
const CORBA::PolicyType REQUEST_PROCESSING_POLICY_ID = 22;
```

Defines an ID for the policy `RequestProcessingPolicy`.

PortableServer::RequestProcessingPolicyValue Enumeration

```
// IDL
enum RequestProcessingPolicyValue {
    USE_ACTIVE_OBJECT_MAP_ONLY,
    USE_DEFAULT_SERVANT,
    USE_SERVANT_MANAGER
};
```

One of the following values can be supplied when creating a `RequestProcessingPolicy` policy.

`USE_ACTIVE_OBJECT_MAP_ONLY` If the `ObjectId` is not found in the active object map, an `OBJECT_NOT_EXIST` exception is returned to the client. The `RETAIN` policy value is also required.

USE_DEFAULT_SERVANT	If the ObjectId is not found in the active object map or the NON_RETAIN policy value is present, and a default servant has been registered with the POA using set_servant(), the request is dispatched to the default servant. If no default servant has been registered, an OBJ_ADAPTER exception is returned to the client. The MULTIPLE_ID policy value is also required.
USE_SERVANT_MANAGER	If the ObjectId is not found in the active object map or the NON_RETAIN policy value is present, and a servant manager has been registered with the POA using set_servant_manager(), the servant manager is given the opportunity to locate a servant or raise an exception. If no servant manager has been registered, an OBJECT_ADAPTER exception is returned to the client.

PortableServer::Servant Native Type

```
// IDL
native Servant;

// C++ in namespace PortableServer
typedef ServantBase* Servant;
```

The native Servant type has a language-specific mapping.

PortableServer::SERVANT_RETENTION_POLICY_ID Constant

```
//IDL
const CORBA::PolicyType SERVANT_RETENTION_POLICY_ID = 21;
```

Defines an ID for the policy ServantRetentionPolicy.

PortableServer::ServantRetentionPolicyValue Enumeration

```
// IDL
enum ServantRetentionPolicyValue {
    RETAIN,
    NON_RETAIN
};
```

One of the following values can be supplied when creating a `ServantRetentionPolicy` policy.

`RETAIN`	The POA will retain active servants in its active object map.
`NON_RETAIN`	Servants are not retained by the POA. The `NON_RETAIN` policy requires either the `USE_DEFAULT_SERVANT` or `USE_SERVANT_MANAGER` policy values.

PortableServer::THREAD_POLICY_ID Constant

```
//IDL
const CORBA::PolicyType THREAD_POLICY_ID = 16;
```

Defines an ID for the policy `ThreadPolicy`.

PortableServer::ThreadPolicyValue Enumeration

```
//IDL
enum ThreadPolicyValue {
    ORB_CTRL_MODEL,
    SINGLE_THREAD_MODEL
};
```

One of the following values can be supplied when creating a `ThreadPolicy` policy.

`ORB_CTRL_MODEL`	The ORB is responsible for assigning requests for an ORB-controlled POA to threads. In a multi-threaded environment, concurrent requests may be delivered using multiple threads.

`SINGLE_THREAD_MODEL`	Requests for a single-threaded POA are processed sequentially. In a multi-threaded environment, all up-calls made by this POA to implementation code (servants and servant managers) are made in a manner that is safe for code that is multi-thread-unaware.

In some environments, using a value of `SINGLE_THREAD_MODEL` may mean that the POA will use only the main thread, in which case the application programmer is responsible to ensure that the main thread is given to the ORB, using `ORB::perform_work()` or `ORB::run()`. POAs using a value of `SINGLE_THREAD_MODEL` may need to cooperate to ensure that calls are safe even when a servant manager is shared by multiple single-threaded POAs. These models presume that the ORB and the application are using compatible threading primitives in a multi-threaded environment.

PortableServer::AdapterActivator Interface

Adapter activators are associated with POAs. An adapter activator supplies a POA with the ability to create child POAs on demand, as a side-effect of receiving a request that names the child POA (or one of its children), or when `find_POA()` is called with an activate parameter value of TRUE. An application server that creates all its needed POAs at the beginning of execution does not need to use or provide an adapter activator; it is necessary only for the case in which POAs need to be created during request processing.

While a request from the POA to an adapter activator is in progress, all requests to objects managed by the new POA (or any descendant POAs) will be queued. This serialization allows the adapter activator to complete any initialization of the new POA before requests are delivered to that POA.

```
//IDL
interface AdapterActivator {
    boolean unknown_adapter(
        in POA parent,
        in string name);
};
```

AdapterActivator::unknown_adapter()

```
//IDL
boolean
unknown_adapter(
    in POA parent,
    in string name
);
```

```
//C++
virtual CORBA::Boolean unknown_adapter(
    POA_ptr parent,
    const char* name
```

```
) = 0;
```

Recreates a POA name through the adapter activator of its parent POA. This method returns either true or false:

True	The required POA is created; the ORB continues processing the request.
False	The required POA was not created; the ORB returns an exception of OBJECT_NOT_EXIST to the client.

Parameters

parent	The parent POA.
name	The new name for the POA.

This method is called on the parent POA's adapter activator when the ORB receives a request for an object reference whose POA does not exist. If the POA of the requested object has ancestor POAs that also no longer exist, the ORB calls this method on the adapter activator of each POA that must be recreated. The ORB iterates over the ancestral tree of the missing POA, starting with the most immediate existing ancestor—that is, the parent of the first missing POA. For each missing child POA (specified in parameter name), the ORB invokes this method on its parent's adapter activator until name resolves to the POA that contains the requested object reference.

For example, the ORB might seek an object reference in POA *x*, where *x* is descended from POA *b*, which in turn is a child of the root POA. If *b* and *x* no longer exist, the ORB must restore both POAs in order to recreate the context of the target object reference. By evaluating the object reference, it determines which POAs it needs to restore and calls unknown_adapter() on each one's parent:

1. Calls unknown_adapter() on the adapter activator of the root POA to recreate POA *b*.
2. If the first call to unknown_adapter() returns TRUE, the ORB calls unknown_adapter() on POA *b*'s adapter activator in order to recreate POA *x*.

Until this method returns, all requests to objects managed by the POAs that it creates are queued. If unknown_adapter() returns FALSE, ARTreplies to all queued requests with OBJECT_NOT_EXIST.

Note: `POA::find_POA()` calls this method if the POA to be found does not exist and its `activate_it` parameter is set to TRUE. If `unknown_adapter()` creates the POA and returns TRUE, `find_POA()` returns the required POA.

Exceptions

`OBJECT_NOT_EXIST`	Raised by the ORB to the client if the parent of a POA that needs to be recreated does not have an adapter activator.
`OBJ_ADAPTER`	Raised to the client if the adapter activator raises a system exception.

See Also `PortableServer::POA::find_POA()`

PortableServer::Current Interface

The Current interface, derived from CORBA::Current, provides method implementations with access to the identity of the object on which the method was invoked. The Current interface supports servants that implement multiple objects, but can be used within the context of POA-dispatched method invocations on any servant.

You obtain an instance of Current by calling CORBA::ORB::resolve_initial_references("POACurrent"). Thereafter, it can be used within the context of a method dispatched by the POA to obtain the POA and ObjectId that identify the object on which that operation was invoked.

```
//IDL
interface Current : CORBA::Current {
    exception NoContext {};
    POA get_POA() raises (NoContext);
    ObjectId get_object_id() raises (NoContext);
};
```

Current::get_object_id()

```
//IDL
objectId get_object_id()
    raises(NoContext);

//C++
virtual ObjectId* get_object_id() = 0;
```

When called within the context of a request, this method returns the ObjectId of the target CORBA object.

Use this method to differentiate among different objects that map to the same servant, in a POA that has a MULTIPLE_ID policy value.

Exceptions

NoContext	get_object_id() is called outside the context of a POA-dispatched operation.

Current::get_POA()

```
//IDL
POA get_POA()
    raises(NoContext);

//C++
virtual POA_ptr get_POA() = 0;
```

When called within the context of a request, this method returns a reference to the POA that implements the target CORBA object.

Exceptions

`NoContext`	`get_POA()` is called outside this context.

Current::NoContext Exception

```
// IDL
exception NoContext {};
```

Indicates a `Current` method was called outside the context of POA-dispatched method invocations on a servant.

PortableServer:: DynamicImplementation Class

In C++, DSI servants inherit from the standard `DynamicImplementation` class. This class inherits from the `ServantBase` class and is also defined in the `PortableServer` namespace. The Dynamic Skeleton Interface (DSI) is implemented through servants that are members of classes that inherit from dynamic skeleton classes.

```
// C++
namespace PortableServer {
    class DynamicImplementation : public virtual ServantBase {
    public:
        CORBA::Object_ptr _this();
        virtual void invoke(CORBA::ServerRequest_ptr request) = 0;
        virtual CORBA::RepositoryId _primary_interface(
            const ObjectId& oid,
            POA_ptr poa
        ) = 0;
    };
}
```

DynamicImplementation::invoke()

```
//C++
virtual void invoke(
    CORBA::ServerRequest_ptr request
) = 0;
```

The `invoke()` method receives requests issued to any CORBA object incarnated by the DSI servant and performs the processing necessary to execute the request.

The `invoke()` method should only be invoked by the POA in the context of serving a CORBA request. Invoking this method in other circumstances may lead to unpredictable results.

DynamicImplementation::_primary_interface()

```
virtual CORBA::RepositoryId _primary_interface(
    const ObjectId& oid,
    POA_ptr poa
) = 0;
```

The `_primary_interface()` method returns a valid `RepositoryId` representing the most-derived interface for that `oid`.

Parameters

`oid`	An object identifier.
`poa`	A POA reference.

The `_primary_interface()` method should only be invoked by the POA in the context of serving a `CORBA` request. Invoking this method in other circumstances may lead to unpredictable results.

DynamicImplementation::_this()

```
CORBA::Object_ptr _this();
```

The `_this()` method returns a `CORBA::Object_ptr` for the target object. Unlike `_this()` for static skeletons, its return type is not interface-specific because a DSI servant may very well incarnate multiple CORBA objects of different types.

Exceptions

`PortableServer::WrongPolicy`	`DynamicImplementation::_this()` is invoked outside of the context of a request invocation on a target object being served by the DSI servant .

PortableServer::IdAssignmentPolicy Interface

You obtain an `IdAssignmentPolicy` object by using `POA::create_id_assignment_policy()` and passing the policy to `POA::create_POA()` to specify whether `ObjectId` values in the created POA are generated by the application or by the ORB. This is a policy class derived from `CORBA::Policy`.

If no `IdAssignmentPolicy` value is specified at POA creation, the default value is `SYSTEM_ID`.

```
// IDL
interface IdAssignmentPolicy : CORBA::Policy {
    readonly attribute IdAssignmentPolicyValue value;
};
```

```
// C++ in namespace PortableServer
    class IT_POA_API IdAssignmentPolicy :
        public virtual ::CORBA::Policy
    {
      public:

        typedef PortableServer::IdAssignmentPolicy_ptr _ptr_type;
        typedef PortableServer::IdAssignmentPolicy_var _var_type;
        virtual ~IdAssignmentPolicy();
        static IdAssignmentPolicy_ptr _narrow(
            CORBA::Object_ptr obj
        );
        static IdAssignmentPolicy_ptr _unchecked_narrow(
            CORBA::Object_ptr obj
        );
        inline static IdAssignmentPolicy_ptr _duplicate(
            IdAssignmentPolicy_ptr p
        );
        inline static IdAssignmentPolicy_ptr _nil();

        virtual IdAssignmentPolicyValue value() = 0;
```

```
        static const IT_FWString _it_fw_type_id;
    };
```

See page 5 for descriptions of the standard helper functions:

- `_duplicate()`
- `_narrow()`
- `_nil()`
- `_unchecked_narrow()`

IdAssignmentPolicy::value()

```
// C++
virtual IdAssignmentPolicyValue value() = 0;
```

Returns the value of this POA policy.

PortableServer::IdUniquenessPolicy Interface

You obtain an `IdUniquenessPolicy` object by using `POA::create_id_uniqueness_policy()` and passing the policy to `POA::create_POA()` to specify whether the servants activated in the created POA must have unique object identities. This is a policy class derived from `CORBA::Policy`.

If no `IdUniquenessPolicy` value is specified at POA creation, the default value is `UNIQUE_ID`.

```
// IDL
interface IdUniquenessPolicy : CORBA::Policy {
    readonly attribute IdUniquenessPolicyValue value;
};

// C++ in namespace PortableServer
    class IT_POA_API IdUniquenessPolicy :
        public virtual ::CORBA::Policy
    {
      public:

        typedef PortableServer::IdUniquenessPolicy_ptr _ptr_type;
        typedef PortableServer::IdUniquenessPolicy_var _var_type;
        virtual ~IdUniquenessPolicy();
        static IdUniquenessPolicy_ptr _narrow(
            CORBA::Object_ptr obj
        );
        static IdUniquenessPolicy_ptr _unchecked_narrow(
            CORBA::Object_ptr obj
        );
        inline static IdUniquenessPolicy_ptr _duplicate(
            IdUniquenessPolicy_ptr p
        );
        inline static IdUniquenessPolicy_ptr _nil();
```

```
        virtual IdUniquenessPolicyValue value() = 0;

        static const IT_FWString _it_fw_type_id;
    };
```

See page 5 for descriptions of the standard helper functions:

- `_duplicate()`
- `_narrow()`
- `_nil()`
- `_unchecked_narrow()`

IdUniquenessPolicy::value()

```
// C++
virtual IdUniquenessPolicyValue value() = 0;
```

Returns the value of this POA policy.

PortableServer::
ImplicitActivationPolicy Interface

You obtain an `ImplicitActivationPolicy` object by using `POA::create_implicit_activation_policy()` and passing the policy to `POA::create_POA()` to specify whether implicit activation of servants is supported in the created POA. This is a policy class derived from `CORBA::Policy`.

If no `ImplicitActivationPolicy` value is specified at POA creation, the default value is `NO_IMPLICIT_ACTIVATION`.

```
// IDL
interface ImplicitActivationPolicy : CORBA::Policy {
    readonly attribute ImplicitActivationPolicyValue value;
};

// C++ in namespace PortableServer
    class IT_POA_API ImplicitActivationPolicy :
        public virtual ::CORBA::Policy
    {
      public:

        typedef PortableServer::ImplicitActivationPolicy_ptr
   _ptr_type;
        typedef PortableServer::ImplicitActivationPolicy_var
   _var_type;
        virtual ~ImplicitActivationPolicy();
        static ImplicitActivationPolicy_ptr _narrow(
            CORBA::Object_ptr obj
        );
        static ImplicitActivationPolicy_ptr _unchecked_narrow(
            CORBA::Object_ptr obj
        );
        inline static ImplicitActivationPolicy_ptr _duplicate(
            ImplicitActivationPolicy_ptr p
        );
        inline static ImplicitActivationPolicy_ptr _nil();
```

```
        virtual ImplicitActivationPolicyValue value() = 0;

        static const IT_FWString _it_fw_type_id;
    };
```

See page 4 for descriptions of the standard helper functions:

- `_duplicate()`
- `_narrow()`
- `_nil()`
- `_unchecked_narrow()`

ImplicitActivationPolicy::value()

```
// C++
virtual ImplicitActivationPolicyValue value() = 0;
```

Returns the value of this POA policy.

PortableServer::LifespanPolicy Interface

You obtain a `LifespanPolicy` object by using `POA::create_lifespan_policy()` and passing the policy to `POA::create_POA()` to specify the lifespan of the objects implemented in the created POA. This is a policy class derived from `CORBA::Policy`.

If no `LifespanPolicy` object is passed to `create_POA()`, the lifespan policy value defaults to `TRANSIENT`.

```
// IDL
interface LifespanPolicy : CORBA::Policy {
    readonly attribute LifespanPolicyValue value;
};
// C++ in namespace PortableServer
    class IT_POA_API LifespanPolicy :
        public virtual ::CORBA::Policy
    {
      public:

        typedef PortableServer::LifespanPolicy_ptr _ptr_type;
        typedef PortableServer::LifespanPolicy_var _var_type;
        virtual ~LifespanPolicy();
        static LifespanPolicy_ptr _narrow(
            CORBA::Object_ptr obj
        );
        static LifespanPolicy_ptr _unchecked_narrow(
            CORBA::Object_ptr obj
        );
        inline static LifespanPolicy_ptr _duplicate(
            LifespanPolicy_ptr p
        );
        inline static LifespanPolicy_ptr _nil();

        virtual LifespanPolicyValue value() = 0;

        static const IT_FWString _it_fw_type_id;
```

```
};
```

See page 5 for descriptions of the standard helper functions:

- `_duplicate()`
- `_narrow()`
- `_nil()`
- `_unchecked_narrow()`

LifespanPolicy::value()

```
// C++
virtual LifespanPolicyValue value() = 0;
```

Returns the value of this POA policy.

PortableServer::POA Interface

A POA object manages the implementation of a collection of objects. The POA supports a name space for the objects, which are each identified by an ObjectId. A PPOAOA also provides a name space for POAs. A POA is created as a child of an existing POA, which forms a hierarchy starting with the root POA.

```
//IDL
interface POA {
    exception AdapterAlreadyExists {};
    exception AdapterInactive {};
    exception AdapterNonExistent {};
    exception InvalidPolicy { unsigned short index; };
    exception NoServant {};
    exception ObjectAlreadyActive {};
    exception ObjectNotActive {};
    exception ServantAlreadyActive {};
    exception ServantNotActive {};
    exception WrongAdapter {};
    exception WrongPolicy {};

    //--------------------------------------------------
    // POA creation and destruction
    //--------------------------------------------------
    POA create_POA(
        in string adapter_name,
        in POAManager a_POAManager,
        in CORBA::PolicyList policies
    )
        raises (AdapterAlreadyExists, InvalidPolicy);

    POA find_POA(
        in string adapter_name,
        in boolean activate_it
    )
        raises (AdapterNonExistent);

    void destroy(
```

```
    in boolean etherealize_objects,
    in boolean wait_for_completion
);

// ************************************************
// Factories for Policy objects
// ************************************************
ThreadPolicy create_thread_policy(
    in ThreadPolicyValue value
);

LifespanPolicy create_lifespan_policy(
    in LifespanPolicyValue value
);

IdUniquenessPolicy create_id_uniqueness_policy(
    in IdUniquenessPolicyValue value
);

IdAssignmentPolicy create_id_assignment_policy(
    in IdAssignmentPolicyValue value
);

ImplicitActivationPolicy create_implicit_activation_policy(
    in ImplicitActivationPolicyValue value
);

ServantRetentionPolicy create_servant_retention_policy(
    in ServantRetentionPolicyValue value
);

RequestProcessingPolicy create_request_processing_policy(
    in RequestProcessingPolicyValue value
);

//------------------------------------------------
// POA attributes
//------------------------------------------------
readonly attribute string the_name;
readonly attribute POA the_parent;
readonly attribute POAManager the_POAManager;
attribute AdapterActivator the_activator;
```

```
//-------------------------------------------------
// Servant Manager registration
//-------------------------------------------------
ServantManager get_servant_manager()
    raises (WrongPolicy);

void set_servant_manager(in ServantManager imgr)
    raises (WrongPolicy);

//-------------------------------------------------
// operations for the USE_DEFAULT_SERVANT policy
//-------------------------------------------------
Servant get_servant()
    raises (NoServant, WrongPolicy);

void set_servant(in Servant servant)
    raises (WrongPolicy);

// ************************************************
// object activation and deactivation
// ************************************************
ObjectId activate_object(in Servant servant)
    raises (ServantAlreadyActive, WrongPolicy);

void activate_object_with_id(
    in ObjectId id,
    in Servant servant
)
    raises(
        ServantAlreadyActive,
        ObjectAlreadyActive,
        WrongPolicy
    );

void deactivate_object(in ObjectId oid)
    raises (ObjectNotActive, WrongPolicy);

// ************************************************
// reference creation operations
// ************************************************
Object create_reference(in CORBA::RepositoryId intf)
    raises (WrongPolicy);
```

```
    Object create_reference_with_id(
        in ObjectId oid,
        in CORBA::RepositoryId intf
    )

    //--------------------------------------------------
    // Identity mapping operations
    //--------------------------------------------------
    ObjectId servant_to_id(in Servant servant)
        raises (ServantNotActive, WrongPolicy);

    Object servant_to_reference(in Servant servant)
        raises (ServantNotActive, WrongPolicy);

    Servant reference_to_servant(in Object reference)
        raises (ObjectNotActive, WrongAdapter, WrongPolicy);

    ObjectId reference_to_id(in Object reference)
        raises (WrongAdapter, WrongPolicy);

    Servant id_to_servant(in ObjectId oid)
        raises (ObjectNotActive, WrongPolicy);

    Object id_to_reference(in ObjectId oid)
        raises (ObjectNotActive, WrongPolicy);
};
```

The exceptions defined for the POA class consists of the following:

```
AdapterAlreadyExists
AdapterInactive
AdapterNonExistent
InvalidPolicy
NoServant
ObjectAlreadyActive
ObjectNotActive
ServantAlreadyActive
ServantNotActive
WrongAdapter
WrongPolicy
```

The POA methods are described as follows:

POA::activate_object()

```
//IDL
ObjectId activate_object(in Servant servant)
    raises(ServantAlreadyActive, WrongPolicy);

//C++
virtual ObjectId* activate_object(
    Servant servant
) = 0;
```

Returns a system-generated object ID and associates it with `servant` in the POA's active object map. This method can only be issued in a POA that has policies of `SYSTEM_ID` and `RETAIN`; otherwise, it raises an exception of `WrongPolicy`.

If the specified servant is already in the active object map and the POA has the `UNIQUE_ID` policy, the `ServantAlreadyActive` exception is raised.

Exceptions `ServantAlreadyActive`, `WrongPolicy`

See Also `PortableServer::POA::deactivate_object()`

POA::activate_object_with_id()

```
//IDL
void activate_object_with_id(
    in ObjectId oid,
    in Servant servant
)
  raises (
        ObjectAlreadyActive,
        ServantAlreadyActive,
        WrongPolicy);

//C++
virtual void
activate_object_with_id(
    const ObjectId &  id,
    Servant servant
) = 0;
```

Associates object `oid` with servant `servant` in the POA's active object map. This method can only be issued in a POA that has the `RETAIN` policy.

If you call `activate_object_with_id()` on a POA that has a policy of `SYSTEM_ID` policy, the object ID must be generated by that POA. To get the object ID of a system-generated object reference, call `reference_to_id()`.

Exceptions

`ObjectAlreadyActive`	Object `oid` is already active in this POA—that is, it is associated with a servant in the active object map.
`ServantAlreadyActive`	The POA has the `UNIQUE_ID` policy and the servant is already associated with another object.
`WrongPolicy`	The POA has the `NON_RETAIN` policy.

POA::create_id_assignment_policy()

```
//IDL
IdAssignmentPolicy create_id_assignment_policy(
    in IdAssignmentPolicyValue value
);

//C++
virtual IdAssignmentPolicy_ptr create_id_assignment_policy(
    IdAssignmentPolicyValue value
) = 0;
```

Creates an object of the `IdAssignmentPolicy` interface. This object can be added to the policies list (`CORBA::PolicyList`) of a new POA. The ID assignment policy determines whether object IDs are generated by the POA or the application.

Parameters Specify the POA's ID assignment policy by supplying one of these values for the `value` parameter:

SYSTEM_ID: (default) Only the POA can assign IDs to its objects. If the POA's lifespan policy is set to `PERSISTENT`, object IDs are unique across all instantiations of the same POA.

USER_ID: Only the application can assign object IDs to objects in this POA. The application must ensure that all user-assigned IDs are unique across all instantiations of the same POA.

Typically, a POA with a SYSTEM_ID policy manages objects that are active for only a short period of time, and so do not need to outlive their server process. In this case, the POA also has an object lifespan policy of TRANSIENT.

USER_ID is usually assigned to a POA that has an object lifespan policy of PERSISTENT—that is, it generates object references whose validity can span multiple instantantations of a POA or server process, so the application requires explicit control over object IDs.

See Also PortableServer::POA::create_poa()

POA::create_id_uniqueness_policy()

```
//IDL
IdUniquenessPolicy create_id_uniqueness_policy(
    in IdUniquenessPolicyValue value
);

//C++
virtual IdUniquenessPolicy_ptr
create_id_uniqueness_policy(
    IdUniquenessPolicyValue value
) = 0;
```

Creates an object of the IdUniquenessPolicy interface. This object can be added to the policies list (CORBA::PolicyList) of a new POA. The ID uniqueness policy determines whether a servant can be associated with multiple objects in this POA.

Parameters Specify the POA's ID uniqueness policy by supplying one of these values for the value parameter:

UNIQUE_ID: (default) Each servant in the POA can be associated with only one object ID.

MULTIPLE_ID: Any servant in the POA can be associated with multiple object IDs.

Note: If the same servant is used by different POAs, that servant conforms to the uniqueness policy of each POA. Thus, it is possible for the same servant to be associated with multiple objects in one POA, and be restricted to one object in another.

See Also `PortableServer::POA::create_poa()`

POA::create_implicit_activation_policy()

```
//IDL
ImplicitActivationPolicy create_implicit_activation_policy(
    in ImplicitActivationPolicyValue value );

//C++
virtual ImplicitActivationPolicy_ptr
create_implicit_activation_policy(
    ImplicitActivationPolicyValue value
) = 0;
```

Creates an object of the `ImplicitActivationPolicy` interface. This object can be added to the policies list (`CORBA::PolicyList`) of a new POA. The activation policy determines whether the POA supports implicit activation of servants.

Parameters Specify the POA's activation policy by supplying one of these values for the value parameter:

NO_IMPLICIT_ACTIVATION: (default) The POA only supports explicit activation of servants.

IMPLICIT_ACTIVATION: The POA supports implicit activation of servants. This policy requires that the POA's object ID assignment policy be set to `SYSTEM_ID`, and its servant retention policy be set to `RETAIN`.

See Also
```
PortableServer::POA::create_poa()
PortableServer::POA::create_id_assignment_policy
PortableServer::POA::create_servant_retention_policy
```

POA::create_lifespan_policy()

```
//IDL
LifespanPolicy create_lifespan_policy(
    in LifespanPolicyValue value
);

//C++
virtual LifespanPolicy_ptr create_lifespan_policy(
    LifespanPolicyValue value
) = 0;
```

Creates an object of the `LifespanPolicy` interface. This object can be added to the policies list (`CORBA::PolicyList`) of a new POA. The lifespan policy determines whether object references outlive the process in which they were created.

Parameters Specify a POA's lifespan policy by supplying one of these values for the `value` parameter:

TRANSIENT: (default) Object references do not outlive the POA in which they are created. After a transient object's POA is deactivated, attempts to reference this object yield the exception `CORBA:OBJECT_NOT_EXIST`.

PERSISTENT Object references can outlive the POA in which they are created.

When a POA creates an object reference, it encapsulates it within an IOR. If the POA has a `TRANSIENT` policy, the IOR contains the server process's current location—its host address and port. Consequently, that object reference is valid only as long as the server process remains alive. If the server process dies, the object reference becomes invalid.

If the POA has a `PERSISTENT` policy, the IOR contains the address of the location domain's implementation repository, which maps all servers and their POAs to their current locations. Given a request for a persistent object, the Orbix daemon uses the object's "virtual" address first, and looks up the server process's actual location via the implementation repository.

A POA typically correlates its lifespan and ID assignment policies. `TRANSIENT` and `SYSTEM_ID` are the default settings for a new POA, out of recognition that system-assigned ID's are generally sufficient for transient object references.

PERSISTENT and USER_ID policies are usually set together, inasmuch as an application typically requires explicit control over the object IDs of its persistent object references.

See Also

PortableServer::POA::create_poa()
PortableServer::AdapterActivator::unknown_adapter()

POA::create_POA()

```
//IDL
POA create_POA(
    in string adapter_name,
    in POAManager a_POAManager,
    in CORBA::PolicyList policies
)
    raises(AdapterAlreadyExists, InvalidPolicy);

//C++
virtual POA_ptr create_POA(
    const char* adapter_name,
    POAManager_ptr a_POAManager,
    const CORBA::PolicyList &  policies
) = 0;
```

Creates a portable object adapter (POA). An application calls this method on the parent of the new POA. The name of the new POA adapter_name must be unique among the names of all existing sibling POAs.

You control a POA's behavior through the policy objects that are created and attached to it through the policies parameter. A new POA has the following policy defaults

Table 16: *Policy Defaults for POAs*

Policy	Default Setting
IdAssignmentPolicy	SYSTEM_ID
IdUniquenessPolicy	UNIQUE_ID
ImplicitActivationPolicy	NO_IMPLICIT_ACTIVATION
LifespanPolicy	TRANSIENT

Table 16: *Policy Defaults for POAs*

Policy	Default Setting
RequestProcessingPolicy	USE_ACTIVE_OBJECT_MAP_ONLY
ServantRetentionPolicy	RETAIN
ThreadPolicy	ORB_CTRL_MODEL

Policy objects are copied to the POA before this operation returns, so the application can destroy them while the POA is in use.

You can register either an existing POA manager or a new one with the POA through the a_POAManager parameter. If a_POAManager is null, a new POAManager object is registered with the POA. To obtain the POAManager object of the current POA, call PortableServer::the_POAManager().

When you create a POA, it is in the state of its POA manager—typically, either active or holding. If you create a new POA manager with the POA, it is initially in a holding state. To process requests, it must be put into an active state through PortableServer::POAManager::activate().

If you register an existing manager with the new POA and the manager is in an active state, the new POA might receive requests for objects before it is prepared to process them—that is, before its adapter activator, servant manager, or default servant is initialized. You can avoid this problem in one of these ways:

- Create the POA indirectly through its parent's adapter activator. For example, call find_POA() on the new POA's parent, supplying parameters adapter_name and activate_it arguments of the new (non-existent) POA and TRUE. Orbix queues all incoming requests on the new POA until the adapter activator returns on successful initialization of the POA.
- Before creating the POA, set its manager to a holding state through PortableServer::POAManager::hold_requests().

Exceptions

`AdapterAlreadyExists`	The parent POA already has a child POA with the specified name.
`InvalidPolicy`	Raised for one of these reasons: • A policy object is not valid for the ORB implementation. • Conflicting policy objects are specified—for example, `NON_RETAIN` and `USE_ACTIVE_OBJECT_MAP_ONLY`. The exception index number specifies the first offending policy object through the corresponding index in the `policies` parameter.

POA::create_reference()

```
//IDL
Object create_reference(
    in CORBA::RepositoryId intf
)

//C++
virtual CORBA::Object_ptr create_reference(
    CORBA::RepositoryId const char* intf
) = 0;
```

Creates a CORBA object and returns an object reference. The object reference encapsulates a POA-generated object ID value and the specified interface repository ID. This reference can be passed to clients so it can make requests on the corresponding object.

This operation requires the `SYSTEM_ID` policy. To obtain the generated object ID value call `POA::reference_to_id()` with the created reference. The returned object ID can then be used to associate the servant with an object (and thereby activate the object) by calling `activate_object_with_id()`.

See Also `PortableServer::POA::create_reference_with_id`

POA::create_reference_with_id()

```
//IDL
Object create_reference_with_id(
```

```
    in ObjectId oid,
    in CORBA::RepositoryId intf
)

//C++
virtual CORBA::Object_ptr create_reference_with_id(
    const ObjectId & oid,
    CORBA::RepositoryId const char* intf
) = 0;
```

Returns an object reference that encapsulates the specified object and interface repository identifiers. The resulting reference can be returned to clients to initiate requests on that object.

See Also `PortableServer::POA::create_reference`

POA::create_request_processing_policy()

```
//IDL
RequestProcessingPolicy create_request_processing_policy(
    in RequestProcessingPolicyValue value
);

//C++
virtual RequestProcessingPolicy_ptr
    create_request_processing_policy(
        RequestProcessingPolicyValue value
) = 0;
```

Creates an object of the `RequestProcessingPolicy` interface. This object can be added to the policies list (`CORBA::PolicyList`) of a new POA. This policy determines how the POA finds servants to implement requests.

Specify the POA's request processing policy by supplying one of these values:

- `USE_ACTIVE_OBJECT_MAP_ONLY` (default): Assumes that all object IDs are mapped to a servant in the active object map. The active object map maintains an object-servant mapping until the object is explicitly deactivated through `deactivate_object()`.

  This policy is typically used for a POA that processes requests for a small number of objects. If the object ID is not found in the active object map, an `OBJECT_NOT_EXIST` exception is raised to the client. This policy requires that the POA have a servant retention policy of `RETAIN`.

- USE_DEFAULT_SERVANT: Dispatch requests to the default servant when the POA cannot find a servant for the requested object. This can occur because the object's ID is not in the active object map, or the POA's servant retention policy is set to NON_RETAIN. This policy is typically used for a POA that needs to process many objects that are instantiated from the same class, and thus can be implemented by the same servant.

  This policy requires that the application register a default servant with the POA via set_servant(); otherwise, an OBJ_ADAPTER exception is raised to the client. It also requires the POA's ID uniqueness policy be set to MULTIPLE_ID, so multiple objects can use the default servant.
- USE_SERVANT_MANAGER: The POA's servant manager finds a servant for the requested object when the object's ID is not in the active object map, or when the POA's servant retention policy is set to NON_RETAIN. If the servant manager fails to locate a servant, it raises an exception. This policy requires that the application register a servant manager with the POA via set_servant_manager(); otherwise, an OBJ_ADAPTER exception is returned to the client.

  An application can implement either a servant activator or servant locator as a POA's servant manager, according to the POA's servant retention policy:
- A POA with a policy of RETAIN can register a servant activator. The servant activator incarnates servants for inactive objects on request; these objects remain active until the servant activator etherealizes them.
- A POA with a policy of NON_RETAIN can register a servant locator. The servant locator incarnates a servant for an inactive object each time the object is requested; the servant locator must etherealize the object and delete the servant from memory after the request returns.

  A POA with a of USE_SERVANT_MANAGER policy allows the application to manage object activation directly.

See Also

```
PortableServer::POA::create_poa()
PortableServer::POA::create_servant_retention_policy
PortableServer::POA::create_id_uniqueness_policy
```

POA::create_servant_retention_policy()

```
//IDL
ServantRetentionPolicy create_servant_retention_policy(
```

```
    in ServantRetentionPolicyValue value
);

//C++
virtual ServantRetentionPolicy_ptr
create_servant_retention_policy(
    ServantRetentionPolicyValue value
) = 0;
```

Creates an object of the `ServantRetentionPolicy` interface. This object can be added to the policies list (`CORBA::PolicyList`) of a new POA. This policy determines whether the POA has an active object map to maintain servant-object associations.

Parameters Specify the servant retention policy by supplying one of these arguments for the `value` parameter:

RETAIN: (default) The POA retains active servants in its active object map. If combined with a policy of `USE_SERVANT_MANAGER`, the POA uses a servant activator as its servant manager.

NON_RETAIN: The POA has no active object map. For each request, the POA relies on the servant manager or default servant to map between an object and its servant; all mapping information is destroyed when request processing returns. Therefore, a `NON_RETAIN` policy also requires that the POA have a request processing policy of `USE_DEFAULT_SERVANT` or `USE_SERVANT_MANAGER`.

See Also `PortableServer::POA::create_poa()`
`PortableServer::POA::create_request_processing_policy()`

POA::create_thread_policy()

```
//IDL
ThreadPolicy create_thread_policy(
    in ThreadPolicyValue value
);

//C++
virtual ThreadPolicy_ptr create_thread_policy(
    ThreadPolicyValue value
) = 0;
```

Creates an object of the `ThreadPolicy` interface. This object can be added to the policies list (`CORBA::PolicyList`) of a new POA.

Parameters Specify the POA's thread policy by supplying one of these values for the value parameter:

ORB_CTRL_MODEL: (default) The ORB is responsible for assigning requests for an ORB-controlled POA to threads. In a multi-threaded environment, concurrent requests can be delivered using multiple threads.

SINGLE_THREAD_MODEL: Requests for a single-threaded POA are processed sequentially. In a multi-threaded environment, all calls by a single-threaded POA to implementation code (servants and servant managers) are made in a manner that is safe for code that does not account for multi-threading.

Orbix uses the main thread for a single-threaded POA. In this case, make sure that the main thread is given to the ORB via `ORB::perform_work()` or `ORB::run()`. Multiple single-threaded POAs might need to cooperate to ensure that calls are safe when they share implementation code such as a servant manager.

Both threading policies assume that the ORB and the application are using compatible threading primitives in a multi-threaded environment. All uses of the POA within the server must conform to its threading policy.

See Also `PortableServer::POA::create_poa()`

POA::deactivate_object()

```
//IDL
void deactivate_object(
    in ObjectId oid
)
    raises(ObjectNotActive, WrongPolicy);

//C++
virtual void
deactivate_object(
    const ObjectId & oid
) = 0;
```

Deactivates object `oid` by removing its servant association from the active object map. Call this method only for a POA with a `RETAIN` policy. If the POA has policies of `RETAIN` and `USE_SERVANT_MANAGER`, it calls the servant activator's `etherealize()` method. `deactivate_object()` returns immediately after its call to `etherealize()`.

Exceptions

`ObjectNotActive`	The specified object ID is not associated with a servant.
`WrongPolicy`	The POA has a `NON_RETAIN` policy.

POA::destroy()

```
//IDL
void destroy(
    in boolean etherealize_objects
    in boolean wait_for_completion
);

//C++
virtual void destroy(
    CORBA::Boolean etherealize_objects,
    CORBA::Boolean wait_for_completion
) = 0;
```

Destroys the target POA and all its descendant POAs. A POA thus destroyed can be recreated later on the same server process.

When a POA is destroyed, requests that already began execution on it or its descendents continue to completion. Requests that have not started execution are processed as if they were newly arrived—that is, the ORB tries to recreate the destroyed POA after all of its pending requests have finished processing.

`etherealize_objects` can be set to TRUE for a POA that has policies of `RETAIN` and `USE_SERVANT_MANAGER`. This parameter determines whether to call the servant activator's `etherealize()` method on each active object. Orbix perceives the POA to be destroyed, and therefore unavailable to requests, before any calls to `etherealize()` are made.

If `wait_for_completion` is set to TRUE, `destroy()` returns only after all requests in process and all calls to `etherealize()` return. If set to FALSE, `destroy()` returns after destroying the target POAs.

See Also `PortableServer::POAManager::deactivate()`

POA::find_POA()

```
//IDL
POA find_POA(
    in string adapter_name,
    in boolean activate_it
)
    raises(AdapterNonExistent);

//C++
virtual POA_ptr find_POA(
    const char* adapter_name,
    CORBA::Boolean activate_it
) = 0;
```

Returns a pointer to POA `adapter_name` if it is a child of the target POA. If the target POA has no child of the specified name and `activate_it` is set to TRUE, `find_POA()` invokes the target POA's adapter activator, if one exists. The adapter activator attempts to restore POA `adapter_name`; if successful, `find_POA()` returns the specified POA object.

Exceptions

`AdapterNonExistent`	No POA is returned.

See Also `PortableServer::AdapterActivator::unknown_adapter()`

POA::get_servant()

```
//IDL
Servant get_servant()
    raises(NoServant, WrongPolicy);

//C++
virtual  Servant get_servant() = 0;
```

Returns the POA's default servant. This method can only be called on a POA that has the USE_DEFAULT_SERVANT policy.

Exceptions

NoServant	No default servant is associated with the POA.
WrongPolicy	The POA should have the USE_DEFAULT_SERVANT policy.

POA::get_servant_manager()

```
//IDL
ServantManager get_servant_manager()
    raises(WrongPolicy);

//C++
virtual ServantManager_ptr get_servant_manager() = 0;
```

Returns the POA's servant manager. If no servant manager is associated with the POA, the method returns a null reference.

See Also PortableServer::AdapterActivator::set_servant_manager()

POA::id_to_reference()

```
//IDL
Object id_to_reference(
    in ObjectId oid
)
    raises(ObjectNotActive, WrongPolicy);

//C++
virtual CORBA::Object_ptr id_to_reference(
    const ObjectId & oid
) = 0;
```

Returns an object reference for active object oid. The object reference encapsulates the information used to direct requests to this object.

Exceptions

WrongPolicy	The POA has a policy of NON_RETAIN policy.
ObjectNotActive	The active object map does not contain the specified object ID.

POA::id_to_servant()

```
//IDL
Servant id_to_servant(
    in ObjectId oid
)
    raises(ObjectNotActive, WrongPolicy);

//C++
virtual  Servant id_to_servant(
    const ObjectId &  oid
) = 0;
```

Returns the servant that is associated with object ID `oid` in the active object map. This method call is valid only for a POA with a `RETAIN` policy.

Exceptions

`ObjectNotActive`	The POA's active object map does not contain the specified object ID.
`WrongPolicy`	The POA has a policy of `NON_RETAIN`.

POA::reference_to_id()

```
//IDL
ObjectId reference_to_id(
    in Object reference
)
    raises WrongAdapter, WrongPolicy);

//C++
virtual ObjectId* reference_to_id(
    CORBA::Object_ptr  reference
) = 0;
```

Returns the object ID that is encapsulated by the specified object reference, where `reference` can specify an active or inactive object. Call this method only if the target POA created `reference`.

Exceptions

`WrongAdapter`	The object reference was not created by this POA.
`WrongPolicy`	Reserved for future extensions.

POA::reference_to_servant()

```
//IDL
Servant
reference_to_servant(
    in Object reference
)
    raises(ObjectNotActive, WrongAdapter, WrongPolicy);

//C++
virtual  Servant reference_to_servant(
    CORBA::Object_ptr  reference
) = 0;
```

Returns the servant that incarnates the `reference`-specified object if one of these conditions is true:

- The POA has a policy of `RETAIN` and the specified object is in the active object map.
- The POA has the `USE_DEFAULT_SERVANT` policy and a default servant is registered with the POA.

Exceptions

`ObjectNotActive`	The POA policies are correct but no servant is associated with the specified object.
`WrongAdapter`	The object reference was not created by this POA.
`WrongPolicy`	The POA does not have a policy of either `RETAIN` or `USE_DEFAULT_SERVANT`.
`WrongAdapter`	The object reference was not created by this POA.

POA::servant_to_id()

```
//IDL
ObjectId
servant_to_id(
    in Servant servant
)
    raises(ServantNotActive, WrongPolicy);

//C++
virtual ObjectId* servant_to_id(
```

```
    Servant servant
) = 0;
```

Returns an object ID for an object that is incarnated by `servant`.

Depending on the POA's policies, the method can take one of the following actions:

- Returns the ID of an already active object if the POA has the `UNIQUE_ID` policy and `servant` already incarnates an object.
- Associates `servant` with a POA-generated object ID and returns that ID if the POA has the `IMPLICIT_ACTIVATION` policy *and* one of these conditions is true:
  - The POA has the `MULTIPLE_ID` policy.
  - `servant` is not associated with any object.

Exceptions

`WrongPolicy`	The POA must have policies of `RETAIN` and either `UNIQUE_ID` or `IMPLICIT_ACTIVATION`; otherwise, it raises this exception. For example, if a POA has a policy of `RETAIN`, `NO_IMPLICIT_ACTIVATION`, and `MULTIPLE_ID`, `servant_to_id()` cannot tell which of the objects that this servant potentially incarnates it should return.
`ServantNotActive`	`servant` is not associated with any object and the POA has a `NO_IMPLICIT_ACTIVATION` policy; therefore, no activation occurs.

POA::servant_to_reference()

```
//IDL
Object servant_to_reference(
    in Servant servant
)
    raises (ServantNotActive, WrongPolicy);

//C++
virtual CORBA::Object_ptr servant_to_reference(
    Servant servant
) = 0;
```

Returns an object reference for an object that is incarnated by `servant`.

Depending on the POA's policies, the method can take one of the following actions:

- If the POA has the UNIQUE_ID policy and servant already incarnates an active object, the method returns an object reference that encapsulates the information used to activate that object.
- If the POA has the IMPLICIT_ACTIVATION policy, and the POA has the MULTIPLE_ID policy or servant is not associated with any object, the servant is associated with a POA-generated object ID—thereby activating the object—and a corresponding object reference is returned.

Exceptions

WrongPolicy	The POA policy must have the RETAIN and either the UNIQUE_ID or IMPLICIT_ACTIVATION policies; otherwise, the exception is raised. For example, if a POA has a policy of RETAIN, NO_IMPLICIT_ACTIVATION, and MULTIPLE_ID, servant_to_reference() cannot ascertain which of the many objects potentially incarnated by the specified servant it should specify in its return.
ServantNotActive	servant is not associated with any object and the POA has a NO_IMPLICIT_ACTIVATION policy; therefore, no activation occurs.

POA::set_servant()

```
//IDL
void
set_servant(
    in Servant servant
)
    raises(WrongPolicy);

//C++
virtual void set_servant(
    Servant servant
) = 0;
```

Registers servant with the POA as the default servant. This servant is used in a POA that has the USE_DEFAULT_SERVANT policy; it services any requests for objects that are not registered in the active object map

Exceptions

`WrongPolicy`	The POA does not have the `USE_DEFAULT_SERVANT` policy.

POA::set_servant_manager()

```
//IDL
void set_servant_manager(
    in ServantManager imgr
)
    raises(WrongPolicy);

//C++
virtual void set_servant_manager(
    ServantManager_ptr imgr
) = 0;
```

Sets the default servant manager for the target POA.

Exceptions

`WrongPolicy`	Raised if the POA does not have a policy of `USE_SERVANT_MANAGER`.

POA::the_name()

```
//C++
virtual char* the_name() = 0;
```

Returns the name of the target POA.

POA::the_parent()

```
//C++
virtual POA_ptr the_parent() = 0;
```

Returns a pointer to the target POA's parent.

POA::the_POAManager()

```
//C++
virtual POAManager_ptr the_POAManager() = 0;
```

Returns a pointer to the target POA's manager.

POA::the_activator()

```
//C++
virtual AdapterActivator_ptr the_activator() = 0;


virtual void the_activator(
    AdapterActivator_ptr  _the_activator
) = 0;
```

Returns or sets a pointer to the target POA's adapter activator.

PortableServer::POAManager Interface

A POAManager is associated with one or more POA objects. (Each POA object has an associated POAManager object.) A POA manager encapsulates the processing state of its POAs. Using operations on the POA manager, an application can cause requests for those POAs to be queued or discarded, and can cause the POAs to be deactivated.

POA managers are created and destroyed implicitly. Unless an explicit POAManager object is provided at POA creation time, a POA manager is created when a POA is created and is automatically associated with that POA. A POAManager object is implicitly destroyed when all of its associated POAs have been destroyed.

```
//IDL
interface POAManager {
    exception AdapterInactive{};
    enum State { HOLDING, ACTIVE, DISCARDING, INACTIVE };
    void activate()
        raises(AdapterInactive);
    void hold_requests(
        in boolean wait_for_completion)
        raises(AdapterInactive);
    void discard_requests(
        in boolean wait_for_completion)
        raises(AdapterInactive);
    void deactivate(
        in boolean etherealize_objects,
        in boolean wait_for_completion)
        raises(AdapterInactive);
    State get_state();
};
```

POAManager::activate()

```
//IDL
void
activate()
    raises (AdapterInactive);

//C++
virtual void activate() = 0;
```

Changes the state of the POA manager to active so it can process requests. When a POA manager is active, all associated POAs can receive requests.

Note: A POA manager's ability to process requests is dependent on resource limits. Orbix provides queues whose lengths are configurable, and raises a system exception of `TRANSIENT` when the queues are full.

When a POA manager is created, it is initially in a holding state. All requests sent to that POA are queued until you call `activate()` on a POA's manager. `activate()` can also reactivate a POA manager that has reverted to a holding state (due to a `hold_requests()` call) or is in a discarding state (due to a `discard_requests()` call).

If a new POA is associated with an existing active POA manager, it is unnecessary to call this method. However, it is generally, a good idea to put a POA manager in a holding state before creating a new POA with it.

Exceptions

`AdapterInactive`	This method is issued on an inactive POA manager.

See Also

`PortableServer::POAManager::activate()`
`PortableServer::POAManager::deactivate()`
`PortableServer::POAManager::discard_requests()`
`PortableServer::POAManager::hold_requests()`

POAManager::AdapterInactive Exception

```
// IDL
exception AdapterInactive{};
```

Indicates that the `POAManager` is inactive and unable to process requests.

POAManager::deactivate()

```
//IDL
void deactivate(
    in boolean etherealize_objects,
    in boolean wait_for_completion
);
    raises(AdapterInactive);

//C++
virtual void deactivate(
    CORBA::Boolean etherealize_objects,
    CORBA::Boolean wait_for_completion
) = 0;
```

Causes the POA manager to shut down. A POA manager deactivates before its associated POAs are destroyed. When it is in an inactive state, the POA manager allows all outstanding requests to complete processing, but refuses all incoming requests.

Parameters The method takes two boolean parameters:

`etherealize_objects`	Determines whether the target POAs etherealize their servants after all request processing is complete. This argument applies only to POAs that have a servant retention policy of `RETAIN` and request processing policy of `USE_SERVANT_MANAGER`.
`wait_for_completion`	Determines whether the method returns immediately or waits until the completion of all requests whose processing began before the call to `deactivate()`.

Exceptions

`AdapterInactive`	The method is issued on a POA manager that is already inactive.

See Also `PortableServer::POA::destroy()`

POAManager::discard_requests()

```
//IDL
void discard_requests(
    in boolean wait_for_completion
```

```
)
    raises(AdapterInactive);

//C++
virtual void discard_requests(
    CORBA::Boolean wait_for_completion
) = 0;
```

Causes the POA manager to discard all incoming requests. When a request is discarded, the server raises a `TRANSIENT` system exception to the client so it can reissue the request. This method can return immediately or wait until the return of all requests whose processing had already begun, according to the argument supplied for `wait_for_completion`.

Parameters

`wait_for_completion` determines whether the method returns immediately or waits until the completion of all requests whose processing began before the call.

This method is typically called when an application detects that an object or the POA in general is overwhelmed by incoming requests. A POA manager should be in a discarding state only temporarily. On resolution of the problem that required this call, the application should restore the POA manager to its active state with `activate()`.

Exceptions

`AdapterInactive`	The method is issued on an inactive POA manager.

See Also

```
PortableServer::POAManager::activate()
PortableServer::POAManager::discard_requests()
PortableServer::POAManager::hold_requests()
```

POAManager::get_state()

```
//IDL
State get_state();

//C++
PortableServer::POAManager::State get_state();
```

Returns the current state of the `POAManager`.

POAManager::hold_requests()

```
//IDL
void hold_requests(
    in boolean wait_for_completion
);
    raises(AdapterInactive);

//C++
virtual void hold_requests(
    CORBA::Boolean wait_for_completion
) = 0;
```

Causes all POAs associated with this manager to queue incoming requests.

The number of requests that can be queued is implementation-dependent. . Set `wait_for_completion` to determine whether this method returns immediately or waits until the return of all requests whose processing began before this call.

A POA manager is always created in a holding state.

Exceptions

`AdapterInactive` The method is issued on an inactive POA manager.

See Also

`PortableServer::POAManager::activate()`
`PortableServer::POAManager::deactivate()`
`PortableServer::POAManager::discard_requests()`

POAManager::State Enumeration

```
//IDL
enum State { HOLDING, ACTIVE, DISCARDING, INACTIVE };
```

The possible state values consist of the following:

`HOLDING`
`ACTIVE`
`DISCARDING`
`INACTIVE`

PortableServer::RequestProcessingPolicy Interface

You obtain a RequestProcessingPolicy object by using POA::create_request_processing_policy() and passing the policy to POA::create_POA() to specify how requests are processed by the created POA. This is a policy class derived from CORBA::Policy.

If no RequestProcessingPolicy value is specified at POA creation, the default value is USE_ACTIVE_OBJECT_MAP_ONLY.

You can define many possible combinations of behaviors with the policies RequestProcessingPolicy and ServantRetentionPolicy.

- RETAIN and USE_ACTIVE_OBJECT_MAP_ONLY

  This combination represents the situation where the POA does no automatic object activation (that is, the POA searches only the active object map). The server must activate all objects served by the POA explicitly, using either activate_object() or activate_object_with_id().

- RETAIN and USE_SERVANT_MANAGER

  This combination represents a very common situation, where there is an active object map and a ServantManager. Because RETAIN is in effect, the application can call activate_object() or activate_object_with_id() to establish known servants in the active object map for use in later requests. If the POA doesn't find a servant in the active object map for a given object, it tries to determine the servant by means of invoking incarnate() in the ServantManager (specifically a ServantActivator) registered with the POA. If no ServantManager is available, the POA raises the OBJECT_ADAPTER system exception.

- RETAIN and USE_DEFAULT_SERVANT

  This combination represents the situation where there is a default servant defined for all requests involving unknown objects. Because RETAIN is in effect, the application can call activate_object() or activate_object_with_id() to establish known servants in the active

object map for use in later requests. The POA first tries to find a servant in the active object map for a given object. If it does not find such a servant, it uses the default servant. If no default servant is available, the POA raises the OBJECT_ADAPTER system exception.

- NON-RETAIN and USE_SERVANT_MANAGER

  This combination represents the situation where one servant is used per method call. The POA doesn't try to find a servant in the active object map because the active object map does not exist. In every request, it will call preinvoke() on the ServantManager (specifically a ServantLocator) registered with the POA. If no ServantManager is available, the POA will raise the OBJECT_ADAPTER system exception.

- NON-RETAIN and USE_DEFAULT_SERVANT

  This combination represents the situation where there is one single servant defined for all CORBA objects. The POA does not try to find a servant in the active object map because the active object map doesn't exist. In every request, the POA will invoke the appropriate operation on the default servant registered with the POA. If no default servant is available, the POA will raise the OBJECT_ADAPTER system exception.

```
// IDL
interface RequestProcessingPolicy : CORBA::Policy {
    readonly attribute RequestProcessingPolicyValue value;
};

// C++ in namespace PortableServer
    class IT_POA_API RequestProcessingPolicy :
        public virtual ::CORBA::Policy
    {
      public:

        typedef PortableServer::RequestProcessingPolicy_ptr
   _ptr_type;
        typedef PortableServer::RequestProcessingPolicy_var
   _var_type;
        virtual ~RequestProcessingPolicy();
        static RequestProcessingPolicy_ptr _narrow(
            CORBA::Object_ptr obj
        );
        static RequestProcessingPolicy_ptr _unchecked_narrow(
```

```
            CORBA::Object_ptr obj
        );
        inline static RequestProcessingPolicy_ptr _duplicate(
            RequestProcessingPolicy_ptr p
        );
        inline static RequestProcessingPolicy_ptr _nil();

        virtual RequestProcessingPolicyValue value() = 0;

        static const IT_FWString _it_fw_type_id;
    };
```

See page 4 for descriptions of the standard helper functions:

- `_duplicate()`
- `_narrow()`
- `_nil()`
- `_unchecked_narrow()`

RequestProcessingPolicy::value()

```
// C++
virtual RequestProcessingPolicyValue value() = 0;
```

Returns the value of this POA policy.

PortableServer::ServantActivator Interface

When a POA has the RETAIN policy value, it uses the ServantActivator type of servant manager.

```
//IDL
interface ServantActivator : ServantManager {
    Servant incarnate(
        in ObjectId oid,
        in POA adapter
    )
        raises (ForwardRequest);

    void etherealize(
        in ObjectId oid,
        in POA adapter,
        in Servant serv,
        in boolean cleanup_in_progress,
        in boolean remaining_activations
    );
};
```

ServantActivator::etherealize()

```
//IDL
void etherealize(
    in ObjectId oid,
    in POA adapter,
    in Servant serv
    in boolean cleanup_in_progress
    in boolean remaining_activations
);
```

```
//C++
virtual void etherealize(
    const ObjectId &  oid,
```

```
    POA_ptr  adapter,
    Servant serv,
    CORBA::Boolean cleanup_in_progress,
    CORBA::Boolean remaining_activations
) = 0;
```

Destroys a servant in a POA that has the `RETAIN` and `USE_SERVANT_MANAGER` policy values.

The `cleanup_in_progress` parameter indicates the context in which this method was called. If is set to true, `etherealize()` was called because of calls to either `PortableServer::POAManager::deactivate()` or `PortableServer::POA::destroy()` with its `etherealize_objects` parameter set to true. If `cleanup_in_progress` is false, this method was called for other reasons.

Because a servant can incarnate multiple objects, `etherealize()` checks the `remaining_activations` parameter to determine whether this servant incarnates any other objects within this POA; if `remaining_activations` is set to FALSE and the servant is not used by other POAs, the method can safely delete the servant from memory.

Before the POA calls on a servant manager's `etherealize()` method, it takes steps to ensure the safe destruction of servants in a multi-threaded environment:

- Removes the target object and its servant from the active object map. Thus, new requests for the target object cannot be invoked on the servant while it undergoes etherealization.
- Calls `etherealize()` on the servant only after all outstanding requests finish processing.

A servant can be etherealized by a servant activator other than the one that originally incarnated it.

ServantActivator::incarnate()

```
//IDL
Servant incarnate(
    in ObjectId oid,
    in POA adapter
)
```

```
    raises (ForwardRequest);

//C++
virtual  Servant incarnate(
    const ObjectId &  oid,
    POA_ptr  adapter
) = 0;
```

Called by the POA when it receives a request for object `oid`, where `oid` contains the ID of an inactive object. `incarnate()` returns an appropriate servant for the requested object; this servant is associated with `oid` in the POA's active object map, thereby activating the object. Subsequent requests for this object are mapped directly to the servant.

This method is only called by the POA on a servant activator, which the POA uses as its servant manager when it has policies of `USE_SERVANT_MANAGER` and `RETAIN`. When using a servant activator, the active object map retains a servant-object association until the servant is etherealized. A servant can only incarnate a given object once. If the POA has a policy of `UNIQUE_ID`, `incarnate()` can only return a servant that does not incarnate any object other than `oid` in that POA.

Note: If the same servant is used by different POAs, that servant conforms to the uniqueness policy of each POA. Thus, it is possible for the same servant to be associated with multiple objects in one POA, and be restricted to one object in another.

Exceptions

`ForwardRequest` The client is instructed to send this request and subsequent requests for `oid` to the object specified in the exception's `forward_reference` member—in IIOP, through a `LOCATION_FORWARD` reply.

See Also

`PortableServer::ServantActivator::etherealize()`
`PortableServer::ServantLocator::preinvoke()`

PortableServer::ServantLocator Interface

When the POA has the NON_RETAIN policy value it uses servant locators as its servant managers. Because the POA knows that the servant returned by a ServantLocator will be used only for a single request, it can supply extra information to the servant manager's operations. Also, the servant manager's pair of operations may be able to cooperate to do something different than a ServantActivator.

```
//IDL
interface ServantLocator : ServantManager {
    native Cookie;
    Servant preinvoke(
            in ObjectId oid,
            in POA adapter,
            in CORBA::Identifier operation,
            out Cookie the_cookie
    )    raises (ForwardRequest);

    void postinvoke(
            in ObjectId oid,
            in POA adapter,
            in CORBA::Identifier operation,
            in Cookie the_cookie,
            in Servant the_servant
    );
};
```

ServantLocator::Cookie Native Type

```
// IDL
native Cookie;
```

```
// C++
typedef void* Cookie;
```

The Cookie native type is opaque to the POA. It can be set by the servant manager for use later by postinvoke().

ServantLocator::postinvoke()

```
//IDL
void postinvoke(
    in ObjectId oid,
    in POA adapter
    in CORBA::Identifier operation,
    in Cookie the_cookie,
    in Servant the_servant
);
```

```
//C++
virtual void postinvoke(
    const ObjectId & oid,
    POA_ptr adapter,
    CORBA::Identifier const char* operation,
    Cookie the_cookie,
    Servant the_servant
) = 0;
```

Called on a POA's servant locator to delete a servant when processing of a request for object oid is complete.

Each postinvoke() call is paired to an earlier preinvoke() call. In order to explicitly map data between the two calls, you set the preinvoke() method's the_cookie parameter. This can be especially useful in a multi-threaded environment where it is important to ensure that a pair of preinvoke() and postinvoke() calls operate on the same servant. For example, each preinvoke() call can set its the_cookie parameter to data that identifies its servant; the postinvoke() code can then compare that data to its the_servant parameter.

The POA calls this method only on a servant locator, which the POA uses as its servant manager when it has policies of USE_SERVANT_MANAGER and NON_RETAIN.

See Also

PortableServer::ServantLocator::preinvoke()
PortableServer::POA::create_reference_with_id()

ServantLocator::preinvoke()

```
//IDL
Servant preinvoke(
    in ObjectId oid,
    in POA adapter,
    in CORBA::Identifier operation,
    out Cookie the_cookie
)
    raises (ForwardRequest);

//C++
virtual Servant preinvoke(
    const ObjectId & oid,
    POA_ptr adapter,
    CORBA::Identifier const char* operation,
    Cookie& the_cookie
) = 0;
```

Returns an appropriate servant for the requested object. This method is called on a POA's servant locator when the POA receives a request for object `oid`, where `oid` contains the ID of an inactive object.

This method is only called by the POA on a servant locator, which the POA uses as its servant manager when it has policies of `USE_SERVANT_MANAGER` and `NON_RETAIN`.

The lack of an active object map can require the following behavior:

- After processing on the requested object is complete, the POA calls `postinvoke()` on the object and etherealizes its servant.
- Each request for an object is treated independently, irrespective of the status of earlier requests for that object. So, it is possible for a POA to accept multiple requests for the same object concurrently and for its servant locator to incarnate several servants for that object simultaneously.

Alternatively, the application can maintain its own object-servant map in order to allow a servant to process multiple requests for the same object, or to process requests for multiple objects. For example, a database server can use a servant locator to direct concurrent operations to the same servant; database transactions are opened and closed within the `preinvoke()` and `postinvoke()` operations.

Each `preinvoke()` call is paired to an later `postinvoke()` call. In order to explicitly map data between the two calls, set `preinvoke()`'s `the_cookie` parameter. This can be especially useful in a multi-threaded environment where it is important to ensure that a pair of `preinvoke()` and `postinvoke()` calls operate on the same servant. For example, each `preinvoke()` call can set its cookie parameter to data that identifies its servant; the `postinvoke()` code can then compare that data to its `the_servant` parameter.

Exceptions

`ForwardRequest` The client is instructed to send this request and subsequent requests for `oid` to the object specified in the exception's `forward_reference` member—in IIOP, through a `LOCATION_FORWARD` reply.

See Also `PortableServer::ServantLocator::postinvoke()`

PortableServer::ServantManager Interface

A servant manager supplies a POA with the ability to activate objects on demand when the POA receives a request targeted at an inactive object. A servant manager is registered with a POA as a callback object, to be invoked by the POA when necessary.

A servant manager is used in servers only for the case in which an object must be activated during request processing. An application server that activates all its needed objects at the beginning of execution does not need to use a servant manager.

The ServantManager interface is an empty base interface that is inherited by the interfaces ServantActivator and ServantLocator. These two types of servant managers have the following corresponding policy values:

Table 17: *Corresponding Policies for Servant Managers*

Servant Manager	POA Policy Value
ServantActivator	RETAIN
ServantLocator	NON_RETAIN

```
//IDL
interface ServantManager
{ };
```

See Also

PortableServer::ServantActivator
PortableServer::ServantLocator

PortableServer::
ServantRetentionPolicy Interface

You obtain a ServantRetentionPolicy object by using POA::create_servant_retention_policy() and passing the policy to POA::create_POA() to specify whether the created POA retains active servants in an active object map. This is a policy class derived from CORBA::Policy.

If no ServantRetentionPolicy value is specified at POA creation, the default value is RETAIN.

See Also PortableServer::RequestProcessingPolicy

```
// IDL
interface ServantRetentionPolicy : CORBA::Policy {
    readonly attribute ServantRetentionPolicyValue value;
};
// C++ in namespace PortableServer
    class IT_POA_API ServantRetentionPolicy :
        public virtual ::CORBA::Policy
    {
      public:

        typedef PortableServer::ServantRetentionPolicy_ptr
   _ptr_type;
        typedef PortableServer::ServantRetentionPolicy_var
   _var_type;
        virtual ~ServantRetentionPolicy();
        static ServantRetentionPolicy_ptr _narrow(
            CORBA::Object_ptr obj
        );
        static ServantRetentionPolicy_ptr _unchecked_narrow(
            CORBA::Object_ptr obj
        );
        inline static ServantRetentionPolicy_ptr _duplicate(
            ServantRetentionPolicy_ptr p
        );
```

```
        inline static ServantRetentionPolicy_ptr _nil();

        virtual ServantRetentionPolicyValue value() = 0;

        static const IT_FWString  _it_fw_type_id;
    };
```

See page 4 for descriptions of the standard helper functions:

- `_duplicate()`
- `_narrow()`
- `_nil()`
- `_unchecked_narrow()`

ServantRetentionPolicy::value()

```
// C++
virtual ServantRetentionPolicyValue value() = 0;
```

Returns the value of this POA policy.

PortableServer::ThreadPolicy Interface

You obtain a `ThreadPolicy` object by using `POA::create_thread_policy()` and passing the policy to `POA::create_POA()` to specify the threading model used with the created POA. This is a policy class derived from `CORBA::Policy`.

```
// IDL
interface ThreadPolicy : CORBA::Policy {
    readonly attribute ThreadPolicyValue value;
};

// C++ in namespace PortableServer

    class IT_POA_API ThreadPolicy :
        public virtual ::CORBA::Policy
    {
      public:

        typedef PortableServer::ThreadPolicy_ptr _ptr_type;
        typedef PortableServer::ThreadPolicy_var _var_type;
        virtual ~ThreadPolicy();
        static ThreadPolicy_ptr _narrow(
            CORBA::Object_ptr obj
        );
        static ThreadPolicy_ptr _unchecked_narrow(
            CORBA::Object_ptr obj
        );
        inline static ThreadPolicy_ptr _duplicate(
            ThreadPolicy_ptr p
        );
        inline static ThreadPolicy_ptr _nil();

        virtual ThreadPolicyValue value() = 0;

        static const IT_FWString _it_fw_type_id;
    };
```

See page 4 for descriptions of the standard helper functions:

- `_duplicate()`
- `_narrow()`
- `_nil()`
- `_unchecked_narrow()`

ThreadPolicy::value()

```
// C++
virtual ThreadPolicyValue value() = 0;
```

Returns the value of this POA policy.

IT_PortableServer Module

IT_PortableServer Overview

This module contains Orbix policy enhancements to the `PortableServer` module. The `IT_PortableServer` policies are:

- `ObjectDeactivationPolicy`
- `PersistenceModePolicy`
- `DispatchWorkQueuePolicy`

The `IT_PortableServer` module also contains the following common data structures and constants related to the policies:

- `OBJECT_DEACTIVATION_POLICY_ID`
- `ObjectDeactivationPolicyValue`
- `PERSISTENCE_MODE_POLICY_ID`
- `PersistenceModePolicyValue`
- `DISPATCH_WORKQUEUE_POLICY_ID`

IT_PortableServer::OBJECT_DEACTIVATION_POLICY_ID Constant

```
// IDL
const CORBA::PolicyType OBJECT_DEACTIVATION_POLICY_ID =
   IT_PolicyBase::IONA_POLICY_ID + 1;

// C++
IT_POA_API IT_NAMESPACE_STATIC const
    CORBA::ULong OBJECT_DEACTIVATION_POLICY_ID;
```

Defines a policy ID for object deactivation.

Enhancement This is an Orbix enhancement.

IT_PortableServer::ObjectDeactivationPolicyValue Enumeration

```
// IDL
```

```
enum ObjectDeactivationPolicyValue {
    DISCARD,
    DELIVER,
    HOLD
};

// C++
enum ObjectDeactivationPolicyValue {
    DISCARD,
    DELIVER,
    HOLD,
    _dummy_ObjectDeactivationPolicyValue = 0x80000000
};
```

An object deactivation policy value. Valid values consist of:

```
DISCARD
DELIVER
HOLD
```

Enhancement This is an Orbix enhancement.

See Also IT_PortableServer::ObjectDeactivationPolicy

IT_PortableServer::PERSISTENCE_MODE_POLICY_ID Constant

```
// IDL
const CORBA::PolicyType PERSISTENCE_MODE_POLICY_ID =
   IT_PolicyBase::IONA_POLICY_ID + 3;

// C++
IT_POA_API IT_NAMESPACE_STATIC const
    CORBA::ULong PERSISTENCE_MODE_POLICY_ID;
```

Defines a policy ID for the mode of object persistence.

Enhancement This is an Orbix enhancement.

IT_PortableServer::PersistenceModePolicyValue Enumeration

```
// IDL
enum PersistenceModePolicyValue {
```

```
    DIRECT_PERSISTENCE,
    INDIRECT_PERSISTENCE
};

enum PersistenceModePolicyValue {
    DIRECT_PERSISTENCE,
    INDIRECT_PERSISTENCE,
    _dummy_PersistenceModePolicyValue = 0x80000000
};
```

A persistence mode policy value. Valid values consist of:

```
DIRECT_PERSISTENCE
INDIRECT_PERSISTENCE
```

Enhancement This is an Orbix enhancement.

See Also IT_PortableServer::PersistenceModePolicy

IT_PortableServer::DISPATCH_WORKQUEUE_POLICY_ID Constant

```
const CORBA::PolicyType DISPATCH_WORKQUEUE_POLICY_ID =
IT_PolicyBase::IONA_POLICY_ID + 42;

// C++
IT_POA_API IT_NAMESPACE_STATIC const
    CORBA::ULong DISPATCH_WORKQUEUE_POLICY_ID;
```

Defines the policy ID for using WorkQueues to process ORB requests.

Enhancement This is an Orbix enhancement.

IT_PortableServer::ObjectDeactivationPolicy Class

This is an interface for a local policy object derived from CORBA::Policy. You create instances of ObjectDeactivationPolicy by calling CORBA::ORB::create_policy().

```
// IDL
interface ObjectDeactivationPolicy : CORBA::Policy {
    readonly attribute ObjectDeactivationPolicyValue value;
};
```

```
// C++ in namespace IT_PortableServer
class IT_POA_API ObjectDeactivationPolicy :
    public virtual ::CORBA::Policy {
  public:

    typedef IT_PortableServer::ObjectDeactivationPolicy_ptr
        _ptr_type;
    typedef IT_PortableServer::ObjectDeactivationPolicy_var
        _var_type;

    virtual ~ObjectDeactivationPolicy();

    static ObjectDeactivationPolicy_ptr _narrow(
        CORBA::Object_ptr obj
    );

    static ObjectDeactivationPolicy_ptr _unchecked_narrow(
        CORBA::Object_ptr obj
    );

    inline static ObjectDeactivationPolicy_ptr _duplicate(
        ObjectDeactivationPolicy_ptr p
    );

    inline static ObjectDeactivationPolicy_ptr _nil();
```

```
    virtual ObjectDeactivationPolicyValue value() = 0;

    static const IT_FWString _it_fw_type_id;
};
```

See page 5 for descriptions of the standard helper functions:

- `_duplicate()`
- `_narrow()`
- `_nil()`
- `_unchecked_narrow()`

ObjectDeactivationPolicy::~ObjectDeactivationPolicy() Destructor

```
// C++
virtual ~ObjectDeactivationPolicy();
```

The destructor.

Enhancement This is an Orbix enhancement.

ObjectDeactivationPolicy::value()

```
// C++
virtual ObjectDeactivationPolicyValue value() = 0;

// Java
public ObjectDeactivationPolicyValue value()
```

Returns the value of this object deactivation policy.

Enhancement This is an Orbix enhancement.

IT_PortableServer:: PersistenceModePolicy Class

This is an interface for a local policy object derived from CORBA::Policy. You create instances of PersistenceModePolicy by calling CORBA::ORB::create_policy().

```
// IDL
interface PersistenceModePolicy : CORBA::Policy {
    readonly attribute PersistenceModePolicyValue value;
};

// C++ in namespace IT_PortableServer
class IT_POA_API PersistenceModePolicy :
    public virtual ::CORBA::Policy {
  public:

    typedef IT_PortableServer::PersistenceModePolicy_ptr
   _ptr_type;
    typedef IT_PortableServer::PersistenceModePolicy_var
   _var_type;

    virtual ~PersistenceModePolicy();

    static PersistenceModePolicy_ptr _narrow(
        CORBA::Object_ptr obj
    );
    static PersistenceModePolicy_ptr _unchecked_narrow(
        CORBA::Object_ptr obj
    );
    inline static PersistenceModePolicy_ptr _duplicate(
        PersistenceModePolicy_ptr p
    );
    inline static PersistenceModePolicy_ptr _nil();

    virtual PersistenceModePolicyValue value() = 0;

    static const IT_FWString
```

```
    _it_fw_type_id;
};
```

See page 5 for descriptions of the standard helper functions:

- `_duplicate()`
- `_narrow()`
- `_nil()`
- `_unchecked_narrow()`

PersistenceModePolicy::~PersistenceModePolicy() Destructor

```
virtual ~PersistenceModePolicy();
```

The destructor.

Enhancement This is an Orbix enhancement.

PersistenceModePolicy::value()

```
// C++
virtual PersistenceModePolicyValue value() = 0;
```

Returns the value of this persistent mode policy.

Enhancement This is an Orbix enhancement.

IT_PortableServer::DispatchWorkQueuePolicy Interface

This is policy used to specify a `WorkQueue` to process ORB requests. It is derived from `CORBA::Policy`. You create instances of the policy by calling `CORBA::ORB::create_policy()`.

```
//IDL
local interface DispatchWorkQueuePolicy : CORBA::Policy
{
  readonly attribute IT_WorkQueue::WorkQueue workqueue;
}
```

WorkQueues

IT_WorkQueue Module

The IT_WorkQueue module defines the interfaces needed to create and manage user defined work queues.

WorkItem Interface

```
// IDL
enum WorkItemStatus
  {
    STOP_WORKING,
    CONTINUE_WORKING
  };

interface WorkItem
  {
    WorkItemStatus execute();
    void destroy();
  };
```

The WorkItem interface defines requests placed on the work queue. For most purposes, you do not need to implement this interface. The ORB will place requests on the queue and execute them under the covers. You can implement this interface if you want to have additional processing done by the work queues thread pool.

WorkItem::execute()

```
WorkItemStatus execute();
```

Processes the request encapsulated in the WorkItem object. The only times you need to call this method, is when using a ManualWorkQueue and removing items from the queue using dequeue(). Also, if you have made a custom WorkItem, you will need to implement this method.

WorkItem::Destroy

```
void destroy();
```

Releases the resources for the current WorkItem object.

WorkQueue Interface

```
// IDL
interface WorkQueue
  {
    readonly attribute long max_size;
    readonly attribute unsigned long count;

    boolean enqueue(in WorkItem work, in long timeout);
    boolean enqueue_immediate(in WorkItem work);
    boolean is_full();
    boolean is_empty();
    boolean activate();
    boolean deactivate();
    void flush();
    boolean owns_current_thread();
  };
```

The WorkQueue interface defines the base functionality for the ManualWorkQueue interface and the AutomaticWorkQueue interface.

max_size attribute

```
readonly attribute long max_size;
```

Specifies the maximum number of WorkItems a queue can hold before it is full.

WorkQueue::enqueue()

```
boolean enqueue(in WorkItem work, in long timeout);
```

Places work items into the queue for processing. For CORBA requests, the ORB takes care of placing items into the queue. For custom work items that you wish to handle in the queue, you must explicitly place them on the queue.

Parameters

`work`	The `WorkItem` to be placed into the queue.
`timeout`	The time in seconds that the item will be valid on the queue.

WorkQueue::enqueue_immediate()

```
boolean enqueue_immediate()
```

Returns TRUE and places the work item onto the queue for processing if the work queue is not full and the number of threads is below the high water mark. Effectively, this causes the work item to be processed immediately with out waiting for any current thread to complete. Returns FALSE if the work item cannot immediately placed on the work queue.

Parameters

`work`	The `WorkItem` to be placed into the queue.

WorkQueue::is_full()

```
boolean is_full();
```

Returns TRUE if the `WorkQueue` has reached `max_size`. Returns FALSE otherwise.

WorkQueue::is_empty()

```
boolean is_empty();
```

Returns TRUE if the `WorkQueue` is empty. Returns FALSE otherwise.

WorkQueue::activate()

```
boolean activate();
```

Puts the queue into a state where it is ready to receive and process work requests.

WorkQueue::deactivate()

```
boolean deactivate();
```

Puts the queue into a state where it will no longer process work requests.

WorkQueue::owns_current_thread()

```
boolean owns_current_thread();
```

Returns `TRUE` if the thread making the call is managed by the work queue.

WorkQueue::flush()

```
void flush();
```

Removes all of the items from the queue without processing them.

ManualWorkQueue Interface

```
// IDL
interface ManualWorkQueue : WorkQueue
  {
     boolean dequeue(out WorkItem work, in long timeout);
     boolean do_work(in long number_of_jobs, in long timeout);
     void shutdown(in boolean process_remaining_jobs);
  };
```

The `ManualWorkQueue` interface specifies the methods for managing a manual work queue.

ManualWorkQueue::dequeue()

```
boolean dequeue(out WorkItem work, in long timeout);
```

Removes a single `WorkItem` from the head of the queue. You must explicitly call `execute()` on the `WorkItem` to process the request using this method.

Parameters

`work`	The `WorkItem` returned by dequeue(). If the call is unsucessfull, `work` will be `NULL`.
`timeout`	The maximum amount of time the call will block before returning `NULL`.

ManualWorkQueue::do_work()

```
boolean do_work(in long number_of_jobs, in long timeout);
```

Removes the specified number of requests from the queue and processes them. If there are less than the specified number of items on the queue, `do_work()` will block for a specified amount of time to wait for items to be queued.

Parameters

`number_of_jobs`	The maximum number of items to process.
`timeout`	The maximum amount of time the call will block before returning.

ManualWorkQueue::shutdown()

```
void shutdown(in boolean process_remaining_jobs);
```

Deactivates the queue and releases all resources associated with it.

Parameters

`process_remainig_jobs`	`TRUE` specifies that any items in the queue should be processed before shutting down the queue. `FALSE` specifies that any items in the queue should be flushed.

ManualWorkQueueFactory Interface

```
// IDL
local interface ManualWorkQueueFactory
```

```
  {
    ManualWorkQueue create_work_queue(in long max_size);
  };
```

Defines the method used to obtain a ManualWorkQueue. The ManualWorkQueueFactory is obtained by calling resolve_initial_references("IT_ManualWorkQueueFactory").

ManualWorkQueueFactory::create_work_queue()

```
ManualWorkQueue create_work_queue(in long max_size);
```

Creates a ManualWorkQueue object.

Parameters

max_size	Specifies the maximum number of work items the queue can hold.

AutomaticWorkQueue Interface

```
// IDL
interface AutomaticWorkQueue : WorkQueue
  {
    readonly attribute unsigned long threads_total;
    readonly attribute unsigned long threads_working;

    attribute long high_water_mark;
    attribute long low_water_mark;

    void shutdown(in boolean process_remaining_jobs);
  };
```

The AutomaticWorkQueue interface specifies the method used to shutdown an automatic work queue. It also specifies the attributes that limit the size of the queue's thread pool and monitor thread usage.

threads_total Attribute

```
readonly attribute unsigned long threads_total;
```

The total number of threads in the AutomaticWorkqueue which can process work items. This will indicate how many threads the workqueue currently has if it has been configured to dynamically create and destroy threads as the workload changes.

threads_working Attribute

```
readonly attribute unsigned long threads_working;
```

Indicates the total number of threads that are busy processing work items at that point in time. This value will vary as the workload of the server changes.

high_water_mark Attribute

```
attribute long high_water_mark;
```

Specifies the maximum number of threads an AutomaticWorkQueue instance can have in its active thread pool.

low_water_mark Attribute

```
attribute long low_water_mark;
```

Specifies the minimum number of threads available to an AutomaticWorkQueue instance.

AutomaticWorkQueue::shutdown()

```
void shutdown(in boolean process_remaining_jobs);
```

Deactivates the queue and releases all resources associated with it.

Parameters

`process_remainig_jobs` `TRUE` specifies that any items in the queue should be processed before shutting down the queue.

`FALSE` specifies that any items in the queue should be flushed.

AutomaticWorkQueueFactory Interface

```
// IDL
local interface AutomaticWorkQueueFactory
  {
    AutomaticWorkQueue create_work_queue(
      in long          max_size,
      in unsigned long initial_thread_count,
      in long          high_water_mark,
      in long          low_water_mark);

    AutomaticWorkQueue create_work_queue_with_thread_stack_size(
      in long          max_size,
      in unsigned long initial_thread_count,
      in long          high_water_mark,
      in long          low_water_mark,
      in long          thread_stack_size);
  };
```

The `AutomaticWorkQueueFactory` interface specifies two methods for obtaining an `AutomaticWorkQueue`. The `AutomaticWorkQueueFactory` is obtained by calling `resolve_initial_references("IT_AutomaticWorkQueueFactory")`.

AutomaticWorkQueueFactory::create_work_queue()

```
AutomaticWorkQueue create_work_queue(
      in long          max_size,
      in unsigned long initial_thread_count,
      in long          high_water_mark,
      in long          low_water_mark);
```

Creates an AutomaticWorkQueue.

Parameters

max_size	The maximum number of items the queue can hold.
initial_thread_count	The initial number of threads the queue has available for processing work items.
high_water_mark	The maximum number of threads the queue can generate to process work items.
low_water_mark	The minimum number of threads the queue can have available to process work items.

AutomaticWorkQueueFactory:: create_work_queue_with_thread_stack_size()

```
AutomaticWorkQueue create_work_queue_with_thread_stack_size(
      in long          max_size,
      in unsigned long initial_thread_count,
      in long          high_water_mark,
      in long          low_water_mark,
      in long          thread_stack_size);
```

Creates an AutomaticWorkQueue and specify the size of the thread stack.

Parameters

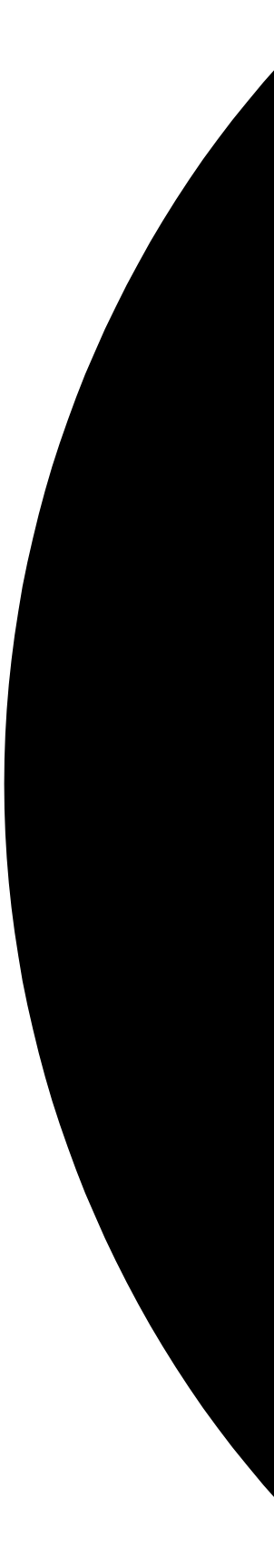
max_size	The maximum number of items the queue can hold.
initial_thread_count	The initial number of threads the queue has available for processing work items.
high_water_mark	The maximum number of threads the queue can generate to process work items.
low_water_mark	The minimum number of threads the queue can have available to process work items.
thread_stack_size	The size, in bytes, of the thread stack used by the queue.

WorkQueuePolicy Interface

```
// IDL
local interface WorkQueuePolicy : CORBA::Policy
  {
    readonly attribute WorkQueue work_queue;
  };
```

The `WorkPolicy` interface is the object you pass to `create_policy()` when associating you `WorkQueue` with a POA.

Threading and Synchronization Toolkit



Threading and Synchronization Toolkit Overview

The Threading and Synchronization (TS) toolkit provides an object-oriented and platform-neutral abstraction that hides the diverse, lower-level, thread packages. Table 18 shows the threading and synchronization (TS) classes organized into some useful groups.

Table 18: *TS Thread Classes*

Thread Management	IT_CurrentThread IT_Thread IT_ThreadBody IT_ThreadFactory IT_TerminationHandler IT_TSVoidStar
Thread Errors and Exceptions	IT_TSBadAlloc IT_DefaultTSErrorHandler IT_TSError IT_TSErrorHandler IT_TSLogic IT_TSRuntime
Mutex Locks	IT_Locker IT_Mutex IT_PODMutex IT_RecursiveMutex IT_RecursiveMutexLocker
Thread Synchronization	IT_Condition IT_Gateway IT_Semaphore IT_TimedCountByNSemaphore IT_TimedOneshot IT_TimedSemaphore

The rest of this overview covers these topics:

Timeouts

Timeouts are expressed in milliseconds. They represent the time period from the invocation of the timed method until the expiration of the timer. This time-out period is approximate because it is affected by the number and kind of interrupts received and by the changes external sources may make to the system's time.

Execution Modes

The TS classes are designed to be efficient and to help you write code that is correct and portable across various platforms. You can build TS applications in either of the following modes:

Unchecked	This is the normal production mode. Inexpensive checks, such as checking values returned by the API, are performed, but a minimum of memory, locking, and system calls are used to implement TS features.
Checked	In this mode, extra-checking is performed to detect erroneous or non-portable situations. On platforms that support exceptions, exceptions are raised to report such errors. This mode may be less time or space efficient than the unchecked mode.

The effect of a program that runs correctly (the program does not create any TS error object) in the checked mode is identical to that of the unchecked mode.

TS provides two kinds of classes in different sets of header files. These include wrapper and inline classes.

Wrapper Classes

Wrapper classes are the recommended classes to use because you can switch between checked and unchecked modes by simply re-linking without recompiling your application. These clean, platform-neutral wrapper classes simply delegate to the appropriate inlined classes for whichever mode you are using.

The wrapper classes are in header files ending in `.h`.

Inlined Classes

To minimize the delegation overhead of wrapper classes, the TS toolkit also provides C++ classes with only inlined member methods and pre-preprocessor directives. These inline classes accommodate the differences between the underlying thread packages.

Delegation overhead for a normal method call is generally negligible, but you can save on this overhead by using these inlined classes directly. However by using these header files, you will need to recompile your application whenever you want to switch between checked and unchecked modes, and each time even minor improvements are made to the TS implementation.

The inline classes are in header files ending in `_i.h`.

Setting an Execution Mode

Table 19 shows the default settings for each platform.

Table 19: *Default Thread Settings*

Platform	Thread Primitives	Default Mode
HPUX 11 Solaris 2.6	Posix	unchecked
HPUX 10.20	DCE	unchecked
Other Solaris	UI	unchecked
Win32	Win32	unchecked

To set a different mode, you reset the library by inserting the preferred `lib` subdirectory at the beginning of your `LD_LIBRARY_PATH` or `SHLIB_PATH`. For example, to reset to the checked mode, do the following for your respective platform:

Solaris	Put the following at the beginning of your `LD_LIBRARY_PATH`: `/vob/common/ts/lib/posix/checked`
HPUX 10.20	Put the following at the beginning of your `SHLIB_PATH`: `/vob/common/ts/lib/dce/checked`
HPUX 11.00	Put the following at the beginning of your `SHLIB_PATH`: `/vob/common/ts/lib/posix/checked`
NT	Put the following at the beginning of your `PATH`: `/common/ts/lib/win32/checked`

Errors and Exceptions

Table 20 summarizes the TS error classes:

Table 20: *Error and Exception Classes*

Control	Exceptions
`IT_DefaultTSErrorHandler`	`IT_TSBadAlloc`
`IT_TSError`	`IT_TSLogic`
`IT_TSErrorHandler`	`IT_TSRuntime`

The TS API allows you to use either error parameters or exceptions. The last parameter of almost every TS method is a reference to an error handler object of the class `IT_TSErrorHandler`. When a TS method detects an error, it creates an `IT_TSError` object and passes it to `IT_TSErrorHandler::handle()`.

TS errors form the hierarchy shown in Figure 1. An `IT_TSRuntime` error generally signals an error detected by the operating system or the underlying thread package. An `IT_TSLogic` error reports a logic error in your program, for example, when a thread tries to release a lock it does not own. Logic

errors are either detected by the underlying thread package, or by extra checking code in checked mode. An IT_TSBadAlloc error signals that the new operator failed.

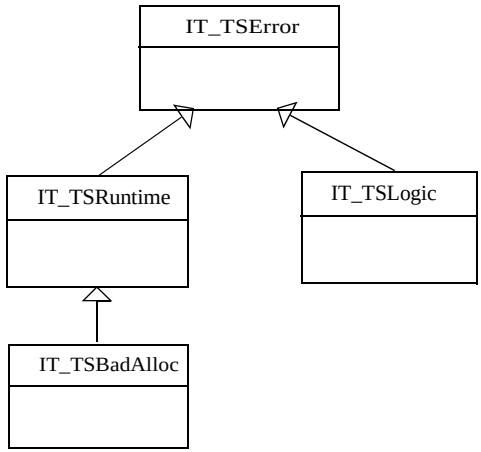


Figure 1: *The TS Error Class Hierarchy*

The TS API provides a default, static, and stateless error handler named IT_DefaultTSErrorHandler. If you use exceptions, this error handler throws IT_TSError objects. In environments that do not use exceptions this handler aborts the process.

For most applications, the default error handler object provides the desired behavior. In this situation, instead of passing an IT_DefaultTSErrorHandler object each time you call a TS method, you can define in your build command the environment variable IT_TS_DEFAULTED. This will instruct the TS API to use the default error handler object for the error handler parameter. For example:

```
#ifndef IT_TS_DEFAULT_ERROR_HANDLER
#ifdef IT_TS_DEFAULTED
#define IT_TS_DEFAULT_ERROR_HANDLER = IT_DefaultTSErrorHandler
#else
#define IT_TS_DEFAULT_ERROR_HANDLER
#endif
#endif
```

C++ destructors do not have parameters, and as result, cannot be given an error handler object parameter. In the checked mode, the TS API reports errors in destructors to the default error handler object. In the unchecked mode, the TS API does not report errors that occur in destructors.

Because default parameters are not part of the function-type in C++, the TS library can be built with or without defining `IT_TS_DEFAULTED`. Also, the same library can be used by modules that use the defaulted parameter and by modules built without defining `IT_TS_DEFAULTED`.

If you intend to use your own error handler objects in your application, it is strongly recommended that you do not define `IT_TS_DEFAULTED` to avoid using the default error handler object by mistake. If you want to consistently use the same error handler object, you can define `IT_TS_DEFAULT_ERROR_HANDLER` in your command or in a non-exported file. For example:

```
#define IT_TS_DEFAULT_ERROR_HANDLER = myErrorHandler;
```

IT_Condition Class

The IT_Condition class provides a signalling mechanism that events use to synchronize when sharing a mutex. In one atomic operation, a condition wait both releases the mutex and waits until another thread signals or broadcasts a change of state for the condition.

```
class IT_Condition {
public:
    IT_Condition(
        IT_TSErrorHandler& eh IT_TS_DEFAULT_ERROR_HANDLER
    );
    ~IT_Condition();
    void wait(
        IT_Mutex& app_mutex,
        IT_TSErrorHandler& eh IT_TS_DEFAULT_ERROR_HANDLER
    );
    void wait(
        IT_MutexLocker& locker,
        IT_TSErrorHandler& eh IT_TS_DEFAULT_ERROR_HANDLER
    );
    void signal(
        IT_TSErrorHandler& eh IT_TS_DEFAULT_ERROR_HANDLER
    );
    void broadcast(
        IT_TSErrorHandler& eh IT_TS_DEFAULT_ERROR_HANDLER
    );
};
```

IT_Condition::broadcast()

```
void broadcast(
    IT_TSErrorHandler& eh IT_TS_DEFAULT_ERROR_HANDLER
);
```

Wakes up all waiting threads. One thread acquires the mutex and resumes with the associated mutex lock. The rest of the threads continue waiting.

Parameters

eh	A reference to an error handler object.

Enhancement Orbix enhancement.

See Also IT_Mutex

IT_Condition::IT_Condition() Constructor

```
IT_Condition(
    IT_TSErrorHandler& eh IT_TS_DEFAULT_ERROR_HANDLER
);
```

The constructor for an `IT_Condition` object.

Parameters

eh	A reference to an error handler object.

Enhancement Orbix enhancement.

IT_Condition::~IT_Condition() Destructor

```
~IT_Condition();
```

The destructor for an `IT_Condition` object.

Enhancement Orbix enhancement.

IT_Condition::signal()

```
void signal(
    IT_TSErrorHandler& eh IT_TS_DEFAULT_ERROR_HANDLER
);
```

Wakes up a single waiting thread. The thread resumes with the associated mutex locked.

Parameters

eh	A reference to an error handler object.

Enhancement Orbix enhancement.

IT_Condition::wait()

```
void wait(
    IT_Mutex& app_mutex,
    IT_TSErrorHandler& eh IT_TS_DEFAULT_ERROR_HANDLER
);

void wait(
    IT_MutexLocker& locker,
    IT_TSErrorHandler& eh IT_TS_DEFAULT_ERROR_HANDLER
);
```

Atomically releases the mutex, and waits until another thread calls `signal()` or `broadcast()`.

Parameters

`app_mutex`	Use the mutex `app_mutex`.
`locker`	Use the mutex in `locker`.
`eh`	

The mutex must always be locked when `wait()` is called. When a condition wakes up from a wait, it resumes with the mutex locked.

Enhancement Orbix enhancement.

IT_CurrentThread Class

The `IT_CurrentThread` class gives access to the current thread. It has only static member methods.

```
class IT_TS_API IT_CurrentThread {
public:
    static IT_Thread self(
        IT_TSErrorHandler& eh IT_TS_DEFAULT_ERROR_HANDLER
    );

    static int is_main_thread();

    static void cleanup();

    static void yield();

    static void sleep(
        unsigned long milliseconds
    );

    static long id();
};
```

IT_CurrentThread::cleanup()

```
static void cleanup();
```

Cleans up thread-specific data. A thread typically calls `cleanup()` before exiting. Threads created with an `IT_ThreadFactory` do this automatically.

Enhancement Orbix enhancement.

IT_CurrentThread::id()

```
static long id();
```

Returns a unique identifier for the current thread.

Enhancement Orbix enhancement.

IT_CurrentThread::is_main_thread()

```
static int is_main_thread();
```

Returns 1 if the caller is the main thread, but returns 0 if it is not.

Enhancement Orbix enhancement.

IT_CurrentThread::self()

```
static IT_Thread self(
    IT_TSErrorHandler& eh IT_TS_DEFAULT_ERROR_HANDLER
);
```

Returns an `IT_Thread` object for the thread that calls this method.

Parameters

eh	A reference to an error handler object.

Enhancement Orbix enhancement.

IT_CurrentThread::sleep()

```
static void sleep(
    unsigned long milliseconds
);
```

Suspends the current thread for the approximate number of milliseconds input.

Parameters

`milliseconds`	The length of time in milliseconds to suspend the thread.

Enhancement Orbix enhancement.

IT_CurrentThread::yield()

```
static void yield();
```

Yields the CPU to another thread of equal priority, if one is available.

Enhancement Orbix enhancement.

IT_DefaultTSErrorHandler Class

The `IT_DefaultTSErrorHandler` class is the default TS error handler. If you use exceptions, this error handler throws `IT_TSError` objects. In environments that do not use exceptions this handler aborts the process.

```
class IT_DefaultTSErrorHandler : public IT_TSErrorHandler{
public:
    virtual ~IT_DefaultTSErrorHandler()
    virtual void handle(
        const IT_TSError& this_error
    );
};
```

See page 692 for more on error handling.

IT_DefaultTSErrorHandler::handle()

```
void handle(
    const IT_TSError& this_error
);
```

Do appropriate processing for the given error.

Parameters

`this_error` A reference to an error object.

Enhancement Orbix enhancement.

IT_DefaultTSErrorHandler::~IT_DefaultTSErrorHandler() Destructor

```
~IT_DefaultTSErrorHandler()
```

The destructor for the error handler object.

Enhancement Orbix enhancement.

IT_Gateway Class

The IT_Gateway class provides a gate where a set of threads can only do work if the gate is open.

```
class IT_Gateway {
public:
    IT_Gateway(
        IT_TSErrorHandler& eh IT_TS_DEFAULT_ERROR_HANDLER
    );
    ~IT_Gateway();

    void open(
        IT_TSErrorHandler& eh IT_TS_DEFAULT_ERROR_HANDLER
    );

    void close(
        IT_TSErrorHandler& eh IT_TS_DEFAULT_ERROR_HANDLER
    );

    void wait(
        IT_TSErrorHandler& eh IT_TS_DEFAULT_ERROR_HANDLER
    );

private:
...
```

IT_Gateway::close()

```
void close(
    IT_TSErrorHandler& eh IT_TS_DEFAULT_ERROR_HANDLER
);
```

Close the gateway so no threads can do any work.

Parameters

eh	A reference to an error handler object.

Enhancement Orbix enhancement.

IT_Gateway::IT_Gateway() Constructor

```
IT_Gateway(
    IT_TSErrorHandler& eh IT_TS_DEFAULT_ERROR_HANDLER
);
```

The gateway constructor.

Parameters

eh	A reference to an error handler object.

Enhancement Orbix enhancement.

IT_Gateway::~IT_Gateway() Destructor

```
~IT_Gateway();
```

The destructor.

Enhancement Orbix enhancement.

IT_Gateway::open()

```
void open(
    IT_TSErrorHandler& eh IT_TS_DEFAULT_ERROR_HANDLER
);
```

Open the gateway to allow threads to work.

Parameters

eh	A reference to an error handler object.

Enhancement Orbix enhancement.

IT_Gateway::wait()

```
void wait(
    IT_TSErrorHandler& eh IT_TS_DEFAULT_ERROR_HANDLER
);
```

Wait for a thread to finish.

Parameters

eh	A reference to an error handler object.

Enhancement Orbix enhancement.

IT_Locker Template Class

IT_Locker is a helper class for locking and unlocking non-recursive mutexes, including IT_Mutex and IT_PODMutex objects. Typically a locker locks a mutex in its constructor and releases it in its destructor. This is particularly useful for writing clean code that behaves properly when an exception is raised.

An IT_Locker object must be created on the stack of a particular thread, and must never be shared by more than one thread.

The IT_Locker method definitions are inlined directly in the class declaration, because these methods call each other. If a definition calls a method that is not previously declared inlined, this method is generated out of line, regardless of its definition (which can be provided later in the translation unit with the inline keyword).

```
template<class T> class IT_Locker {
public:
    IT_Locker(
        T& mutex,
        IT_TSErrorHandler& eh IT_TS_DEFAULT_ERROR_HANDLER
    ) :
        m_mutex(mutex),
        m_locked(0),
        m_error_handler(eh)
        {
            lock();
        }

    IT_Locker(
        T& mutex,
        int wait,
        IT_TSErrorHandler& eh IT_TS_DEFAULT_ERROR_HANDLER
    ) :
        m_mutex(mutex),
        m_locked(0),
        m_error_handler(eh)
        {
            if (wait)
```

```
        {
            lock();
        }
        else
        {
            trylock();
        }
    }

~IT_Locker()
{
    cancel();
}

void cancel()
{
    if (m_locked)
    {
        m_mutex.unlock(m_error_handler);
        m_locked = 0;
    }
}

int is_locked()
{
    return m_locked;
}

void lock()
{
    m_mutex.lock(m_error_handler);
    m_locked = 1;
}

int trylock()
{
    return (m_locked = m_mutex.trylock(m_error_handler));
}

T& mutex()
{
    return m_mutex;
}
```

```
private:
...
```

IT_Locker::cancel()

```
void cancel() {
    if (m_locked)
    {
        m_mutex.unlock(m_error_handler);
        m_locked = 0;
    }
}
```

Releases the mutex only if it is locked by this locker. You can call `cancel()` safely even when the mutex is not locked.

Enhancement Orbix enhancement.

Exceptions Errors that can be reported include:

IT_TSRuntime
IT_TSLogic

IT_Locker::is_locked()

```
int is_locked() {
    return m_locked;
}
```

returns 1 if this mutex locker has the lock and returns 0 if it does not.

Enhancement Orbix enhancement.

IT_Locker::IT_Locker()

```
IT_Locker(
    T& mutex,
    IT_TSErrorHandler& eh IT_TS_DEFAULT_ERROR_HANDLER
) :
    m_mutex(mutex),
    m_locked(0),
```

```
    m_error_handler(eh)
{
    lock();
}
```

A constructor for a locker object that locks the given mutex.

```
IT_Locker(
    T& mutex,
    int wait,
    IT_TSErrorHandler& eh IT_TS_DEFAULT_ERROR_HANDLER
) :
    m_mutex(mutex),
    m_locked(0),
    m_error_handler(eh)
{
    if (wait)
    {
        lock();
    }
    else
    {
        trylock();
    }
}
```

A constructor for a locker object.

Parameters

`mutex`	The mutex to which the locker applies.
`wait`	If wait has a value of 1, this constructor waits to acquire the lock. If wait has a value of 0, the constructor only tries to lock the mutex.
`eh`	A reference to an error handler object.

Enhancement Orbix enhancement.

See Also `IT_Locker::trylock()`

IT_Locker::~IT_Locker()

```
~IT_Locker()
{
    cancel();
}
```

The destructor releases the mutex if it is locked by this locker.

Enhancement Orbix enhancement.

Exceptions Errors that can be reported include:

IT_TSLogic
IT_TSRuntime

IT_Locker::lock()

```
void lock()
{
    m_mutex.lock(m_error_handler);
    m_locked = 1;
}
```

Locks the mutex associated with the locker.

Enhancement Orbix enhancement.

Exceptions Errors that can be reported include:

IT_TSLogic
IT_TSRuntime

IT_Locker::mutex()

```
T& mutex()
{
    return m_mutex;
}
```

Returns direct access to the locker's mutex.

Enhancement Orbix enhancement.

IT_Locker::trylock()

```
int trylock()
{
    return (m_locked = m_mutex.trylock(m_error_handler));
}
```

Tries to lock the mutex. Returns 1 if the mutex is successfully locked or 0 if it is not locked.

Enhancement Orbix enhancement.

Exceptions Errors that can be reported include:

IT_TSLogic
IT_TSRuntime

IT_Mutex Class

An IT_Mutex object is a synchronization primitive for mutual exclusion locks.

When a thread has successfully locked, it is said to own the IT_Mutex. IT_Mutex objects have scope only within a single process (they are not shared by several processes) and they are not recursive. When a thread that owns an IT_Mutex attempts to lock it again, a deadlock occurs.

You use an IT_Mutex in conjunction with an IT_Locker object to lock and unlock your mutexes.

```
class IT_Mutex {
public:
    IT_Mutex(
        IT_TSErrorHandler& eh IT_TS_DEFAULT_ERROR_HANDLER
    );

    ~IT_Mutex();

    void lock(
        IT_TSErrorHandler& eh IT_TS_DEFAULT_ERROR_HANDLER
    );

    void unlock(
        IT_TSErrorHandler& eh IT_TS_DEFAULT_ERROR_HANDLER
    );

    int trylock(
        IT_TSErrorHandler& eh IT_TS_DEFAULT_ERROR_HANDLER
    );

private:
// …
};
```

See Also

IT_Locker
IT_RecursiveMutex

IT_Mutex::IT_Mutex() Constructor

```
IT_Mutex(I
    IT_TSErrorHandler& eh IT_TS_DEFAULT_ERROR_HANDLER
);
```

Constructs an IT_Mutex object. It is initially unlocked.

Parameters

eh	A reference to an error handler object.

Enhancement Orbix enhancement.

Exceptions The IT_TSRuntime error can be reported.

IT_Mutex::~IT_Mutex() Destructor

```
IT_Mutex();
```

The destructor for the mutex.

Enhancement Orbix enhancement.

IT_Mutex::lock()

```
void lock(
    IT_TSErrorHandler& eh IT_TS_DEFAULT_ERROR_HANDLER
);
```

Blocks until the IT_Mutex can be acquired.

Parameters

eh	A reference to an error handler object.

Enhancement Orbix enhancement.

Exceptions Errors that can be reported include:

IT_TSLogic
IT_TSRuntime

IT_Mutex::trylock()

```
int trylock(
    IT_TSErrorHandler& eh IT_TS_DEFAULT_ERROR_HANDLER
);
```

Tries to acquire the lock. If successful, the method returns a 1 immediately, otherwise it returns a 0 and does not block.

Parameters

eh	A reference to an error handler object.

Enhancement Orbix enhancement.

Exceptions Errors that can be reported include:

IT_TSLogic
IT_TSRuntime

IT_Mutex::unlock()

```
void unlock(
    IT_TSErrorHandler& eh IT_TS_DEFAULT_ERROR_HANDLER
);
```

Releases this `IT_Mutex`. Only the owner thread of an `IT_Mutex` is allowed to release an `IT_Mutex`.

Parameters

eh	A reference to an error handler object.

Enhancement Orbix enhancement.

Exceptions Errors that can be reported include:

IT_TSLogic
IT_TSRuntime

IT_PODMutex Structure

An IT_PODMutex is a mutex for a “plain old data” (POD) structure. Just as with a standard C++ PODS, an IT_PODMutex can be fully initialized at compile time without the overhead of an explicit constructor call. This is particularly useful for static objects. Likewise, the object can be destroyed without an explicit destructor call (in a manner similar to the C language).

You can use the built-in definition IT_POD_MUTEX_INIT to easily initialize an IT_PODMutex to zero. For example:

```
static IT_PODMutex my_global_mutex = IT_POD_MUTEX_INIT;
```

You use an IT_PODMutex in conjunction with an IT_Locker object to lock and unlock your mutexes. The structure members for an IT_PODMutex include the following:

```
struct IT_TS_API IT_PODMutex {
    void lock(
        IT_TSErrorHandler& eh IT_TS_DEFAULT_ERROR_HANDLER
    );
    int trylock(
        IT_TSErrorHandler& eh IT_TS_DEFAULT_ERROR_HANDLER
    );
    void unlock(
        IT_TSErrorHandler& eh IT_TS_DEFAULT_ERROR_HANDLER
    );
    // DO NOT USE and DO NOT MAKE PRIVATE
    unsigned char m_index;
};
```

See Also

IT_Locker
IT_Mutex

IT_PODMutex::lock()

```
void lock(
    IT_TSErrorHandler& eh IT_TS_DEFAULT_ERROR_HANDLER
);
```

Blocks until the mutex can be acquired.

Parameters

eh	A reference to an error handler object.

Enhancement Orbix enhancement.

Exceptions Errors that can be reported include:

IT_TSLogic
IT_TSRuntime

IT_PODMutex::m_index Data Type

```
unsigned char m_index;
```

Note: For internal use only.

IT_PODMutex::trylock()

```
int trylock(
    IT_TSErrorHandler& eh IT_TS_DEFAULT_ERROR_HANDLER
);
```

Tries to acquire the mutex lock. If `trylock()` succeeds, it returns a 1 immediately. Otherwise it returns 0.

Parameters

eh	A reference to an error handler object.

Enhancement Orbix enhancement.

Exceptions Errors that can be reported include:

IT_TSLogic
IT_TSRuntime

IT_PODMutex::unlock()

```
void unlock(
    IT_TSErrorHandler& eh IT_TS_DEFAULT_ERROR_HANDLER
);
```

Releases the mutex lock. Only the owner of a mutex is allowed to release it.

Parameters

eh	A reference to an error handler object.

Enhancement Orbix enhancement.

Exceptions Errors that can be reported include:

IT_TSLogic
IT_TSRuntime

IT_RecursiveMutex Class

An IT_RecursiveMutex object is a synchronization primitive for mutual exclusion. In general do not used it directly.

Note: It is strongly recommended that you use the IT_RecursiveMutexLocker to lock and unlock your recursive mutexes.

In most respects an IT_RecursiveMutex object is similar to an IT_Mutex object. However, it can be locked recursively, which means that a thread that already owns a recursive mutex object can lock it again in a deeper scope without creating a deadlock condition.

When a thread has successfully locked a recursive mutex, it is said to own it. Recursive mutex objects have process-scope which means that they are not shared by several processes.

To release an IT_RecursiveMutex, its owner thread must call unlock() the same number of times that it called lock().

```
class IT_RecursiveMutex {
public:
    IT_RecursiveMutex(
        IT_TSErrorHandler& eh IT_TS_DEFAULT_ERROR_HANDLER
    );

    ~IT_RecursiveMutex();

    void lock(
        IT_TSErrorHandler& eh IT_TS_DEFAULT_ERROR_HANDLER
    );

    void unlock(
        IT_TSErrorHandler& eh IT_TS_DEFAULT_ERROR_HANDLER
    );

    int trylock(
        IT_TSErrorHandler& eh IT_TS_DEFAULT_ERROR_HANDLER
```

```
    );
private:
...
```

See Also IT_Mutex
IT_RecursiveMutexLocker

IT_RecursiveMutex::IT_RecursiveMutex() Constructor

```
IT_RecursiveMutex(
    IT_TSErrorHandler& eh IT_TS_DEFAULT_ERROR_HANDLER
);
```

Constructs an IT_RecursiveMutex object. It is initially unlocked.

Parameters

eh	A reference to an error handler object.

Enhancement Orbix enhancement.

Exceptions The IT_TSRuntime error can be reported.

IT_RecursiveMutex::~IT_RecursiveMutex() Destructor

```
~IT_RecursiveMutex();
```

Destructor for an IT_RecursiveMutex object.

Enhancement Orbix enhancement.

IT_RecursiveMutex::lock()

```
void lock(
    IT_TSErrorHandler& eh IT_TS_DEFAULT_ERROR_HANDLER
);
```

Blocks until the recursive mutex can be acquired.

Parameters

eh	A reference to an error handler object.

Enhancement Orbix enhancement.

Exceptions The IT_TSRuntime error can be reported.

IT_RecursiveMutex::trylock()

```
int trylock(
    IT_TSErrorHandler& eh IT_TS_DEFAULT_ERROR_HANDLER
);
```

Tries to acquire the recursive mutex. If it succeeds, returns 1 immediately; otherwise returns 0.

Parameters

eh	A reference to an error handler object.

Enhancement Orbix enhancement.

Exceptions The IT_TSRuntime error can be reported.

IT_RecursiveMutex::unlock()

```
void unlock(
    IT_TSErrorHandler& eh IT_TS_DEFAULT_ERROR_HANDLER
);
```

Releases this recursive mutex (one count). Only the owner of a mutex is allowed to release it.

Parameters

eh	A reference to an error handler object.

Enhancement Orbix enhancement.

Exceptions Errors that can be reported include:

IT_TSRuntime
IT_TSLogic

IT_RecursiveMutexLocker Class

The IT_RecursiveMutexLocker is a locker for recursive mutexes. The IT_RecursiveMutexLocker methods are defined as inline in the class declaration, because these methods call each other.

```
class IT_RecursiveMutexLocker {
public:
    IT_RecursiveMutexLocker(
        IT_RecursiveMutex& m,
        IT_TSErrorHandler& eh IT_TS_DEFAULT_ERROR_HANDLER
    ) :
        m_recursive_mutex(m),
        m_lock_count(0),
        m_error_handler(eh)
    {
        lock();
    }

    IT_RecursiveMutexLocker(
        IT_RecursiveMutex& m,
        int wait,
        IT_TSErrorHandler& eh IT_TS_DEFAULT_ERROR_HANDLER
    ) :
        m_recursive_mutex(m),
        m_lock_count(0),
        m_error_handler(eh)
    {
        if (wait)
        {
            lock();
        }
        else
        {
            trylock();
        }
    }

    ~IT_RecursiveMutexLocker()
```

```
{
    cancel();
}

void cancel()
{
    while (m_lock_count > 0)
    {
        m_recursive_mutex.unlock(m_error_handler);
        m_lock_count--;
    }
}

void lock()
{
    m_recursive_mutex.lock(m_error_handler);
    m_lock_count++;
}

unsigned int lock_count()
{
    return m_lock_count;
}

int trylock()
{
    if (m_recursive_mutex.trylock(m_error_handler) == 1)
    {
        m_lock_count++;
        return 1;
    }
    else
    {
        return 0;
    }
}

void unlock()
{
    m_recursive_mutex.unlock(m_error_handler);
    m_lock_count--;
}
```

```
    IT_RecursiveMutex& mutex()
    {
        return m_recursive_mutex;
    }

Private:
...
```

IT_RecursiveMutexLocker::cancel()

```
void cancel() {
    while (m_lock_count > 0)
    {
        m_recursive_mutex.unlock(m_error_handler);
        m_lock_count--;
    }
}
```

Releases all locks held by this recursive mutex locker. The `cancel()` method can be called safely even when the recursive mutex is not locked.

Enhancement Orbix enhancement.

IT_RecursiveMutexLocker::IT_RecursiveMutexLocker() Constructors

```
IT_RecursiveMutexLocker(
    IT_RecursiveMutex& m,
    IT_TSErrorHandler& eh IT_TS_DEFAULT_ERROR_HANDLER
) :
    m_recursive_mutex(m),
    m_lock_count(0),
    m_error_handler(eh)
{
    lock();
}
```

Constructs a recursive mutex locker object. This constructor locks the given recursive mutex.

```
IT_RecursiveMutexLocker(
    IT_RecursiveMutex& m,
```

```
    int wait,
    IT_TSErrorHandler& eh IT_TS_DEFAULT_ERROR_HANDLER
) :
    m_recursive_mutex(m),
    m_lock_count(0),
    m_error_handler(eh)
{
    if (wait)
    {
        lock();
    }
    else
    {
        trylock();
    }
}
```

Constructs a recursive mutex locker object.

Parameters

`m`	The mutex to which the locker applies.
`wait`	If wait has a value of 1, this constructor waits to acquire the lock. If wait has a value of 0, it only tries to lock the recursive mutex.
`eh`	A reference to an error handler object.

Enhancement Orbix enhancement.

IT_RecursiveMutexLocker::~IT_RecursiveMutexLocker() Destructor

```
~IT_RecursiveMutexLocker()
{
    cancel();
}
```

The destructor releases all locks held by this recursive mutex locker.

Enhancement Orbix enhancement.

IT_RecursiveMutexLocker::lock()

```
void lock()
{
    m_recursive_mutex.lock(m_error_handler);
    m_lock_count++;
}
```

Acquires the lock.

Enhancement Orbix enhancement.

IT_RecursiveMutexLocker::lock_count()

```
unsigned int lock_count()
{
    return m_lock_count;
}
```

Returns the number of locks held by this recursive mutex locker.

Enhancement Orbix enhancement.

IT_RecursiveMutexLocker::mutex()

```
IT_RecursiveMutex& mutex()
{
    return m_recursive_mutex;
}
```

Returns direct access to the locker's recursive mutex.

Enhancement Orbix enhancement.

IT_RecursiveMutexLocker::trylock()

```
int trylock()
{
    if (m_recursive_mutex.trylock(m_error_handler) == 1)
    {
        m_lock_count++;
```

```
        return 1;
    }
    else
    {
        return 0;
    }
}
```

Tries to acquire one lock for the recursive mutex. Returns 1 if the mutex lock is successfully acquired or 0 if it is not.

Enhancement Orbix enhancement.

IT_RecursiveMutexLocker::unlock()

```
void unlock()
{
    m_recursive_mutex.unlock(m_error_handler);
    m_lock_count--;
}
```

Releases one lock held by this recursive mutex.

Enhancement Orbix enhancement.

IT_Semaphore Class

A semaphore is a non-negative counter, typically used to coordinate access to some resources.

```
class IT_Semaphore {
public:
    IT_Semaphore(
        size_t initialCount,
        IT_TSErrorHandler& eh IT_TS_DEFAULT_ERROR_HANDLER
    );

    ~IT_Semaphore();

    void post(
        IT_TSErrorHandler& eh IT_TS_DEFAULT_ERROR_HANDLER
    );

    void wait(
        IT_TSErrorHandler& eh IT_TS_DEFAULT_ERROR_HANDLER
    );

    int trywait(
        IT_TSErrorHandler& eh IT_TS_DEFAULT_ERROR_HANDLER
    );

private:
// …
};
```

IT_Semaphore::IT_Semaphore() Constructor

```
IT_Semaphore(
    size_t initialCount,
    IT_TSErrorHandler& eh IT_TS_DEFAULT_ERROR_HANDLER
);
```

A semaphore constructor that initializes the semaphore's counter with the value `initialCount`.

Parameters

`initialCount`	A positive integer value.
eh	A reference to an error handler object.

Enhancement Orbix enhancement.

Exceptions The `IT_TSRuntime` error can be reported.

IT_Semaphore::~IT_Semaphore() Destructor

```
~IT_Semaphore();
```

Destroys the semaphore.

Enhancement Orbix enhancement.

IT_Semaphore::post()

```
void post(
    IT_TSErrorHandler& eh IT_TS_DEFAULT_ERROR_HANDLER
);
```

Posts a resource thread with the semaphore. This method increments the semaphore's counter and wakes up a thread that might be blocked on `wait()`.

Parameters

eh	A reference to an error handler object.

Enhancement Orbix enhancement.

Exceptions The `IT_TSRuntime` error can be reported.

IT_Semaphore::trywait()

```
int trywait(
    IT_TSErrorHandler& eh IT_TS_DEFAULT_ERROR_HANDLER
);
```

Tries to get a resource thread. The method returns 1 if it succeeds, and 0 if it fails.

Parameters

eh	A reference to an error handler object.

Enhancement Orbix enhancement.

Exceptions An error that can be reported is `IT_TSRuntime`.

IT_Semaphore::wait()

```
void wait(
    IT_TSErrorHandler& eh IT_TS_DEFAULT_ERROR_HANDLER
);
```

Waits for one resource. The `wait()` method blocks if the semaphore's counter value is 0 and decrements the counter if the counter's value is greater than 0.

Parameters

eh	A reference to an error handler object.

Enhancement Orbix enhancement.

Exceptions Errors that can be reported include:

`IT_TSLogic`
`IT_TSRuntime`

See Also `IT_TimedSemaphore`
`IT_TimedCountByNSemaphore`

IT_TerminationHandler Class

The `IT_TerminationHandler` class enables server applications to handle delivery of CTRL_C and similar events in a portable manner. On UNIX, the termination handler handles the following signals:

```
SIGINT
SIGTERM
SIGQUIT
```

On Windows, the termination handler is a wrapper around `SetConsoleCtrlHandler`, which handles delivery of the following control events:

```
CTRL_C_EVENT
CTRL_BREAK_EVENT
CTRL_SHUTDOWN_EVENT
CTRL_LOGOFF_EVENT
CTRL_CLOSE_EVENT
```

You can create only one termination handler object in a program.

```
#include <it_ts/ts_error.h>

typedef void (*IT_TerminationHandlerFunctionPtr)(long);

class IT_IFC_API IT_TerminationHandler
{
  public:

    IT_TerminationHandler(
       IT_TerminationHandlerFunctionPtr f,
       IT_ExceptionHandler& eh = IT_EXCEPTION_HANDLER
    );

    ~IT_TerminationHandler();
};
```

IT_TerminationHandler()

```
IT_TerminationHandler(
   IT_TerminationHandlerFunctionPtr f,
   IT_ExceptionHandler& eh = IT_EXCEPTION_HANDLER
);
```

Creates a termination handler object on the stack. On POSIX platforms, it is critical to create this object in the main thread before creation of any other thread, and especially before ORB initialization.

Parameters

`f` The callback function registered by the application. The callback function takes a single `long` argument:

- On UNIX, the signal number on Unix/POSIX
- On Windows, the type of event caught

~IT_TerminationHandler()

```
~IT_TerminationHandler();
```

Deregisters the callback, in order to avoid calling it during static destruction.

IT_Thread Class

An IT_Thread object represents a thread of control. An IT_Thread object can be associated with a running thread, associated with a thread that has already terminated, or it can be null, which means it is not associated with any thread.

The important class members are as follows:

```
class IT_Thread {
public:
    IT_Thread();

    ~IT_Thread();

    IT_Thread(
        const IT_Thread& other
    );

    IT_Thread& operator=(
        const IT_Thread& other
    );

    int operator==(
        const IT_Thread& x
    ) const;

    int operator!=(
        const IT_Thread& x
    ) const
        {
            return ! operator==(x);
        }

    int is_null() const;

    static void* const thread_failed;

    void* join(
        IT_TSErrorHandler& eh IT_TS_DEFAULT_ERROR_HANDLER
```

```
    ) const;

    long id() const;
...
};
```

IT_Thread::id()

```
long id() const;
```

Returns a unique thread identifier. This method is useful for debugging.

Enhancement Orbix enhancement.

IT_Thread::is_null()

```
int is_null() const;
```

Tests if this is a null `IT_Thread` object.

Enhancement Orbix enhancement.

IT_Thread::IT_Thread() Constructors

```
IT_Thread(
    IT_Thread_i* t=0
);
```

Constructs a null `IT_Thread` object.

```
IT_Thread (
    const IT_Thread& other
);
```

Copies the `IT_Thread` object. This constructor does not start a new thread.

Parameters

`other`	The original thread to copy.

Enhancement Orbix enhancement.

IT_Thread::~IT_Thread() Destructor

```
~IT_Thread();
```

Destructor for an `IT_Thread` object.

Enhancement Orbix enhancement.

IT_Thread::join()

```
void* join(
    IT_TSErrorHandler& eh IT_TS_DEFAULT_ERROR_HANDLER
) const;
```

Waits until the thread has terminated and returns its exit status. At most one thread can successfully join a given thread, and only `Attached` threads can be joined. Note that even in the checked mode, `join()` does not always detect that you tried to join a `Detached` thread, or that you joined the same thread several times.

Parameters

eh	A reference to an error handler object.

Enhancement Orbix enhancement.

Exceptions Errors that can be reported include:

`IT_TSLogic`
`IT_TSRuntime`

See Also `IT_CurrentThread`
`IT_ThreadBody`

IT_Thread::operator=()

```
IT_Thread& operator=(
    const IT_Thread& other
);
```

Assignment operator that copies the `IT_Thread` object. This does not start a new thread.

Parameters

`other`	The original thread that is copied.

Enhancement Orbix enhancement.

IT_Thread::operator==()

```
int operator==(
    const IT_Thread& x
) const;
```

Operator that checks if two `IT_Thread` objects refer to the same thread. Returns 1 if the two objects refer to the same thread or it returns 0 if they do not refer to the same thread.

Parameters

`x`	The thread to compare to this thread.

Enhancement Orbix enhancement.

IT_Thread::operator!=()

```
int operator!=(
    const IT_Thread& x
) const
```

Operator that checks if two `IT_Thread` objects refer to different threads. Returns 1 if the two objects refer to different threads or it returns 0 if they refer to the same thread.

Parameters

`x`	The thread to compare to this thread.

Enhancement Orbix enhancement.

IT_Thread::thread_failed Constant

```
static void* const thread_failed;
```

The constant `thread_failed` is the return status of a thread to report a failure. It is neither `NULL` nor does it denote a valid address.

Enhancement Orbix enhancement.

IT_ThreadBody Class

IT_ThreadBody is the base class for thread execution methods. To start a thread, derive a class from IT_ThreadBody, add any data members needed by the thread, and provide a run() method which does the thread's work. Then use an IT_ThreadFactory object to start a thread that will execute the run() method of your IT_ThreadBody object.

If a derived IT_ThreadBody contains data, then it must not be destroyed while threads are using it. One way to manage this is to allocate the IT_ThreadBody with the new() operator and have the IT_ThreadBody delete itself at the end of run(). Also, if multiple threads run the same IT_ThreadBody, it is up to you to provide synchronization on shared data.

```
class IT_ThreadBody {
public:
    virtual ~IT_ThreadBody() {}

    virtual void* run() =0;
};
```

IT_ThreadBody::~IT_ThreadBody() Destructor

```
virtual ~IT_ThreadBody();
```

The destructor for the IT_ThreadBody object.

IT_ThreadBody::run()

```
virtual void* run() =0;
```

Does the work and returns a status, which is typically NULL or the address of a static object.

Exceptions

On platforms that support exceptions, if run() throws an exception while used by an attached thread, this thread's exit status will be IT_Thread::thread_failed.

IT_ThreadFactory Class

An `IT_ThreadFactory` object starts threads that share some common properties. You can derive your own class from `IT_ThreadFactory` to control other aspect of thread creation, such as the exact method used to create or start the thread, or the priority of threads when they are created.

```
class IT_ThreadFactory {
public:
    enum DetachState { Detached, Attached };

    IT_ThreadFactory(
        DetachState detachState,
        size_t stackSize =0
    );

    virtual ~IT_ThreadFactory();

    virtual IT_Thread start(
        IT_ThreadBody& body,
        IT_TSErrorHandler& eh IT_TS_DEFAULT_ERROR_HANDLER
    );

    static IT_Thread smf_start(
        IT_ThreadBody& body,
        DetachState detach_state,
        size_t stack_size,
        IT_TSErrorHandler& eh IT_TS_DEFAULT_ERROR_HANDLER
    );

protected:
...
```

IT_ThreadFactory::DetachState Enumeration

```
enum DetachState { Detached, Attached };
```

A thread can be started in a detached or attached state. If a thread is detached, you cannot join it (retrieve its exit status). If a thread is attached you must join it to tell the operating system to forget about it.

Enhancement Orbix enhancement.

IT_ThreadFactory::IT_ThreadFactory() Constructor

```
IT_ThreadFactory(
    DetachState detachState,
    size_t stackSize = 0
);
```

Constructor for an `IT_ThreadFactory` object.

Parameters

`detachState`	Specify whether the manufactured threads are `Detached` or `Attached`.
`stackSize`	As an option, you can specify the stack size of your threads (expressed in bytes). A value of `0` (the default) means that the operating system will use a default.

Enhancement Orbix enhancement.

See Also `IT_Thread::join()`

IT_ThreadFactory::~IT_ThreadFactory() Destructor

```
virtual ~IT_ThreadFactory();
```

The destructor for a thread factory object.

Enhancement Orbix enhancement.

IT_ThreadFactory::smf_start()

```
static IT_Thread smf_start(
    IT_ThreadBody& body,
    DetachState detach_state,
    size_t stack_size,
    IT_TSErrorHandler& eh IT_TS_DEFAULT_ERROR_HANDLER
);
```

A static member method (smf) that starts a thread without creating a thread factory explicitly. This method is useful for simple examples and prototyping but is not as flexible for robust applications.

Enhancement Orbix enhancement.

See Also IT_ThreadFactory::start()

IT_ThreadFactory::start()

```
virtual IT_Thread start(
    IT_ThreadBody& body,
    IT_TSErrorHandler& eh IT_TS_DEFAULT_ERROR_HANDLER
);
```

Starts a thread. This method creates an operating system thread that runs the given body. The method returns an IT_Thread object that represents this thread.

Parameters

body	The thread body to run.
eh	A reference to an error handler object.

Enhancement Orbix enhancement.

Exceptions An error that can be reported includes IT_TSRuntime.

See Also IT_Thread
IT_ThreadBody

IT_TimedCountByNSemaphore Class

This semaphore is a non-negative counter typically used to coordinate access to a set of resources. Several resources can be posted or waited for atomically. For example, if there are 5 resources available, a thread that asks for 7 resources would wait but another thread that later asks for 3 resources would succeed, taking 3 resources.

```
class IT_TimedCountByNSemaphore {
   public:
    enum { infinite_timeout = -1 };
    enum { infinite_size = 0 };

    IT_TimedCountByNSemaphore(
        size_t initial_count,
        size_t max_size,
        IT_TSErrorHandler& eh IT_TS_DEFAULT_ERROR_HANDLER
    );

    ~IT_TimedCountByNSemaphore();

    void post(
        size_t n,
        IT_TSErrorHandler& eh IT_TS_DEFAULT_ERROR_HANDLER
    );

    void wait(
        size_t n,
        IT_TSErrorHandler& eh IT_TS_DEFAULT_ERROR_HANDLER
    );

    int wait(
        size_t n,
        long timeout,
        IT_TSErrorHandler& eh IT_TS_DEFAULT_ERROR_HANDLER
    );

    int trywait(
        size_t n,
```

```
        IT_TSErrorHandler& eh IT_TS_DEFAULT_ERROR_HANDLER
    );

private:
...
};
```

IT_TimedCountByNSemaphore::infinite_size Constant

```
enum { infinite_size = 0 };
```

A constant used to indicate an infinite sized semaphore.

See Also IT_TimedCountByNSemaphore::wait()

IT_TimedCountByNSemaphore::infinite_timeout Constant

```
enum { infinite_timeout = -1 };
```

A constant used to indicate there is no time-out period for the semaphore.

See Also IT_TimedCountByNSemaphore::wait()

IT_TimedCountByNSemaphore:: IT_TimedCountByNSemaphore() Constructor

```
IT_TimedCountByNSemaphore(
    size_t initial_count,
    size_t max_size,
    IT_TSErrorHandler& eh IT_TS_DEFAULT_ERROR_HANDLER
);
```

Initializes the semaphore with `initial_count` and sets its maximum size to `max_size`.

Enhancement Orbix enhancement.

Exceptions An error that can be reported is IT_TSRuntime.

IT_TimedCountByNSemaphore:: ~IT_TimedCountByNSemaphore() Destructor

```
~IT_TimedCountByNSemaphore();
```

The destructor for the semaphore.

Enhancement Orbix enhancement.

IT_TimedCountByNSemaphore::post()

```
void post(
    size_t n,
    IT_TSErrorHandler& eh IT_TS_DEFAULT_ERROR_HANDLER
);
```

Posts the number of resources managed.

Parameters

n	The number of resources. If the value of `n` plus the previous number of resources is greater than `max_size`, then the number of resources remains unchanged and an `IT_TSLogic` error is reported. Calling the method using a value of 0 does nothing.
eh	A reference to an error handler object.

Enhancement Orbix enhancement.

Exceptions Errors that can be reported include:

`IT_TSRuntime`
`IT_TSLogic`

IT_TimedCountByNSemaphore::trywait()

```
int trywait(
    size_t n,
    IT_TSErrorHandler& eh IT_TS_DEFAULT_ERROR_HANDLER
);
```

Equivalent to a `wait(n, 0, eh)`.

Enhancement Orbix enhancement.

Exceptions An error that can be reported is IT_TSRuntime.

See Also IT_TimedCountByNSemaphore::wait()

IT_TimedCountByNSemaphore::wait()

```
void wait(
    size_t n,
    IT_TSErrorHandler& eh IT_TS_DEFAULT_ERROR_HANDLER
);
```

Attempts to take a set of resources atomically.

```
int wait(
    size_t n,
    long timeout,
    IT_TSErrorHandler& eh IT_TS_DEFAULT_ERROR_HANDLER
);
```

Attempts to take a set of resources (n) atomically. Returns 1 upon success or 0 when the operation times out. Calling wait(0, timeout, eh) returns 1 immediately.

Parameters

n	The number of resources attempted. A value of 0 causes the methods to return immediately.
timeout	The number of milliseconds before the call gives up. You can use the constant infinite_timeout.
eh	A reference to an error handler object.

IT_Semaphore and IT_TimedSemaphore can be more efficient than IT_TimedCountByNSemaphore when resources are posted and waited for one by one.

Enhancement Orbix enhancement.

Exceptions An error that can be reported is IT_TSRuntime.

See Also IT_Semaphore
IT_TimedSemaphore

IT_TimedOneshot Class

An `IT_TimedOneshot` class is a synchronization policy typically used to establish a rendezvous between two threads. It can have three states:

- RESET
- SIGNALED
- WAIT

The key class members are as follows:

```
class IT_TimedOneshot {
public:
    enum { infinite_timeout = -1 };

    IT_TimedOneshot(
        IT_TSErrorHandler& eh IT_TS_DEFAULT_ERROR_HANDLER
    );

    ~IT_TimedOneshot();

    void signal(
        IT_TSErrorHandler& eh IT_TS_DEFAULT_ERROR_HANDLER
    );

    void reset(
        IT_TSErrorHandler& eh IT_TS_DEFAULT_ERROR_HANDLER
    );

    void wait(
        IT_TSErrorHandler& eh IT_TS_DEFAULT_ERROR_HANDLER
    );
    int wait(
        long timeout,
        IT_TSErrorHandler& eh IT_TS_DEFAULT_ERROR_HANDLER
    );

    int trywait(
        IT_TSErrorHandler& eh IT_TS_DEFAULT_ERROR_HANDLER
    );
```

```
...
};
```

IT_TimedOneshot::infinite_timeout Constant

```
enum { infinite_timeout = -1 };
```

The `IT_TimedOneshot` class includes the symbolic constant `infinite_timeout`. This constant has a value of `-1`.

Enhancement Orbix enhancement.

See Also `IT_TimedOneshot::wait()`

IT_TimedOneshot::IT_TimedOneshot() Constructor

```
IT_TimedOneshot(
    IT_TSErrorHandler& eh IT_TS_DEFAULT_ERROR_HANDLER
);
```

Initializes the one-shot to the `RESET` state.

Parameters

eh	A reference to an error handler object.

Enhancement Orbix enhancement.

IT_TimedOneshot::~IT_TimedOneshot() Destructor

```
~IT_TimedOneshot();
```

Destroys the one-shot object.

Parameters

eh	A reference to an error handler object.

Enhancement Orbix enhancement.

IT_TimedOneshot::reset()

```
void reset(
    IT_TSErrorHandler& eh IT_TS_DEFAULT_ERROR_HANDLER
);
```

Resets the one-shot object.

- Resetting a one-shot while in the SIGNALED state changes its state to RESET.
- Resetting a one-shot while in the RESET state has no effect.
- Resetting a one-shot in the WAIT state is an error. Note that this error is not always detected, even in the checked mode.

Parameters

eh	A reference to an error handler object.

Enhancement Orbix enhancement.

IT_TimedOneshot::signal()

```
void signal(
    IT_TSErrorHandler& eh IT_TS_DEFAULT_ERROR_HANDLER
);
```

Signals the one-shot.

- Signaling a one-shot while in the RESET state changes its state to SIGNALED.
- Signaling a one-shot while in the WAIT state atomically releases the waiting thread and changes the one-shot state to RESET.
- Signaling a one-shot while in the SIGNALED state is an error.

Parameters

eh	A reference to an error handler object.

Enhancement Orbix enhancement.

IT_TimedOneshot::trywait()

```
int trywait(
    IT_TSErrorHandler& eh IT_TS_DEFAULT_ERROR_HANDLER
);
```

Equivalent to a call to `wait(0, eh)`.

Parameters

eh	A reference to an error handler object.

Enhancement Orbix enhancement.

See Also IT_TimedOneshot::wait()

IT_TimedOneshot::wait()

```
void wait(
    IT_TSErrorHandler& eh IT_TS_DEFAULT_ERROR_HANDLER
);

int wait(
    long timeout,
    IT_TSErrorHandler& eh IT_TS_DEFAULT_ERROR_HANDLER
);
```

Waits for the one-shot.

- Waiting for a one-shot while in the RESET state changes its state to WAIT. the second method returns 1 when another thread signals the one-shot within the time-out period. Otherwise it returns 0 and changes the state back to RESET.
- Waiting for a one-shot while in the SIGNALED state changes its state to RESET. The first method returns immediately and the second method returns 1 immediately.
- Waiting for a one-shot while in the WAIT state is an error.

Parameters

timeout	The number of milliseconds before the call gives up. You can use the constant infinite_timeout.
eh	A reference to an error handler object.

Enhancement Orbix enhancement.

See Also `IT_Semaphore`
`IT_TimedSemaphore`

IT_TimedSemaphore Class

The IT_TimedSemaphore object is a counter with a timer for coordinating access to some resources.

```
class IT_TS_API IT_TimedSemaphore
{
public:
    enum { infinite_timeout = -1 };

    IT_TimedSemaphore(
        size_t initial_count,
        IT_TSErrorHandler& eh IT_TS_DEFAULT_ERROR_HANDLER
    );

    ~IT_TimedSemaphore();

    void post(
        IT_TSErrorHandler& eh IT_TS_DEFAULT_ERROR_HANDLER
    );

    void wait(
        IT_TSErrorHandler& eh IT_TS_DEFAULT_ERROR_HANDLER
    );
    int wait(
        long timeout,
        IT_TSErrorHandler& eh IT_TS_DEFAULT_ERROR_HANDLER
    );

    int trywait(
        IT_TSErrorHandler& eh IT_TS_DEFAULT_ERROR_HANDLER
    );
private:
...
};
```

IT_TimedSemaphore::infinite_timeout Constant

```
enum { infinite_timeout = -1 };
```

The `IT_TimedSemaphore` class includes the symbolic constant `infinite_timeout`. This constant has a value of `-1`.

Enhancement Orbix enhancement.

See Also `IT_TimedSemaphore::wait()`

IT_TimedSemaphore::IT_TimedSemaphore() Constructor

```
IT_TimedSemaphore(
    size_t initial_count,
    IT_TSErrorHandler& eh IT_TS_DEFAULT_ERROR_HANDLER
);
```

A semaphore constructor.

Parameters

`initial_count`	Initializes the semaphore's counter with this value.
eh	A reference to an error handler object.

Enhancement Orbix enhancement.

Exceptions An error that can be reported is `IT_TSRuntime`.

IT_TimedSemaphore::~IT_TimedSemaphore() Destructor

```
~IT_TimedSemaphore();
```

The destructor.

Enhancement Orbix enhancement.

IT_TimedSemaphore::post()

```
void post(
    IT_TSErrorHandler& eh IT_TS_DEFAULT_ERROR_HANDLER
);
```

Parameters

eh	A reference to an error handler object.

Enhancement Orbix enhancement.

Exceptions An error that can be reported is IT_TSRuntime.

IT_TimedSemaphore::trywait()

```
int trywait(
    IT_TSErrorHandler& eh IT_TS_DEFAULT_ERROR_HANDLER
);
```

Returns 1 if a resource has been obtained, 0 otherwise.

Parameters

eh	A reference to an error handler object.

Enhancement Orbix enhancement.

Exceptions An error that can be reported is IT_TSRuntime.

IT_TimedSemaphore::wait()

```
void wait()
    IT_TSErrorHandler& eh IT_TS_DEFAULT_ERROR_HANDLER
);

int wait(
    long timeout,
    IT_TSErrorHandler& eh IT_TS_DEFAULT_ERROR_HANDLER
);
```

Waits for one resource. The `wait()` method blocks if the semaphore's counter value is 0 and decrements the counter if the counter's value is greater than 0.

Parameters

timeout	The number of milliseconds before the call gives up. You can also use the constant infinite_timeout.
eh	A reference to an error handler object.

Enhancement Orbix enhancement.

Exceptions Errors that can be reported include:

IT_TSRuntime
IT_TSLogic

IT_TSBadAlloc Error Class

When `new()` returns `0` an `IT_TSBadAlloc` exception is reported.

```
class IT_TS_API IT_TSBadAlloc : public IT_TSRuntime
public:
    IT_TSBadAlloc();
    virtual ~IT_TSBadAlloc();
    virtual void raise() const;
};
```

See Also

IT_TSRuntime
IT_TSError

IT_TSError Error Class

All errors reported by the TS package are `IT_TSError` objects. The key members of the class are as follows:

```
class IT_TS_API IT_TSError {
public:
    IT_TSError(
        unsigned long TS_errcode,
        long OS_errno = 0
    );
    IT_TSError(
        const IT_TSError& other
    );

    virtual ~IT_TSError();

    unsigned long TS_error_code() const;
    long OS_error_number() const;
    const char* what() const;
    virtual void raise() const;

protected:
...
```

See Also IT_DefaultTSErrorHandler

IT_TSError::IT_TSError() Constructors

```
IT_TSError(
    unsigned long TS_errcode,
    long OS_errno = 0
);

IT_TSError(
    const IT_TSError& other
);
```

Constructs an error with this TS error code and optionally an error number given by the operating system. The second method is the copy constructor.

Enhancement Orbix enhancement.

IT_TSError::~IT_TSError() Destructor

```
virtual ~IT_TSError();
```

The destructor.

Enhancement Orbix enhancement.

IT_TSError::OS_error_number()

```
long OS_error_number() const;
```

Returns the operating system error number that represent the error. Returns 0 if the error is not reported by the operating system.

Enhancement Orbix enhancement.

IT_TSError::raise()

```
virtual void raise() const;
```

When exceptions are supported, this method throws `*this`, a pointer to this `IT_TSError` object. If exceptions are not supported, it calls `::abort()`.

Enhancement Orbix enhancement.

IT_TSError::TS_error_code()

```
unsigned long TS_error_code() const;
```

Returns the TS error code that represents the error.

Enhancement Orbix enhancement.

IT_TSError::what()

```
const char* what();
```

Returns a string describing the error. The caller must not de-allocate the returned string.

Enhancement Orbix enhancement.

See Also IT_TSLogic
IT_TSRuntime
IT_TSBadAlloc

IT_TSErrorHandler Class

The last parameter of almost every TS method is a reference to an object of the class `IT_TSErrorHandler`. When a TS method detects an error, it creates an `IT_TSError` object and passes it to `IT_TSErrorHandler::handle()`.

```
class IT_TS_API IT_TSErrorHandler {
public:
    virtual ~IT_TSErrorHandler();

    virtual void handle(
        const IT_TSError& thisError
    ) = 0;
};
```

See Also `IT_DefaultTSErrorHandler`

IT_TSErrorHandler::handle()

```
virtual void handle(
   const IT_TSError& thisError
) = 0;
```

Handles the given TS error.

Parameters

`thisError`	The error raised.

Enhancement Orbix enhancement.

IT_TSErrorHandler::~IT_TSErrorHandler() Destructor

```
virtual ~IT_TSErrorHandler();
```

The destructor for the error handler object.

Enhancement Orbix enhancement.

IT_TSLogic Error Class

An `IT_TSLogic` error signals an error in the application's logic, for example when a thread attempts to join itself.

```
class IT_TS_API IT_TSLogic : public IT_TSError {
   public:
    IT_TSLogic(
        unsigned long code,
        long fromOS =0
    );

    virtual ~IT_TSLogic();

   virtual void raise() const;

private:
// ...
};
```

See Also

IT_TSError
IT_TSRuntime

IT_TSRuntime Error Class

An `IT_TSRuntime` error is an error detected by the operating system or by the underlying thread package.

```
class IT_TS_API IT_TSRuntime : public IT_TSError {
public:
    IT_TSRuntime(
        unsigned long code,
        long fromOS =0
    );

    virtual ~IT_TSRuntime();

    virtual void raise() const;

private:
...
```

See Also

IT_TSError
IT_TSRuntime

IT_TSVoidStar Class

An IT_TSVoidStar object is a data entry point that can be shared by multiple threads. Each thread can use this entry point to get and set a void* pointer that refers to thread-specific (private) data.

```
class IT_TSVoidStar {
public:
    IT_TSVoidStar(
        void (*destructor)(void*) df,
        IT_TSErrorHandler& eh IT_TS_DEFAULT_ERROR_HANDLER
    );

    ~IT_TSVoidStar();

    void* get(
        IT_TSErrorHandler& eh IT_TS_DEFAULT_ERROR_HANDLER
    ) const;

    void set(
        void* newValue,
        IT_TSErrorHandler& eh IT_TS_DEFAULT_ERROR_HANDLER
    );

private:
...
};
```

IT_TSVoidStar::IT_TSVoidStar() Constructor

```
IT_TSVoidStar(
    void (*destructor)(void*) df,
    IT_TSErrorHandler& eh IT_TS_DEFAULT_ERROR_HANDLER
);
```

Constructs an IT_TSVoidStar object. Initially, all thread-specific pointers are NULL.

Parameters

df	You can optionally associate a non-NULL destructor method with an IT_TSVoidStar object. Before exiting, a thread will call this destructor with its specific pointer value only when its specific pointer value is not NULL.
eh	A reference to an error handler object.

On some platforms, when threads are not started using an IT_ThreadBody, the application might have to call explicitly IT_CurrentThread::cleanup() upon thread exit to perform this cleanup.

Enhancement Orbix enhancement.

Exceptions An error that can be reported is IT_TSRuntime.

See Also IT_TSVoidStar::~IT_TSVoidStar()
IT_CurrentThread::cleanup()

IT_TSVoidStar::~IT_TSVoidStar() Destructor

```
~IT_TSVoidStar();
```

The destructor for an IT_TSVoidStar object.

If a non-NULL destructor method is associated with this IT_TSVoidStar object (by way of the IT_TSVoidStar() constructor), and the thread-specific value of this object is not NULL, the non-NULL destructor method is called with the thread-specific value.

WARNING: If the IT_TSVoidStar object has a non-NULL destructor, do not destroy the object while any other threads have a non-NULL thread-specific pointer. This is because on some platforms, a newly allocated IT_TSVoidStar object might *reincarnate* the destroyed IT_TSVoidStar object and its thread-specific values. This can lead to unexpected results.

Enhancement Orbix enhancement.

See Also IT_TSVoidStar::IT_TSVoidStar()

IT_TSVoidStar::get()

```
void* get(
    IT_TSErrorHandler& eh IT_TS_DEFAULT_ERROR_HANDLER
) const;
```

Gets the pointer associated with the calling thread. Returns `NULL` when the calling thread did not explicitly set this value.

Exceptions An error that can be reported is `IT_TSRuntime`.

Enhancement Orbix enhancement.

IT_TSVoidStar::set()

```
void set(
    void* newValue,
    IT_TSErrorHandler& eh IT_TS_DEFAULT_ERROR_HANDLER
);
```

Sets the pointer associated with the calling thread to `newValue`.

Exceptions An error that can be reported is `IT_TSRuntime`.

Enhancement Orbix enhancement.

Event Service

Event Service Overview

This part lists the IDL definitions in the CORBA event service modules as specified by the CORBA event service standard.

Orbix implements the IDL definitions in the `CosEventComm` and `CosEventChannelAdmin` modules to support generic event communication. These modules are listed in the files `coseventcomm.idl` and `coseventchanneladmin.idl` in the `idl` directory of your Orbix installation.

The interfaces consist of:

- "CosEventComm::PullConsumer Interface"
- "CosEventComm::PushSupplier Interface"
- "CosEventComm::PullSupplier Interface"
- "CosEventComm::PushConsumer Interface"
- "CosEventChannelAdmin::EventChannel Interface"
- "CosEventChannelAdmin::ConsumerAdmin Interface"
- "CosEventChannelAdmin::SupplierAdmin Interface"
- "CosEventChannelAdmin::ProxyPullConsumer Interface"
- "CosEventChannelAdmin::ProxyPullSupplier Interface"
- "CosEventChannelAdmin::ProxyPushConsumer Interface"
- "CosEventChannelAdmin::ProxyPushSupplier Interface"

Orbix also implements the IDL definitions in the `CosTyedEventComm` and the `CosTypedEventChannelAdmin` modules to support typed event communication. These modules are listed in the files `costypedeventcom.idl` and `costypedeventchanneladmin.idl` in the `idl` directory of your Orbix installation.

The interfaces consist of:

- "CosTypedEventComm::TypedPushConsumer Interface"
- "CosTypedEventChannelAdmin::TypedEventChannel Interface"
- "CosTypedEventChannelAdmin::TypedConsumerAdmin Interface"
- "CosTypedEventChannelAdmin::TypedSupplierAdmin Interface"
- "CosTypedEventChannelAdmin::TypedProxyPushConsumer Interface"

In addition the following IONA specific interfaces, listed in `event_channel_admin.idl`, `typed_event_channel_admin.idl`, and `messaging_admin.idl`, are provided for managing event channels:

- "IT_EventChannelAdmin::EventChannelFactory Interface"
- "IT_MessagingAdmin::Manager Interface"
- "IT_TypedEventChannelAdmin::TypedEventChannelFactory Interface"
- "OrbixEventsAdmin::ChannelManager"

These interfaces support the event communication styles as described in the *CORBA Programmer's Guide*.

OrbixEventsAdmin::ChannelManager

The previous IONA implementation of the CORBA event service, OrbixEvents, provided the event channel administration interface, `ChannelManager`, defined in the module `OrbixEventsAdmin`, to allow Orbix 3.x clients to create and manipulate multiple event channels within an OrbixEvents server.

Orbix defines the `ChannelManager` interface for backwards compatibility with OrbixEvents users. This interface is defined in the file `orbixevents.idl` in the `include/idl` directory.

WARNING: The `orbixevents.idl` file is deprecated. All new clients using the event service should be using the interfaces provided in the `IT_EventChannelAdmin` module (defined in `event_channel_admin.idl`).

Existing clients can contact the event service by calling `resolve_initial_references("EventService")` and narrowing the reference from `OrbixEventsAdmin::ChannelManager`.

```
// IDL
module OrbixEventsAdmin
{
  exception duplicateChannel{ };
  exception noSuchChannel{ };

  interface ChannelManager
  {
    typedef sequence<string> stringSeq;

    CosEventChannelAdmin::EventChannel create(in string
   channel_id)
    raises(duplicateChannel);

    CosEventChannelAdmin::EventChannel find(in string channel_id)
    raises (noSuchChannel);
```

```
    string findByRef(in CosEventChannelAdmin::EventChannel
   channel_ref)
      raises (noSuchChannel);

    stringSeq list();

    CosTypedEventChannelAdmin::TypedEventChannel createTyped(
        in string  channel_id)
    raises (duplicateChannel);

    CosTypedEventChannelAdmin::TypedEventChannel findTyped(
        in string channel_id)
    raises (noSuchChannel);

    string findByTypedRef(
      in CosTypedEventChannelAdmin::TypedEventChannel channel_ref)
    raises (noSuchChannel);

    stringSeq listTyped();
};
```

ChannelManager::create()

```
CosEventChannelAdmin::EventChannel create(in string channel_id)
raises(duplicateChannel);
```

Creates an event channel.

Parameters

`channel_id`	The channel identifier for the event channel. The exception `duplicateChannel` is raised if the channel identifier specified in `channel_id` names an existing channel. "Assigning Identifiers to Event Channels" on page 87 describes the format of channel identifiers.

ChannelManager::find()

```
CosEventChannelAdmin::EventChannel find(in string channel_id)
raises (noSuchChannel);
```

Finds the event channel associated with the channel identifier `channel_id`.

Parameters

`channel_id`	The channel identifier for the event channel. The exception `noSuchChannel` is raised if the channel identifier specified in `channel_id` does not exist. "Assigning Identifiers to Event Channels" on page 87 describes the format of channel identifiers.

ChannelManager::findByRef()

```
string findByRef(
        in CosEventChannelAdmin::EventChannel channel_ref)
raises (noSuchChannel);
```

Finds the channel identifier of the event channel specified in `channel_ref`.

Parameters

`channel_ref`	The object reference for the event channel. If `channel_ref` does not exist within the event server, `findByRef()` raises the exception `noSuchChannel`.

ChannelManager::list()

```
stringSeq list ();
```

Lists the generic event channels contained within the channel manager's event server.

ChannelManager::createTyped()

```
CosTypedEventChannelAdmin::TypedEventChannel createTyped(in string
   channel_id)
raises(duplicateChannel);
```

Creates a typed event channel.

Parameters

`channel_id`	The channel identifier for the typed event channel. The exception `duplicateChannel` is raised if the channel identifier specified in `channel_id` names an existing typed event channel.

ChannelManager::findTyped()

```
CosTypedEventChannelAdmin::TypedEventChannel findTyped(in string
   channel_id)
raises (noSuchChannel);
```

Finds the typed event channel associated with the channel identifier `channel_id`.

Parameters

`channel_id`	The channel identifier for the typed event channel. The exception `noSuchChannel` is raised if the channel identifier specified in `channel_id` does not exist.

ChannelManager::findTypedByRef()

```
string findTypedByRef(in CosTypedEventChannelAdmin::
   TypedEventChannel channel_ref)
raises (noSuchChannel);
```

Finds the channel identifier of the typed event channel specified in `channel_ref`.

Parameters

`channel_ref`	The object reference for the typed event channel. If `channel_ref` does not exist within the event server, `findByRef()` raises the exception `noSuchChannel`.

ChannelManager::listTyped()

```
stringSeq listTyped();
```

Lists the typed event channels contained within the channel manager's event server.

Unsupported Operations

The Application Server Platform event service does not support finding channels by reference. Therefore the `ChannelManager` reference will throw `NO_IMPLEMENT` for the following operations:

- `findByRef()`
- `findByTypedRef()`

CosEventChannelAdmin Module

The `CosEventChannelAdmin` module specifies the interfaces and exceptions for connecting suppliers and consumers to an event channel. It also provides the methods for managing these connections.

It contains the following interfaces:

- CosEventChannelAdmin::ProxyPushConsumer Interface
- CosEventChannelAdmin::ProxyPullSupplier Interface
- CosEventChannelAdmin::ProxyPullConsumer Interface
- CosEventChannelAdmin::ProxyPushSupplier Interface
- CosEventChannelAdmin::ConsumerAdmin Interface
- CosEventChannelAdmin::SupplierAdmin Interface
- CosEventChannelAdmin::EventChannel Interface

CosEventChannelAdmin Exceptions

exception AlreadyConnected {};

An `AlreadyConnected` exception is raised when an attempt is made to connect an object to the event channel when that object is already connected to the channel.

exception TypeError {};

The `TypeError` exception is raised when a proxy object trys to connect an object that does not support the proper typed interface.

CosEventChannelAdmin:: ProxyPushConsumer Interface

After a supplier has obtained a reference to a proxy consumer using the SupplierAdmin interface, they use the ProxyPushConsumer interface to connect to the event channel.

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

ProxyPushConsumer::connect_push_supplier()

```
//IDL
void connect_push_supplier(
       in CosEventComm::PushSupplier push_supplier)
raises (AlreadyConnected);
```

This operation connects the supplier to the event channel. If the supplier passes a nil object reference, the proxy push consumer will not notify the supplier when it is about to be disconnected.

If the proxy push consumer is already connected to the PushSupplier, then the AlreadyConnected exception is raised.

Parameters

push_supplier	The supplier that is trying to connect to the event channel

CosEventChannelAdmin:: ProxyPullSupplier Interface

After a consumer has obtained a proxy supplier using the ConsumerAdmin interface, they use the ProxyPullSupplier interface to connect to the event channel.

```
interface ProxyPullSupplier : CosEventComm::PullSupplier
{
  void connect_pull_consumer(
         in CosEventComm::PullConsumer pull_consumer)
  raises (AlreadyConnected);
};
```

ProxyPullSupplier::connect_pull_consumer()

```
//IDL
void connect_pull_consumer(
       in CosEventComm::PullConsumer pull_consumer)
raises (AlreadyConnected);
```

This operation connects the consumer to the event channel. If the consumer passes a nil object reference, the proxy pull supplier will not notify the consumer when it is about to be disconnected.

If the proxy pull supplier is already connected to the PullConsumer, then the AlreadyConnected exception is raised.

Parameters

pull_consumer	The consumer that is trying to connect to the event channel

CosEventChannelAdmin:: ProxyPullConsumer Interface

After a supplier has obtained a reference to a proxy consumer using the SupplierAdmin interface, they use the ProxyPullConsumer interface to connect to the event channel.

```
interface ProxyPullConsumer : CosEventComm::PushConsumer
{
  void connect_pull_supplier(
         in CosEventComm::PullSupplier pull_supplier)
  raises (AlreadyConnected, TypeError);
};
```

ProxyPullConsumer::connect_pull_supplier()

```
//IDL
void connect_pull_supplier(
       in CosEventComm::PullSupplier pull_supplier)
raises (AlreadyConnected, TypeError);
```

This operation connects the supplier to the event channel.

If the proxy pull consumer is already connected to a PushSupplier, then the AlreadyConnected exception is raised. The TypeError exception is raised when supplier that is being connected does not support the proper typed event structure.

Parameters

pull_supplier	The supplier that is trying to connect to the event channel.

CosEventChannelAdmin::ProxyPushSupplier Interface

After a consumer has obtained a proxy supplier using the ConsumerAdmin interface, they use the ProxyPushSupplier interface to connect to the event channel.

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

ProxyPushSupplier::connect_push_consumer()

```
//IDL
void connect_push_consumer(
       in CosEventComm::PushConsumer push_consumer )
raises (AlreadyConnected, TypeError);
```

This operation connects the consumer to the event channel.

If the proxy push supplier is already connected to the PushConsumer, then the AlreadyConnected exception is raised. The TypeError exception is when the consumer that is being connected does not support the proper typed event structure.

Parameters

push_consumer The consumer that is trying to connect to the event channel

CosEventChannelAdmin:: ConsumerAdmin Interface

Once a consumer has obtained a reference to a `ConsumerAdmin` object (by calling `EventChannel::for_consumers()`), they can use this interface to obtain a proxy supplier. This is necessary in order to connect to the event channel.

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

ConsumerAdmin::obtain_push_supplier()

```
//IDL
ProxyPushSupplier obtain_push_supplier();
```

Returns a `ProxyPushSupplier` object. The consumer can then use this object to connect to the event channel as a push-style consumer.

ConsumerAdmin::obtain_pull_supplier()

```
//IDL
ProxyPushSupplier obtain_pull_supplier();
```

Returns a `ProxyPullSupplier` object. The consumer can then use this object to connect to the event channel as a pull-style consumer.

CosEventChannelAdmin:: SupplierAdmin Interface

Once a supplier has obtained a reference to a `SupplierAdmin` object (by calling `EventChannel::for_suppliers()`), they can use this interface to obtain a proxy consumer. This is necessary in order to connect to the event channel.

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

SupplierAdmin::obtain_push_consumer()

```
//IDL
ProxyPushConsumer obtain_push_consumer();
```

Returns a `ProxyPushConsumer` object. The supplier can then use this object to connect to the event channel as a push-style supplier.

SupplierAdmin::obtain_pull_consumer()

```
//IDL
ProxyPushConsumer obtain_pull_consumer();
```

Returns a `ProxyPullConsumer` object. The supplier can then use this object to connect to the event channel as a pull-style supplier.

CosEventChannelAdmin:: EventChannel Interface

The EventChannel interface lets consumers and suppliers establish a logical connection to the event channel.

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

EventChannel::for_consumers()

```
//IDL
ConsumerAdmin for_consumers();
```

Used by a consumer to obtain an object reference that supports the `ConsumerAdmin` interface.

EventChannel::for_suppliers()

```
//IDL
SupplierAdmin for_suppliers()
```

Used by a supplier to obtain an object reference that supports the `SupplierAdmin` interface.

EventChannel::destroy()

```
//IDL
void destroy();
```

Destroys the event channel. All events that are not yet delivered, as well as all administrative objects created by the channel, are also destroyed. Connected pull consumers and push suppliers are notified when their channel is destroyed.

CosEventComm Module

The `CosEventComm` module specifies the interfaces which define the event service consumers and suppliers.

CosEventComm Exceptions

CosEventComm::Disconnected

```
exception Disconnected {};
```

`Disconnected` is raised when an attempt is made to contact a proxy that has not been connected to an event channel.

CosEventComm::PushConsumer Interface

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

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

PushConsumer::push()

```
//IDL
void push(in any data) raises(Disconnected);
```

Used by a supplier to communicate event data to the consumer. The supplier passes the event data as a parameter of type any. If the event communication has already been disconnected, the OBJECT_NOT_EXIST exception is raised.

Parameters

data	The event data, of type any.

PushConsumer::disconnect_push_consumer()

```
//IDL
void disconnect_push_consumer();
```

Lets the supplier terminate event communication. This operation releases resources used at the consumer to support the event communication. The PushConsumer object reference is discarded.

CosEventComm::PushSupplier Interface

A push-style supplier supports the `PushSupplier` interface.

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

PushSupplier::disconnect_push_supplier()

```
//IDL
void disconnect_push_supplier();
```

Lets the consumer terminate event communication. This operation releases resources used at the supplier to support the event communication. The `PushSupplier` object reference is discarded.

CosEventComm::PullSupplier Interface

A pull-style supplier supports the `PullSupplier` interface to transmit event data. A consumer requests event data from the supplier by invoking either the `pull()` operation or the `try_pull()` operation.

```
interface PullSupplier
{
    any pull() raises (Disconnected);
    any try_pull(out boolean has_event) raises (Disconnected);
    void disconnect_pull_supplier();
};
```

PullSupplier::pull()

```
//IDL
any pull() raises (Disconnected);
```

The consumer requests event data by calling this operation. The operation blocks until the event data is available, in which case it returns the event data to the consumer. Otherwise an exception is raised. If the event communication has already been disconnected, the `OBJECT_NOT_EXIST` exception is raised.

PullSupplier::try_pull()

```
//IDL
any try_pull(out boolean has_event) raises (Disconnected);
```

Unlike the `try` operation, this operation does not block. If the event data is available, it returns the event data and sets the `has_event` parameter to true. If the event is not available, it 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 `OBJECT_NOT_EXIST` exception is raised.

Parameters

has_event	Indicates whether event data is available to the `try_pull` operation

PullSupplier::disconnect_pull_supplier()

```
//IDL
void disconnect_pull_supplier();
```

Lets the consumer terminate event communication. This operation releases resources used at the supplier to support the event communication. The `PullSupplier` object reference is discarded.

CosEventComm::PullConsumer Interface

A pull-style consumer supports the `PullConsumer` interface.

```
interface PullConsumer
{
   void disconnect_pull_consumer();
};
```

PullConsumer::disconnect_pull_consumer()

```
//IDL
void disconnect_pull_consumer();
```

Lets the supplier terminate event communication. This operation releases resources used at the consumer to support the event communication. The `PullConsumer` object reference is discarded.

CosTypedEventChannelAdmin Module

The CosTypedEventChannelAdmin module defines the interfaces for making connections between suppliers and consumers that use either generic or typed communication. Its interfaces are specializations of the corresponding interfaces in the CosEventChannel module.

Note: IONA's implementation of typed events only supports the typed push style of event communication. The TypedProxyPullSupplier interface, the TypedSupplierAdmin::obtain_typed_pull_consumer() operation, and the TypedConsumerAdmin::obtain_typed_pull_supplier() operation are **not** implemented.

CosTypedEventChannelAdmin Exceptions

CosTypedEventChannelAdmin::InterfaceNotSupported

```
exception InterfaceNotSupported {};
```

InterfaceNotSupported is raised when an an attempt to obtain a TypedProxyPushConsumer fails to find an implementation that supports the strongly typed interface required by the client.

CosTypedEventChannelAdmin::NoSuchImplementation

```
exception NoSuchImplementation {};
```

NoSuchImplementation is raised when an attempt to obtain a ProxyPushSupplier fails to find an implementation that supports the strongly typed interface required by the client.

CosTypedEventChannelAdmin Data Types

CostTypedEventChannelAdmin::Key Type

```
typedef string Key;
```

A string that holds the interface repository ID of the strongly typed interface used by a typed event client.

CosTypedEventChannelAdmin::TypedProxyPushConsumer Interface

```
interface TypedProxyPushConsumer :
  CosEventChannelAdmin::ProxyPushConsumer,
  CosTypedEventComm::TypedPushConsumer
{
};
```

The TypedProxyPushConsumer interface extends the functionality of the ProxyPushConsumer to support connecting push suppliers to a typed event channel.

By inheriting from CosEventChannelAdmin::ProxyPushConsumer, this interface supports:

- connection and disconnection of push suppliers.
- generic push operation.

By inheriting from CosTypedEventComm::TypedPushConsumer, it extends the functionality of the generic ProxyPushConsumer to enable its associated supplier to use typed push communication. When a reference to a TypedProxyPushConsumer is returned by get_typed_consumer(), it has the interface identified by the Key.

Unsupported Operations

The TypedProxyPushConsumer reference will throw NO_IMPLEMENT for the push() operation. A supplier should instead call push() on the reference it obtains from the get_typed_consumer() operation.

CosTypedEventChannelAdmin::TypedSupplierAdmin Interface

```
interface TypedSupplierAdmin : CosEventChannelAdmin::SupplierAdmin
{
  TypedProxyPushConsumer obtain_typed_push_consumer(
                           in Key supported_interface)
   raises (InterfaceNotSupported);

  CosEventChannelAdmin::ProxyPullConsumer
   obtain_typed_pull_consumer(in Key uses_interface)
  raises (NoSuchImplementation);
};
```

The `TypedSupplierAdmin` interface extends the functionality of the generic `SupplierAdmin` to support connecting suppliers to a typed event channel.

TypedSupplierAdmin::obtain_typed_push_consumer()

```
TypedProxyPushConsumer obtain_typed_push_consumer(
                         in Key supported_interface)
raises (InterfaceNotSupported);
```

The `obtain_typed_push_consumer()` operation returns a `TypedProxyPushConsumer` that supports the specified interface.

Parameters

`supported_interface` Specifies the interface that the returned `TypedProxyPushConsumer` must support.

Exceptions

`InterfaceNotSupported` Raised if no consumer implementation supporting the specified interface is available.

TypedSupplierAdmin::obtain_typed_pull_consumer()

```
CosEventChannelAdmin::ProxyPullConsumer
  obtain_typed_pull_consumer(in Key uses_interface)
raises (NoSuchImplementation);
```

The `obtain_typed_pull_consumer()` operation returns a `ProxyPullConsumer` that calls operations in the interface `Pull<uses_interface>`.

Parameters

`uses_interface`	Specifies the interface which the returned `ProxyPullConsumer` must support.

Exceptions

`NoSuchImplementation`	Raised if no `ProxyPullConsumer` can be found that supports the specified interface.

Unsupported Operations

The Application Server Platform does not support the typed pull model or the connection of generic suppliers to a typed event channel. Therefore, the `TypedSupplierAdmin` reference will throw `NO_IMPLEMENT` for the following operations:

- `obtain_typed_pull_consumer()`
- `obtain_push_consumer()`
- `obtail_pull_consumer()`

CosTypedEventChannelAdmin:: TypedConsumerAdmin Interface

```
interface TypedConsumerAdmin : CosEventChannelAdmin::ConsumerAdmin
{
  TypedProxyPullSupplier obtain_typed_pull_supplier(
                           in Key supported_interface)
  raises (InterfaceNotSupported);

  CosEventChannelAdmin::ProxyPushSupplier
    obtain_typed_push_supplier(in Key uses_interface)
   raises (NoSuchImplementation);
};
```

The `TypedConsumerAdmin` interface extends the functionality of the generic `ConsumerAdmin` to support connecting consumer to a typed event channel.

TypedConsumerAdmin::obtain_typed_pull_supplier()

```
TypedProxyPullSupplier obtain_typed_pull_supplier(
                         in Key supported_interface)
raises (InterfaceNotSupported);
```

The `obtain_typed_pull_supplier()` operation returns a `TypedProxyPullSupplier` that supports the interface `Pull<supported_interface>`.

Parameters

`supported_interface` Specifies the interface which the returned `TypedProxyPullSuplier` must support.

Exceptions

`InterfaceNotSupported` Raised if `TypedProxyPullSupplier` implementation supporting the specified interface is available.

TypedConsumberAdmin::obtain_typed_push_supplier()

```
CosEventChannelAdmin::ProxyPushSupplier
  obtain_typed_push_supplier(in Key uses_interface)
raises (NoSuchImplementation);
```

The `obtain_typed_push_supplier()` operation returns a `ProxyPushSupplier` that makes calls on interface `uses_interface`.

Parameters

`uses_interface`	Specifies the interface on which the returned `ProxyPushSuppler` must make calls.

Exceptions

`NoSuchImplementation`	Raised if no `ProxyPushConsumer` can be found that supports the specified interface.

Unsupported Opperations

The Application Server Platform does not support the typed pull model or the connection of generic consumers to a typed event channel. Therefore, a TypedConsumerAdmin object will throw NO_IMPLEMENT for the following operations:

- `obtain_typed_pull_supplier()`
- `obtain_push_supplier()`
- `obtain_pull_supplier()`

CosTypedEventChannelAdmin::TypedEventChannel Interface

```
interface TypedEventChannel
{
  TypedConsumerAdmin for_consumers();

  TypedSupplierAdmin for_suppliers();

  void destroy();

};
```

This interface is the equivalent of `CosEventChannelAdmin::EventChannel` for typed events. It provides a factory for `TypedConsumerAdmin` objects and `TypedSuppleriAdmin` objects. Both of which are capable of providing proxies for typed communication.

CosTypedEventComm Module

This module specifies two interfaces used to support typed event communication. `TypedPushConsumer` supports push style typed event communication. Typed event clients retain the capability to use generic event communication.

Note: IONA's implementation of typed events only supports typed push style events. The `TypedPullSupplier` interface is **not** implemented.

CosTypedEventComm:: TypedPushConsumer Interface

```
interface TypedPushConsumer : CosEventComm::PushConsumer
{
  Object get_typed_consumer();
};
```

The TypedPushConsumer interface is used to implement push-style consumers that wish to participate in typed event communication. By inheriting from the generic PushConsumer interface, this interface retains the ability to participate in generic push-style event communication. This inheritance also requires that TypedPushConsumer objects implement the generic push() operation. However, if the consumer will be used solely for typed event communication, the push() implementation can simply raise the standard CORBA exception NO_IMPLEMENT.

TypedPushConsumer::get_typed_consumer()

```
Object get_typed_consumer();
```

get_typed_consumer() returns a reference to a typed push consumer. This reference is returned as a reference to type Object and must be narrowed to the appropriate interface. If the push supplier and the typed push consumer do not support the same interface, the narrow() will fail.

IT_EventChannelAdmin Module

Module IT_EventChannelAdmin describes extensions to the module CosEventChannelAdmin. It defines an interface, EventChannelFactory, for creating or discovering EventChannel objects.

IT_EventChannelAdmin Data Types

IT_EventChannelAdmin::ChannelID Type

```
typedef long ChannelID;
```

The ChannelID is used by the event service to track event channels. This number is assigned by the service when a new event channel is created.

IT_EventChannelAdmin::EventChannelInfo Structure

```
struct EventChannelInfo
  {
    string                                name;
    ChannelID                             id;
    CosEventChannelAdmin::EventChannel    reference;
  };
```

The EventChannelInfo is the unit of information managed by the EventChannelFactory for a given EventChannel instance. name is used for administrative purposes.

IT_EventChannelAdmin::EventChannelInfoList Sequence

```
typedef sequence<EventChannelInfo> EventChannelInfoList;
```

The EventChannelInfoList contains a sequence of EventChannelInfo and is the unit returned by EventChannelFactory::list_channels().

IT_EventChannelAdmin Exceptions

IT_EventChannelAdmin::ChannelAlreadyExists

```
exception ChannelAlreadyExists {string name;};
```

`ChannelAlreadyExists` is raised when an attempt is made to create an event channel with a name that is already in use. It returns with the name of the channel.

IT_EventChannelAdmin::ChannelNotFound

```
exception ChannelNotFound {string name;};
```

`ChannelNotFound` is raised when a call to either `EventChannelFactory::find_channel()` or `EventChannelFactory::find_channel_by_id()` cannot find the specified channel. It returns with the name of the specified channel.

IT_EventChannelAdmin:: EventChannelFactory Interface

```
interface EventChannelFactory : IT_MessagingAdmin::Manager
{
  CosEventChannelAdmin::EventChannel create_channel(
                                      in  string      name,
                                      out ChannelID   id)
   raises (ChannelAlreadyExists);

  CosEventChannelAdmin::EventChannel find_channel(
                                      in  string      name,
                                      out ChannelID   id)
   raises (ChannelNotFound);

  CosEventChannelAdmin::EventChannel find_channel_by_id(
                                      in  ChannelID   id,
                                      out string      name)
  raises (ChannelNotFound);

  EventChannelInfoList list_channels();
};
```

The `EventChannelFactory` interface defines operations for creating and managing untyped event channels. By inheriting from the `Manager` interface, it also has the ability to gracefully shut down the event service.

EventChannelFactory::create_channel()

```
//IDL
CosEventChannelAdmin::EventChannel create_channel(
                                    in string     name,
                                    out ChannelID id)
raises (ChannelAlreadyExists);
```

Creates a new instance of the event service style event channel

Parameters

`name`	The name of the channel to be created
`id`	The id of the created channel

EventChannelFactory::find_channel()

```
//IDL
CosEventChannelAdmin::EventChannel find_channel(
                                        in string     name,
                                        out ChannelID id)
raises (ChannelNotFound);
```

Returns an `EventChannel` instance specified by the provided `name`.

Parameters

`name`	The name of the channel
`id`	The channel id as returned from `create_channel()`

EventChannelFactory::find_channel_by_id()

```
//IDL
CosEventChannelAdmin::EventChannel find_channel_by_id(
                                        in ChannelID id,
                                        out string   name)
raises (ChannelNotFound);
```

Returns an `EventChannel` instance specified by the provided `id`.

Parameters

`id`	The channel id as returned from `create_channel()`
`name`	The name of the channel

EventChannelFactory::list_channels()

```
//IDL
EventChannelInfoList list_channels();
```

Return a list of the `EventChannel` instances associated with the event service.

IT_TypedEventChannelAdmin Module

Module IT_TypedEventChannelAdmin describes extensions to the module CosTypedEventChannelAdmin. It defines an interface, TypedEventChannelFactory, for creating or discovering TypedEventChannel objects.

IT_TypedEventChannelAdmin Data Types

IT_TypedEventChannelAdmin::TypedEventChannelInfo Structure

```
struct TypedEventChannelInfo
  {
    string                                        name;
    IT_EventChannelAdmin::ChannelID               id;
    string                                        interface_id;
    CosTypedEventChannelAdmin::TypedEventChannel reference;
  };
```

The TypedEventChannelInfo is the unit of information managed by the TypedEventChannelFactory for a given TypedEventChannel instance.

IT_TypedEventChannelAdmin::TypedEventChannelInfoList Sequence

```
typedef sequence<TypedEventChannelInfo> TypedEventChannelInfoList;
```

The TypedEventChannelInfoList contains a sequence of TypedEventChannelInfo and is the unit returned by TypedEventChannelFactory::list_typed_channels().

IT_TypedEventChannelAdmin:: TypedEventChannelFactory Interface

```
interface TypedEventChannelFactory : IT_MessagingAdmin::Manager
{
  CosTypedEventChannelAdmin::TypedEventChannel
  create_typed_channel(in  string                          name,
                       out IT_EventChannelAdmin::ChannelID id)
  raises (IT_EventChannelAddmin::ChannelAlreadyExists);

  CosTypedEventChannelAdmin::TypedEventChannel
  find_typed_channel(in  string                           name,
                     out IT_EventChannelAdmin::ChannelID id)
  raises (IT_EventChannelAdmin::ChannelNotFound);

  CosTypeEventChannelAdmin::TypedEventChannel
  find_typed_channel_by_id(in IT_EventChannelAdmin::ChannelID id,
                           out string      name)
  raises (IT_EventChannelAdmin::ChannelNotFound);

  TypedEventChannelInfoList list_typed_channels();
};
```

The `TypedEventChannelFactory` interface defines operations for creating and managing typed event channels. By inheriting from the `IT_MessagingAdmin::Manager` interface, it also has the ability to gracefully shut down the event service.

TypedEventChannelFactory::create_typed_channel()

```
//IDL
CosTypedEventChannelAdmin::TypedEventChannel
  create_typed_channel(in string                           name,
                       out ITEventChannelAdmin::ChannelID id)
raises (IT_EventChannelAdmin::ChannelAlreadyExists);
```

Creates a new instance of a typed event channel

Parameters

`name`	The name of the channel to be created
`id`	The id of the created channel

TypedEventChannelFactory::find_typed_channel()

```
//IDL
CosTypedEventChannelAdmin::TypedEventChannel
  find_type_channel(in string                              name,
                    out IT_EventChannelAdmin::ChannelID id)
raises (IT_EventChannelAdmin::ChannelNotFound);
```

Returns n `TypedEventChannel` instance specified by the provided `name`.

Parameters

`name`	The name of the channel
`id`	The channel id as returned from `create_typed_channel()`

TypedEventChannelFactory::find_typed_channel_by_id()

```
//IDL
CosTypedEventChannelAdmin::TypedEventChannel
  find_typed_channel_by_id(in IT_EventChannelAdmin::ChannelID id,
                           out string                          name)
raises (IT_EventChannelAdmin::ChannelNotFound);
```

Returns an `TypedEventChannel` instance specified by the provided `id`.

Parameters

`id`	The channel id as returned from `create_typed_channel()`
`name`	The name of the channel

TypedEventChannelFactory::list_typed_channels()

```
//IDL
TypedEventChannelInfoList list_typed_channels();
```

Return a list of the `TypedEventChannel` instances associated with the event service.

IT_MessagingAdmin::Manager Interface

Module `IT_MessagingAdmin` describes the administrative interface for the Event service.

The `Manager` interface provides administrative operations on an event service.

```
//IDL
    interface Manager
    {
        readonly attribute string  name;
        readonly attribute string  host;
        void shutdown();
    };
};
```

Manager::shutdown()

```
//IDL
void shutdown();
```

Shuts down an event service.

Firewall Proxy Service

IT_FPS Module

The IT_FPS module defines the constants and interface for the InterdictionPolicy.

```
const unsigned long FPS_POLICY_BASE =
   IT_PolicyBase::IONA_POLICY_ID + 40;

const CORBA::PolicyType INTERDICTION_POLICY_ID = FPS_POLICY_BASE;

enum InterdictionPolicyValue
  {
   DISABLE,
   ENABLE
   };

local interface InterdictionPolicy : CORBA::Policy
  {
  readonly attribute InterdictionPolicyValue value;
  };
```

FPS_POLICY_BASE Constant

```
const unsigned long FPS_POLICY_BASE =
   IT_PolicyBase::IONA_POLICY_ID + 40;
```

Specifies the offset used to identify the InterdictionPolicy.

INTERDICTION_POLICY_ID Constant

```
const CORBA::PolicyType INTERDICTION_POLICY_ID = FPS_POLICY_BASE;
```

Specifies the ID passed to create_policy() when creating an InterdictionPolicy.

InterdictionPolicyValue Enum

```
enum InterdictionPolicyValue
  {
   DISABLE,
   ENABLE
   };
```

Specifies the possible values for the `InterdictionPolicy`. The values are defined as follows:

`ENABLE`	This is the default behavior of the firewall proxy service plug-in. A POA with its `InterdictionPolicy` set to `ENABLE` will be proxified.
`DISABLE`	This setting tells the firewall proxy service plug-in to not proxify the POA. A POA with its `InterdictionPolicy` set to `DISABLE` will not use the firewall proxy service and requests made on objects under its control will come directly from the requesting clients.

IT_FPS::InterdictionPolicy Interface

This is an interface for a local policy object derived from `CORBA::Policy`. You create instances of `InterdictionPolicy` by calling `CORBA::ORB::create_policy()`. It is used to specify if a POA is to be proxified by the firewall proxy service.

```
local interface InterdictionPolicy : CORBA::Policy
  {
  readonly attribute InterdictionPolicyValue value;
  };
```

Naming Service

CosNaming Overview

The CosNaming module contains all IDL definitions for the CORBA naming service. The interfaces consist of:

- "CosNaming::BindingIterator Interface"
- "CosNaming::NamingContext Interface"
- "CosNaming::NamingContextExt Interface"

Use the NamingContext and BindingIterator interfaces to access standard naming service functionality. Use the NamingContextExt interface to use URLs and string representations of names.

The rest of this chapter describes data types common to the CosNaming module that are defined directly within its scope.

CosNaming::Binding Structure

```
// IDL
struct Binding {
    Name binding_name;
    BindingType binding_type;
};
```

A Binding structure represents a single binding in a naming context. A Binding structure indicates the name and type of the binding:

binding_name	The full compound name of the binding.
binding_type	The binding type, indicating whether the name is bound to an application object or a naming context.

When browsing a naming graph in the naming service, an application can list the contents of a given naming context, and determine the name and type of each binding in it. To do this, the application calls the NamingContext::list() method on the target NamingContext object. This method returns a list of Binding structures.

See Also

CosNaming::BindingList
CosNaming::BindingType

NamingContext::list()

CosNaming::BindingList Sequence

```
// IDL
typedef sequence<Binding> BindingList;
```

A sequence containing a set of Binding structures, each of which represents a single name binding.

An application can list the bindings in a given naming context using the NamingContext::list() method. An output parameter of this method returns a value of type BindingList.

See Also

CosNaming::Binding
CosNaming::BindingType
NamingContext::list()

"About Sequences"

CosNaming::BindingType Enumeration

```
// IDL
enum BindingType {nobject, ncontext};
```

The enumerated type BindingType represents these two forms of name bindings:

nobject	Describes a name bound to an application object.
ncontext	Describes a name bound to a naming context in the naming service.

There are two types of name binding in the CORBA naming service: names bound to application objects, and names bound to naming contexts. Names bound to application objects cannot be used in a compound name, except as the last element in that name. Names bound to naming contexts can be used as any component of a compound name and allow you to construct a naming graph in the naming service.

Name bindings created using NamingContext::bind() or NamingContext::rebind() are nobject bindings.

Name bindings created using the operations NamingContext::bind_context() or NamingContext::rebind_context() are ncontext bindings.

See Also

CosNaming::Binding
CosNaming::BindingList

CosNaming::Istring Data Type

```
// IDL
typedef string Istring;
```

Type Istring is a place holder for an internationalized string format.

CosNaming::Name Sequence

```
// IDL
typedef sequence<NameComponent> Name;
```

A Name represents the name of an object in the naming service. If the object name is defined within the scope of one or more naming contexts, the name is a compound name. For this reason, type Name is defined as a sequence of name components.

Two names that differ only in the contents of the kind field of one NameComponent structure are considered to be different names.

Names with no components, that is sequences of length zero, are illegal.

See Also

CosNaming::NameComponent

"About Sequences"

CosNaming::NameComponent Structure

```
// IDL
struct NameComponent {
    Istring id;
    Istring kind;
};
```

A `NameComponent` structure represents a single component of a name that is associated with an object in the naming service. The members consist of:

`id`	The identifier that corresponds to the name of the component.
`kind`	The element that adds secondary type information to the component name.

The `id` field is intended for use purely as an identifier. The semantics of the `kind` field are application-specific and the naming service makes no attempt to interpret this value.

A name component is uniquely identified by the combination of both `id` and `kind` fields. Two name components that differ only in the contents of the `kind` field are considered to be different components.

See Also `CosNaming::Name`

CosNaming::BindingIterator Interface

A `CosNaming.BindingIterator` object stores a list of name bindings and allows application to access the elements of this list.

The `NamingContext.list()` method obtains a list of bindings in a naming context. This method allows applications to specify a maximum number of bindings to be returned. To provide access to all other bindings in the naming context, the method returns an object of type `CosNaming.BindingIterator`.

```
// IDL
// In module CosNaming
interface BindingIterator {
    boolean next_one(
        out Binding b
    );
    boolean next_n(
        in unsigned long how_many,
        out BindingList bl
    );
    void destroy();
};
```

See Also `CosNaming::NamingContext::list()`

BindingIterator::destroy()

```
// IDL
void destroy();
```

Deletes the `CosNaming::BindingIterator` object on which it is called.

BindingIterator::next_n()

```
// IDL
boolean next_n(
    in unsigned long how_many,
    out BindingList bl
```

```
);
```

Gets the next how_many elements in the list of bindings, subsequent to the last element obtained by a call to next_n() or next_one(). If the number of elements in the list is less than the value of how_many, all the remaining elements are obtained.

Returns true if one or more bindings are obtained, but returns false if no more bindings remain.

Parameters

how_many	The maximum number of bindings to be obtained in parameter bl.
bl	The list of name bindings.

See Also

CosNaming::BindingIterator::next_one()
CosNaming::BindingList

BindingIterator::next_one()

```
// IDL
boolean next_one(
    out Binding b
);
```

Gets the next element in the list of bindings, subsequent to the last element obtained by a call to next_n() or next_one().

Returns true if a binding is obtained, but returns false if no more bindings remain.

Parameters

b	The name binding.

See Also

CosNaming::BindingIterator::next_n()
CosNaming::Binding

CosNaming::NamingContext Interface

The interface `CosNaming::NamingContext` provides operations to access the main features of the CORBA naming service, such as binding and resolving names. Name bindings are the associations the naming service maintains between an object reference and a useful name for that reference.

```
// IDL
// In module CosNaming
interface NamingContext {
    enum NotFoundReason {missing_node, not_context, not_object};

    exception NotFound {
        NotFoundReason why;
        Name rest_of_name;
    };
    exception CannotProceed {
        NamingContext cxt;
        Name rest_of_name;
    };
    exception InvalidName {};
    exception AlreadyBound {};
    exception NotEmpty {};

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

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

    void bind_context(
        in Name n,
```

```
            in NamingContext nc
        )
            raises (NotFound, CannotProceed, InvalidName,
    AlreadyBound);

        void rebind_context(
            in Name n,
            in NamingContext nc
        )
            raises (NotFound, CannotProceed, InvalidName );

        Object resolve(
            in Name n
        )
            raises (NotFound, CannotProceed, InvalidName );

        void unbind(
            in Name n
        )
            raises (NotFound, CannotProceed, InvalidName );

        NamingContext new_context();

        NamingContext bind_new_context(
            in Name n
        )
            raises (NotFound, CannotProceed, InvalidName,
    AlreadyBound);

        void destroy() raises (NotEmpty);

        void list(
            in unsigned long how_many,
             out BindingList bl,
            out BindingIterator bi
        );
};
```

NamingContext::AlreadyBound Exception

```
// IDL
exception AlreadyBound {};
```

If an application calls a method that attempts to bind a name to an object or naming context, but the specified name has already been bound, the method throws an exception of type `AlreadyBound`.

The following methods can throw this exception:

- `bind()`
- `bind_context()`
- `bind_new_context()`

NamingContext::bind()

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

Creates a name binding, relative to the target naming context, between a name and an object.

Parameters

`n`	The name to be bound to the target object, relative to the naming context on which the method is called.
`obj`	The application object to be associated with the specified name.

If the name passed to this method is a compound name with more than one component, all except the last component are used to find the sub-context in which to add the name binding.

Exceptions The method can throw these exceptions:

- `NotFound`
- `CannotProceed`
- `InvalidName`
- `AlreadyBound`

The contexts associated with the components must already exist, otherwise the method throws a NotFound exception.

See Also

CosNaming::NamingContext::rebind()
CosNaming::NamingContext::resolve()

NamingContext::bind_context()

```
// IDL
void bind_context(
    in Name n,
    in NamingContext nc
)
    raises (NotFound, CannotProceed, InvalidName, AlreadyBound);
```

Creates a binding, relative to the target naming context, between a name and another, specified naming context.

Parameters

n	The name to be bound to the target naming context, relative to the naming context on which the method is called. All but the final naming context specified in parameter n must already exist.
nc	The NamingContext object to be associated with the specified name. This object must already exist. To create a new NamingContext object, call NamingContext::new_context(). The entries in naming context nc can be resolved using compound names.

This new binding can be used in any subsequent name resolutions. The naming graph built using `bind_context()` is not restricted to being a tree: it can be a general naming graph in which any naming context can appear in any other naming context.

Exceptions

The method can throw these exceptions:

NotFound
CannotProceed
InvalidName
AlreadyBound

This method throws an AlreadyBound exception if the name specified by n is already in use.

See Also

CosNaming.NamingContext.bind_new_context()
CosNaming.NamingContextnew_context()
CosNamingNamingContext.rebind_context()
CosNamingNamingContextresolve()

NamingContext::bind_new_context()

```
// IDL
NamingContext bind_new_context(
    in Name n
)
    raises (NotFound, CannotProceed, InvalidName, AlreadyBound);
```

Creates a new NamingContext object in the naming service and binds the specified name to it, relative to the naming context on which the method is called. The method returns a reference to the newly created NamingContext object.

Parameters

n	The name to be bound to the newly created naming context, relative to the naming context on which the method is called. All but the final naming context specified in parameter n must already exist.

This method has the same effect as a call to NamingContext::new_context() followed by a call to NamingContext::bind_context().

The new name binding created by this method can be used in any subsequent name resolutions: the entries in the returned naming context can be resolved using compound names.

Exceptions

The method can throw these exceptions:

NotFound
CannotProceed
InvalidName
AlreadyBound

This method throws an AlreadyBound exception if the name specified by n is already in use.

See Also

CosNaming::NamingContext::bind_context()
CosNaming::NamingContext::new_context()

NamingContext::CannotProceed Exception

```
// IDL
exception CannotProceed {
    NamingContext cxt;
    Name rest_of_name;
};
```

If a naming service method fails due to an internal error, the method throws a CannotProceed exception.

A CannotProceed exception consists of two member fields:

cxt	The NamingContext object associated with the component at which the method failed.
rest_of_name	The remainder of the compound name, after the binding for the component at which the method failed.

The application might be able to use the information returned in this exception to complete the method later. For example, if you use a naming service federated across several hosts and one of these hosts is currently unavailable, a naming service method might fail until that host is available again.

The following methods can throw this exception:

- bind()
- bind_context()
- bind_new_context()
- rebind()
- rebind_context()
- resolve()
- unbind()

See Also

CosNaming::Name
CosNaming::NamingContext

NamingContext::destroy()

```
// IDL
void destroy()
    raises (NotEmpty);
```

Deletes the NamingContext object on which it is called. Before deleting a NamingContext in this way, ensure that it contains no bindings.

To avoid leaving name bindings with no associated objects in the naming service, call NamingContext.unbind() to unbind the context name before calling destroy(). See resolve() for information about the result of resolving names of context objects that no longer exist.

Exceptions

NamingContext:: NotEmpty — destroy() is called on a NamingContext that contains existing bindings.

See Also

CosNaming::NamingContext::resolve()
CosNaming::NamingContext::unbind()

NamingContext::InvalidName Exception

```
// IDL
exception InvalidName {};
```

If a method receives an in parameter of type CosNaming.Name for which the sequence length is zero, the method throws an InvalidName exception.

The following methods can throw this exception:

- bind()
- bind_context()
- bind_new_context()
- rebind()
- rebind_context()
- resolve()
- unbind()

NamingContext::list()

```
// IDL
void list(
    in unsigned long how_many,
    out BindingList bl,
    out BindingIterator bi
);
```

Gets a list of the name bindings in the naming context on which the method is called.

Parameters

how_many	The maximum number of bindings to be obtained in the BindingList parameter, bl.
bl	The list of bindings contained in the naming context on which the method is called.
bi	A BindingIterator object that provides access to all remaining bindings contained in the naming context on which the method is called. If the naming context contains more than the requested number of bindings, the BindingIterator contains the remaining bindings. If the naming context does not contain any additional bindings, the parameter bi is a nil object reference.

See Also

CosNaming::BindingIterator
CosNaming::BindingList

NamingContext::new_context()

```
// IDL
NamingContext new_context();
```

Creates a new NamingContext object in the naming service, without binding a name to it. The method returns a reference to the newly created NamingContext object.

After creating a naming context with this method, your application can bind a name to it by calling NamingContext::bind_context(). There is no relationship between this object and the NamingContext object on which the application call the method.

See Also

CosNaming::NamingContext::bind_context()
CosNaming::NamingContext::bind_new_context()

NamingContext::NotEmpty Exception

```
// IDL
exception NotEmpty {};
```

An application can call the NamingContext::destroy() method to delete a naming context object in the naming service. For this method to succeed, the naming context must contain no bindings. If bindings exist in the naming context, the method throws a NotEmpty exception.

NamingContext::NotFound Exception

```
// IDL
exception NotFound {
    NotFoundReason why;
    Name rest_of_name;
};
```

Several methods in the interface CosNaming::NamingContext require an existing name binding to be passed as an input parameter. If such an method receives a name binding that it determines is invalid, the method throws a NotFound exception. This exception contains two member fields:

why	The reason why the name binding is invalid.
rest_of_name	The remainder of the compound name following the invalid portion of the name that the method determined to be invalid.

The following methods can throw this exception:

bind()
bind_context()
bind_new_context()
rebind()
rebind_context()
resolve()
unbind()

See Also CosNaming::NamingContext::NotFoundReason

NamingContext::NotFoundReason Enumeration

```
// IDL
enum NotFoundReason {missing_node, not_context, not_object};
```

If an method throws a NotFound exception, a value of enumerated type NotFoundReason indicates the reason why the exception was thrown. The reasons consists of:

missing_node	The component of the name passed to the method did not exist in the naming service.
not_context	The method expected to receive a name that is bound to a naming context, for example using NamingContext::bind_context(), but the name received did not satisfy this requirement.
not_object	The method expected to receive a name that is bound to an application object, for example using NamingContext::bind(), but the name received did not satisfy this requirement.

See Also CosNaming::NamingContext::NotFound

NamingContext::rebind()

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

Creates a binding between an object and a name that is already bound in the target naming context. The previous name is unbound and the new binding is created in its place.

Parameters

n	The name to be bound to the specified object, relative to the naming context on which the method is called.
obj	The application object to be associated with the specified name.

As is the case with NamingContext::bind(), all but the last component of a compound name must exist, relative to the naming context on which you call the method.

Exceptions The method can throw these exceptions:

NotFound
CannotProceed
InvalidName

See Also

CosNaming::NamingContext::bind()
CosNaming::NamingContext::resolve()

NamingContext::rebind_context()

```
// IDL
void rebind_context(
    in Name n,
    in NamingContext nc
)
    raises (NotFound, CannotProceed, InvalidName);
```

The rebind_context() method creates a binding between a naming context and a name that is already bound in the context on which the method is called. The previous name is unbound and the new binding is made in its place.

Parameters

n	The name to be bound to the specified naming context, relative to the naming context on which the method is called.
nc	The naming context to be associated with the specified name.

As is the case for NamingContext::bind_context(), all but the last component of a compound name must name an existing NamingContext.

Exceptions The method can throw these exceptions:

NotFound
CannotProceed
InvalidName

See Also

CosNaming::NamingContext::bind_context()
CosNaming::NamingContext::resolve()

NamingContext::resolve()

```
// IDL
Object resolve(
    in Name n
)
    raises (NotFound, CannotProceed, InvalidName);
```

Returns the object reference that is bound to the specified name, relative to the naming context on which the method was called. The first component of the specified name is resolved in the target naming context.

Parameters

n	The name to be resolved, relative to the naming context on which the method is called.

An IDL `Object` maps to the type `CORBA::Object_ptr` in C++. You must narrow the result to the appropriate type before using it in your application.

Exceptions

The method can throw these exceptions:

NotFound
CannotProceed
InvalidName

If the name n refers to a naming context, it is possible that the corresponding `NamingContext` object no longer exists in the naming service. For example, this could happen if you call `NamingContext::destroy()` to destroy a context without first unbinding the context name. In this case, `resolve()` throws a CORBA system exception.

See Also

CosNaming::NamingContext::CannotProceed
CosNaming::NamingContext::InvalidName
CosNaming::NamingContext::NotFound

NamingContext::unbind()

```
// IDL
void unbind(
    in Name n
)
    raises (NotFound, CannotProceed, InvalidName);
```

Removes the binding between a specified name and the object associated with it.

Parameters

n	The name to be unbound in the naming service, relative to the naming context on which the method is called.

Unbinding a name does not delete the application object or naming context object associated with the name. For example, if you want to remove a naming context completely from the naming service, you should first unbind the corresponding name, then delete the NamingContext object by calling NamingContext::destroy().

Exceptions The method can throw these exceptions:

NotFound
CannotProceed
InvalidName

See Also

CosNaming::NamingContext::CannotProceed
CosNaming::NamingContext::destroy()
CosNaming::NamingContext::InvalidName
CosNaming::NamingContext::NotFound

CosNaming::NamingContextExt Interface

The NamingContextExt interface, derived from NamingContext, provides the capability for applications to use strings and Uniform Resource Locator (URL) strings to access names in the naming service.

```
// IDL
// In module CosNaming
interface NamingContextExt: NamingContext {
    typedef string StringName;
    typedef string Address;
    typedef string URLString;

    StringName to_string(
        in Name n
    )
        raises(InvalidName);

    Name to_name(
        in StringName sn
    )
        raises(InvalidName);

    exception InvalidAddress {};

    URLString to_url(
        in Address addr,
        in StringName sn
    )
        raises(InvalidAddress, InvalidName);

    Object resolve_str(
        in StringName n
    )
        raises(NotFound, CannotProceed, InvalidName,
   AlreadyBound);
};
```

NameContextExt::Address Data Type

```
// IDL
typedef string Address;
```

A URL address component is a host name optionally followed by a port number (delimited by a colon). Examples include the following:

```
my_backup_host.555xyz.com:900
myhost.xyz.com
myhost.555xyz.com
```

NameContextExt::InvalidAddress Exception

```
// IDL
exception InvalidAddress {};
```

The `to_url()` method throws an `InvalidAddress` exception when an invalid URL address component is passed to it.

See Also `CosNaming::NamingContextExt::to_url()`

NameContextExt::resolve_str()

```
// IDL
Object resolve_str(
    in StringName sn
)
    raises(NotFound, CannotProceed, InvalidName, AlreadyBound);
```

Resolves a naming service name to the object it represents in the same manner as `NamingContext::resolve()`. This method accepts a string representation of a name as an argument instead of a `Name` data type.

Parameters

sn	String representation of a name to be resolved to an object reference.

Exceptions The method can throw these exceptions:

`NotFound`
`CannotProceed`
`InvalidName`

AlreadyBound

NameContextExt::StringName Data Type

```
// IDL
typedef string StringName;
```

A string representation of an object's name in the naming service.

See Also CosNaming::Name

NameContextExt::to_name()

```
// IDL
Name to_name(
    in StringName sn
)
    raises(InvalidName);
```

Returns a naming service Name given a string representation of it.

Parameters

sn	String representation of a name in the naming service to be converted to a Name data type.

Exceptions

InvalidName	The string name is syntactically malformed or violates an implementation limit.

NameContextExt::to_string()

```
// IDL
StringName to_string(
    in Name n
)
    raises(InvalidName);
```

Returns a string representation of a naming service Name data type.

Parameters

n	The naming service Name to be converted to a string.

Exceptions

InvalidName	Name is invalid.

NameContextExt::to_url()

```
// IDL
URLString to_url(
    in Address addr,
    in StringName sn
)
    raises(InvalidAddress, InvalidName);
```

Returns a fully formed URL string, given a URL address component and a string representation of a name. It adds the necessary escape sequences to create a valid URLString.

Parameters

addr	The URL address component. An empty address means the local host.
sn	The string representation of a naming service name. An empty string is allowed.

Exceptions The method can throw these exceptions:

InvalidAddress
InvalidName

NameContextExt::URLString Data Type

```
// IDL
typedef string URLString;
```

A valid Uniform Resource Locator (URL) string. URL strings describe the location of a resource that is accessible via the Internet.

IT_Naming Module

The IT_Naming module contains a single interface, IT_NamingContextExt, which provides the method used to bind an object group into the naming service.

IT_NamingContextExt extends CosNaming::NamingContextExt and provides the method bind_object_group which binds an object group to an Iona proprietary naming service.

IT_Naming::IT_NamingContextExt Interface

The complete IT_NamingContextExt is as follows:

```
// IDL in Module IT_Naming
Interface IT_NamingContextExt : CosNaming::NamingContextExt
{
  readonly attribute IT_LoadBalancing::ObjectGroupFactory
   og_factory;
  readonly attribute IT_NamingAdmin::NamingAdmin admin;

  void bind_object_group(
    in CosNaming::Name n,
    in IT_LoadBalancing::ObjectGroup obj_gp )
  raises (CosNaming::NamingContext::NotFound,
          CosNaming::NamingContext::CannotProceed,
          CosNaming::NamingContext::InvalidName,
          CosNaming::NamingContext::AlreadyBound);
};
```

IT_Naming::IT_NamingCotextExt::bind_object_group() Method

Binds an object group to an entry in the naming service.

Parameters

n	A CosNaming::Name specifying the naming service node to bind the object group to.
obj_gp	The object group to bind into the naming service.

Enhancement Orbix enhancement to CORBA.

Exceptions

NamingContext::NotFound	n did not point to a valid entry in the naming service.
NamingContext::CannotProceed	The call failed due an internal error.
NamingContext::InvalidName	n has a sequence length of zero.
NamingContext::AlreadyBound	obj_gp is already bound into the naming service

IT_LoadBalancing Overview

The IT_LoadBalancing module provides operations that allow you to organize object references in the naming service into object groups. Object groups provide a means of controlling object load balancing by distributing work across a pool of objects.

- The ObjectGroup interface provides operations to update object group members.
- The ObjectGroupFactory interface provides operations to create or locate object groups.

The IT_LoadBalancing module also uses the following common data types and exceptions.

Table 21: *IT_LoadBalancing Common Data Types and Exceptions*

Common Data Types	Exceptions
MemberId MemberIdList SelectionMethod Member GroupId GroupList	NoSuchMember DuplicateMember DuplicateGroup NoSuchGroup

IT_LoadBalancing::MemberId Data Type

```
//IDL
typedef string MemberId;
```

An identifying string representing an object group member.

When adding a member to an object group, you must specify a string representing the object. The format of the string is left to the developer. Orbix does not interpret them. The only restriction is that member ids must be unique within each object group.

IT_LoadBalancing::MemberIdList Data Type

```
//IDL
typedef sequence<MemberId> MemberIdList;
```

A list of member ids that belong to an object group.

IT_LoadBalancing::SelectionMethod Data Type

```
//IDL
enum SelectionMethod { ROUND_ROBIN_METHOD, RANDOM_METHOD,
   ACTIVE_METHOD };
```

Specifies the algorithm for mapping a name to a member of an object group.

`ROUND_ROBIN_METHOD`	Sequentially selects objects from the object group to resolve client requests.
`RANDOM_METHOD`	Randomly selects objects from the object group to resolve client requests.
`ACTIVE_METHOD`	Uses load information supplied by the server or the system administrator to select the object with the lightest load from the object group to resolve client requests.

IT_LoadBalancing::Member Data Type

```
//IDL
struct Member
{
  Object obj;
  MemberId id;
};
```

Specifies an object group member.

IT_LoadBalancing::GroupId Data Type

```
// IDL
typedef string GroupId;
```

A string representing an object group.

When creating an object group, you must specify a string representing the object. The format of the string is left to the developer. Orbix does not interpret them. The only restriction is that group ids must be unique among object groups.

IT_LoadBalancing::GroupList Data Type

```
//IDL
typedef sequence<GroupId> GroupList;
```

A list of object group ids.

IT_LoadBalancing::NoSuchMember Exception

```
// IDL
exception NoSuchMember{};
```

Raised when the member id passed to an operation does not specify a member in the current object group.

IT_LoadBalancing::DuplicateMember Exception

```
// IDL
exception DupliccateMember{};
```

Raised by `IT_LoadBalancing::ObjectGroup::add_member` when the member id identifies a member that is already part of the group.

IT_LoadBalancing::DuplicateGroup Exception

Raised by `IT_LoadBalancing::ObjectGroupFactory::create_round_robin`, `IT_LoadBalancing::ObjectGroupFactory::create_random`, and `IT_LoadBalancing::ObjectGroupFactory::create_active` when the group id identifies a preexisting group.

IT_LoadBalancing::NoSuchGroup Exception

Raised when the specified group id does not match any registered group.

IT_LoadBalancing::ObjectGroup Interface

Object groups are controlled by the `ObjectGroup` interface, which defines the operations for manipulating the members of the object group. An `ObjectGroup` is obtained from an `ObjectGroupFactory`.

The `ObjectGroup` interface has the following attributes:

- `id` contains the group's id string specified when the group is created.
- `selection_method` specifies which algorithm is used to resolve client requests

The `ObjectGroup` interface has the following operations:

- `pick` is called by the naming service to map a client request to an active object.
- `add_member()` adds an object's reference to an object group.
- `remove_member()` removes an object's reference from the object group.
- `get_member()` returns the object by its member id.
- `members()` returns a list of all members in the object group.
- `update_member_load()` updates the object's load status.
- `get_member_load()` returns an object's load status.
- `set_member_timeout()` specifies the amount of time between load updates for a specific member. After this time the object will be removed from the group's pool of available objects.
- `get_member_timeout()` returns the member's timeout value.
- `destroy()` removes the object group from the naming service.

The complete `ObjectGroup` interface is as follows:

```
interface ObjectGroup {
 readonly attribute string id;
 attribute SelectionMethod selection_method;
 Object pick();
 void add_member (in Member mem)
 raises (DuplicateMember);
```

```
 void remove_member (in MemberId id)
 raises (NoSuchMember);
 Object get_member (in MemberId id)
 raises (NoSuchMember);
 MemberIdList members();
 void update_member_load(in MemberIdList ids, in double curr_load)
 raises (NoSuchMember);
 double get_member_load(in MemberId id)
 raises (NoSuchMember);
 void set_member_timeout(in MemberIdList ids, in long timeout_sec)
 raises (NoSuchMember);
 long get_member_timeout(in MemberId id)
 raises (NoSuchMember);
 void destroy();
};
```

ObjectGroup::pick()

```
// IDL
Object pick();
```

Returns an object from the group using the selection algorithm specified when the group was created.

See Also IT_LoadBalancing::SelectionMethod, IT_LoadBalancing::ObjectGroupFactory::create_round_robin(), IT_LoadBalancing::ObjectGroupFactory::create_random(), IT_LoadBalancing::ObjectGroupFactory::create_active()

ObjectGroup::add_member()

```
// IDL
void add_member( in Member mem )
raises (DuplicateMember);
```

Adds a reference to an object to the object group and makes it available for picking.

Parameters

`mem`	Specifies the object to be added to the object group. It is made up of a `CORBA::Object` and a `MemberId`.

Exceptions

`IT_LoadBalancing::DuplicateMember`	A member with the same `MemberId` is already associated with the object group.

ObjectGroup::remove_member()

```
// IDL
void remove_member( in MemberId id )
 raises (NoSuchMember);
```

Removes the specified object's reference from the object group. It does not effect any other references to the object stored in the naming service.

Parameters

`id`	A string that identifies the object within the object group

Exceptions

`IT_LoadBalancing::NoSuchMember`	The specified member does not exist in the object group.

ObjectGroup::get_member()

```
// IDL
Object get_member( in MemberId id )
```

Returns the object specified by `id`.

Parameters

`id`	A string that identifies the object within the object group

Exceptions

IT_LoadBalancing::NoSuchMember	The specified member does not exist in the object group.

ObjectGroup::members()

```
// IDL
MemberIdList members();
```

Returns a list containing the ids of all members in the object group.

ObjectGroup::update_member_load()

```
// IDL
 void update_member_load(in MemberIdList ids, in double curr_load)
 raises (NoSuchMember);
```

Specifies the load value used in the ACTIVE_METHOD selection algorithm.

Parameters

ids	A sequence of MemberId values that specify the objects whose load value is being updated.
curr_load	A double that specifies the load on the specified objects. The higher the value, the higher the load. Using the ACTIVE_METHOD members of the group with the lowest load values are picked first.

Exceptions

IT_LoadBalancing::NoSuchMember	One or more of the specified members do not exist in the object group.

See Also IT_LoadBalancing::SelectionMethod, IT_LoadBalancing::ObjectGroupFactory::create_active(), IT_LoadBalancing::ObjectGroup::set_member_timeout()

ObjectGroup::get_member_load()

```
// IDL
double get_member_load(in MemberId id)
raises (NoSuchMember);
```

Returns the load value for a specified object.

Parameters

`id`	A string that identifies the object within the object group

Exceptions

IT_LoadBalancing::NoSuchMember	The specified member does not exist in the object group.

See Also IT_LoadBalancing::ObjectGroup::update_member_load()

ObjectGroup::set_member_timeout()

```
void set_member_timeout(in MemberIdList ids, in long timeout_sec)
raises (NoSuchMember);
```

Specifies the amount of time, in seconds, that a member has between updates of its load value before it is removed from the list of available objects.

Parameters

`ids`	A sequence of MemberIds that specify the members whose timeout values are being set.
`timeout_sec`	A long specifying the number of seconds that an object has between load value updates. After this amount of time has expired the object will be taken off the object groups list of available objects.

Exceptions

IT_LoadBalancing::NoSuchMember	One or more of the specified members do not exist in the object group.

See Also IT_LoadBalancing::ObjectGroup::update_member_load()

ObjectGroup::get_member_timeout()

```
\\ IDL
long get_member_timeout(in MemberId id)
raises (NoSuchMember);
```

Returns the timeout value for the specified object group member.

Parameters

id	A string that identifies the object within the object group

Exceptions

IT_LoadBalancing::NoSuchMember	One or more of the specified members do not exist in the object group.

See Also IT_LoadBalancing::ObjectGroup::set_member_timeout()

ObjectGroup::destroy()

```
// IDL
void destroy()
```

Removes the object group from the naming service. Before calling destroy() on an object group, you must first unbind it.

Exceptions

CosNamimg::NamingContext::NotEmpty	The object group is not unbound from the naming service.

See Also CosNaming::NamingContext::unbind()

IT_LoadBalancing:: ObjectGroupFactory Interface

The `ObjectGroupFactory` interface provides methods for creating and locating object groups in the naming service.

The `ObjectGroupFactory` interface has the following methods to create object groups:

- `create_round_robin()` creates an object group that uses the `ROUND_ROBIN_METHOD` selction algorithm for picking objects.
- `create_random()` creates an object group that uses the `RANDOM_METHOD` selection algorithm for picking objects.
- `create_active()` creates an object group that uses the `ACTIVE_METHOD` seletion algorithm for picking objects.

The ObjectGroupFactory interface has the following methods for locating object groups in the naming service:

- `find_group` returns a specific object group.
- `rr_groups` returns a list of all object groups using the `ROUND_ROBIN_METHOD` selection algorithm.
- `random_groups` returns a list of all object groups using the `RANDOM_METHOD` selection algorithm.
- `active_groups` returns a list of all object groups using the `ACTIVE_METHOD` selection algorithm.

The complete ObjectGroupFactory interface is as follows:

```
interface ObjectGroupFactory {
  ObjectGroup create_round_robin (in GroupId id)
    raises (DuplicateGroup);
  ObjectGroup create_random (in GroupId id)
    raises (DuplicateGroup);
  ObjectGroup create_active (in GroupId id)
    raises (DuplicateGroup);
  ObjectGroup find_group (in GroupId id)
    raises (NoSuchGroup);
```

```
  GroupList rr_groups();
  GroupList random_groups();
  GroupList active_groups();
};
```

ObjectGroupFactory::create_round_robin()

```
// IDL
ObjectGroup create_round_robin (in GroupId id)
  raises (DuplicateGroup);
```

Creates an object group in the naming service. The new group uses the ROUND_ROBIN_METHOD selection algorithm for picking objects.

Parameters

id	A string identifying the object group. The string must be unique among object groups.

Exceptions

IT_LoadBalancing::DuplicateGroup	The id specified is already in use by another object group.

See Also IT_LoadBalancing::ROUND_ROBIN_METHOD

ObjectGroupFactory::create_random()

```
ObjectGroup create_random (in GroupId id)
  raises (DuplicateGroup);
```

Creates an object group in the naming service. The new group uses the RANDOM_METHOD selection algorithm for picking objects.

Parameters

id	A string identifying the object group. The string must be unique among object groups.

Exceptions

IT_LoadBalancing::DuplicateGroup	The id specified is already in use by another object group.

See Also IT_LoadBalancing::RANDOM_METHOD

ObjectGroupFactory::create_active()

```
ObjectGroup create_active (in GroupId id)
  raises (DuplicateGroup);
```

Creates an object group in the naming service. The new group uses the ACTIVE_METHOD selection algorithm for picking objects.

Parameters

id	A string identifying the object group. The string must be unique among object groups.

Exceptions

IT_LoadBalancing::DuplicateGroup	The id specified is already in use by another object group.

See Also IT_LoadBalancing::ACTIVE_METHOD

ObjectGroupFactory::find_group()

```
//IDL
ObjectGroup find_group (in GroupId id)
  raises (NoSuchGroup);
```

Returns the specified object group.

Parameters

id	A string identifying the object group. The string must be unique among object groups.

Exceptions

IT_LoadBalancing::NoSuchGroup	The group specified does not exist.

ObjectGroupFactory::rr_groups()

```
// IDL
GroupList rr_groups();
```

Returns a sequence of GroupId that identify all objects groups in the naming service that use ROUND_ROBIN_METHOD.

ObjectGroupFactory::random_groups()

```
// IDL
GroupList random_groups();
```

Returns a sequence of GroupId that identify all objects groups in the naming service that use RANDOM_METHOD.

ObjectGroupFactory::active_groups()

```
// IDL
GroupList random_groups();
```

Returns a sequence of GroupId that identify all objects groups in the naming service that use ACTIVE_METHOD.

Notification Service

Notification Service Interfaces

The Notification Service is defined in terms of the following IDL modules:

Table 22: *Notification IDL Modules*

IDL Module	Contents
`CosNotification`	Defines the structured event data type, quality of service and administrative properties, and interfaces used to administer these properties.
`CosNotifyFilter`	Defines interfaces to support filter objects. Two different filter objects are defined: **Forwarding filters:** These are instantiated with the `Filter` interface and make decisions about which events to forward, and which to discard. **Mapping filters:** These are instantiated with the `MappingFilter` interface. They determine whether a proxy object alters the properties of an event. In addition to the two types of filter object interfaces defined in this module, the `CosNotifyFilter` module also defines the `FilterFactory` interface that supports operations required to create filter objects, and the `FilterAdmin` interface that supports operations enabling a proxy or admin to manage a list of Filter instances.

Table 22: *Notification IDL Modules*

IDL Module	Contents
`CosNotifyComm`	Defines interfaces supporting Notification Service clients using structured events or sequences of structured events. This module also defines the interfaces that enable: • Event suppliers to be informed when the types of events being subscribed to change. • Event consumers to be informed whenever there is a change in the types of events being produced.
`CosNotifyChannelAdmin`	Defines interfaces to create, configure, and administer instances of a Notification Service event channel. It defines the proxy interfaces that support connections from clients, the admin interfaces, the `EventChannel` interface, and a factory interface for instantiating channels.
`IT_NotifyComm`	Extends the `CosNotifyComm` module by defining interfaces to support consumers in endpoint groups. The module also defines the interface enabling consumers in an endpoint group to be notified when there is a change in the types of events being offered.
`IT_NotifyChannelAdmin`	Extends the `CosNotifyAdmin` channel to support consumers in endpoint groups. It defines interfaces to support the connection of end-point groups to an event channel.

The following chapters describe these modules in detail.

CosNotifyChannelAdmin Module

The CosNotifyChannelAdmin module specifies the interfaces, exceptions, and data types for connecting suppliers and consumers to an event channel. It also provides the methods for managing these connections.

CosNotifyChannelAdmin Data Types

CosNotifyChannelAdmin specifies data types that facilitate the connection of clients to an event channel. The data types specify the proxy type used by a client, the type of events a client can send or recieve, and how the clients recieve subscription information. Several data types identify the client and the event channel objets responsible for managing it.

CosNotifyChannelAdmin::ProxyType Enum

```
// IDL in CosNotifyChannelAdmin
enum ProxyType
{
  PUSH_ANY,
  PULL_ANY,
  PUSH_STRUCTURED,
  PULL_STRUCTURED,
  PUSH_SEQUENCE,
  PULL_SEQUENCE,
  PUSH_TYPED,
  PULL_TYPED
}
```

Specifies the type of proxy used by a client to connect to an event channel. The type of proxy must match the type of client it connects to the channel. For example, a structured push consumer must use a PUSH_STRUCTURED proxy.

CosNotifyChannelAdmin::ObtainInfoMode Enum

```
// IDL in CosNotifyChannelAdmin Module
enum ObtainInfoMode
{
  ALL_NOW_UPDATES_ON,
  ALL_NOW_UPDATES_OFF,
  NONE_NOW_UPDATES_ON,
  NONE_NOW_UPDATES_OFF
}
```

Specifies how the client wishes to be notified of changes in subscription/publication information. The values have the following meanings:

`ALL_NOW_UPDATES_ON` Returns the current subscription/publication information and enables automatic updates.

`ALL_NOW_UPDATES_OFF` Returns the current subscription/publication information and disables automatic updates.

`NONE_NOW_UPDATES_ON` Enables automatic updates of subscription/publication information without returning the current information.

`NON_NOW_UPDATES_OFF` Disables automatic updates of subscription/publication information without returning the current information.

CosNotifyChannelAdmin::ProxyID Type

```
typedef long ProxyID;
```

Specifies the ID of a proxy in an event channel.

CosNotifyChannelAdmin::ProxyIDSeq Type

```
typedef sequence <ProxyID> ProxyIDSeq
```

Contains a list of `ProxyID` values.

CosNotifyChannelAdmin::ClientType Enum

```
// IDL in CosNotifyChannelAdmin
```

```
enum ClientType
{
  ANY_EVENT,
  STRUCTURED_EVENT,
  SEQUENCE_EVENT
}
```

Specifies the type of messages a client handles. The values have the following meanings:

ANY_EVENT	The client sends or receives messages as an Any. Consumers set with ANY_EVENT can receive structured messages, but the consumer is responsible for decoding it.
STRUCTURED_EVENT	The client sends or receives messages as a CosNotification::StructuredEvent.
SEQUENCE_EVENT	The client sends or receives messages as a CosNotification::EventBatch.

CosNotifyChannelAdmin::InterFilterGroupOperator Enum

```
// IDL in CosNotifyChannelAdmin
enum InterFilterGroupOperator
{
  AND_OP,
  OR_OP
}
```

Specifies the relationship between filters set on an admin object and the filters set on its associated filter objects. The values have the following meanings:

AND_OP	Events must pass at least one filter in both the proxy and the admin in order to be forwarded along the delivery path.
OR_OP	Events must pass at least one filter in either the proxy or the admin in order to be forwarded along the delivery path.

CosNotifyChannelAdmin::AdminID Type

```
typedef long AdminID;
```

Specifies the ID of an admin object in an event channel.

CosNotifyChannelAdmin::AdminIDSeq

```
typedef sequence <AdminID> AdminIDSeq;
```

Contains a list of IDs for admin objects in an event channel.

CosNotifyChannelAdmin::AdminLimit Type

```
//IDL in CosNotifyChannelAdmin
struct AdminLimit
{
  CosNotification::PropertyName name;
  CosNotification::PropertyValue value;
}
```

Specifies the administration property whose limit is exceeded and the value of that property. It is returned by an CosNotifyChannelAdmin:: AdminLimitExceeded exception.

Members

`name`	Name of the admin property that caused the exception.
`value`	The current value of the property.

CosNotifyChannelAdmin::ChannelID Type

```
typedef long ChannelID;
```

Specifies an event channel in the notification service.

CosNotifyChannelAdmin::ChannelIDSeq Type

```
typedef sequence <ChannelID> ChannelIDSeq;
```

Contains a list of IDs for event channels in the notification service.

CosNotifyChannelAdmin Exceptions

The `CosNotifyChannelAdmin` module defines exceptions to handle errors generated while managing client connections to an event channel.

CosNotifyChannelAdmin::ConnectionAlreadyActive Exception

```
exception ConnectionAlreadyActive{};
```

Raised when attempting to resume an already active connection between a client and an event channel.

CosNotifyChannelAdmin::ConnetionAlreadyInactive Exception

```
exception ConnectionAlreadyInactive{};
```

Raised when attempting to suspend a connection between a client and an event channel while it is suspended.

CosNotifyChannelAdmin::NotConnected Exception

```
exception NotCennected{};
```

Raised when attempting to suspend or resume a connection between a client and an event channel when the client is not connected to the channel.

CosNotifyChannelAdmin::AdminNotFound Exception

```
exception AdminNotFound{};
```

Raised when the specified Admin ID cannot be resolved.

CosNotifyChannelAdmin::ProxyNotFound Exception

```
exception ProxyNotFound{};
```

Raised when the specified proxy ID cannot be resolved.

CosNotifyChannelAdmin::AdminLimitExceeded Exception

```
exception AdminLimitExceeded{ AdminLimit admin_property_err };
```

Raised when an attempt to obtain a proxy and the new connection will put the event channel over the limit set by its `MaxConsumers` or `MaxSuppliers` setting.

The returned `AdminLimit` specifies which property caused the exception and the current setting of the property.

CosNotifyChannelAdmin::ChannelNotFound Exception

```
exception ChannelNotFound{};
```

Raised when the specified channel ID cannot be resolved.

CosNotifyChannelAdmin:: ProxyConsumer Interface

```
//IDL in CosNotifyChannelAdmin
interface ProxyConsumer:
        CosNotification::QoSAdmin,
        CosNotifyFilter::FilterAdmin
{
  readonly attribute ProxyType MyType;
  readonly attribute SupplierAdmin MyAdmin;

  CosNotification::EventTypeSeq obtain_subscription_types(
                                in ObtainInfoMode mode );

  void validate_event_qos (
         in CosNotification::QoSProperties required_qos,
         out CosNotification::NamedPropertyRangeSeq available_qos)
  raises(CosNotification::UnsupportedQoS);
};
```

The ProxyConsumer interface is an abstract interface that is inherited by the different proxy consumers that can be instantiated within an event channel. It 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 proxy consumers to administer a list of associated QoS properties. The latter inheritance enables proxy consumers to administer a list of associated filter objects. Locally, the ProxyConsumer interface defines a read-only attribute that contains a reference to the SupplierAdmin object that created it. The ProxyConsumer interface also defines an operation to return the list of event types a given proxy consumer instance can forward, and an operation to determine which QoS properties can be set on a per-event basis.

ProxyConsumer::obtain_subscription_types()

```
CosNotification::EventTypeSeq obtain_subscription_types(
                                in ObtainInfoMode mode);
```

Returns a list of event type names that consumers connected to the channel are interested in receiving.

Parameters

mode	Specifies whether to automatically notify the supplier of changes to the subsrciption list.

ProxyConsumer::validate_event_qos()

```
void validate_event_qos (
        in CosNotification::QoSProperties required_qos,
        out CosNotification::NamedPropertyRangeSeq available_qos)
    raises (CosNotification::UnsupportedQoS);
```

Checks whether the target proxy object will honor the setting of the specified QoS properties on a per-event basis. If all requested QoS property value settings can be satisfied by the target object, the operation returns successfully with an output parameter that contains a sequence of NamedPropertyRange data structures.

Parameters

required_qos	A sequence of QoS property name-value pairs that specify a set of QoS settings that a client is interested in setting on an event.

Note: 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.

available_qos	A sequence of NamedPropertyRange. Each element includes the name of a an additional QoS property whose setting is supported by the target object on a per-event basis. Each element also includes the range of values that are acceptable for each property.

Exceptions

UnsupportedQoS	Raised if any of the requested settings cannot be honored by the target object. This exception contains as data a sequence of data structures identifying the name of a QoS property in the input list whose requested setting could not be satisfied, along with an error code and a range of valid settings for the property.

CosNotifyChannelAdmin:: ProxySupplier Interface

```
//IDL
interface ProxySupplier :
        CosNotification::QoSAdmin,
        CosNotifyFilter::FilterAdmin
{
    readonly attribute ConsumerAdmin MyAdmin;
    readonly attribute ProxyType MyType;
    attribute CosNotifyFilter::MappingFilter priority_filter;
    attribute CosNotifyFilter::MappingFilter lifetime_filter;

    CosNotification::EventTypeSeq obtain_offered_types(
                                  in ObtainInfoMode mode );

    void validate_event_qos (
         in CosNotification::QoSProperties required_qos,
         out CosNotification::NamedPropertyRangeSeq available_qos)
        raises (CosNotification::UnsupportedQoS);
};
```

The `ProxySupplier` interface is an abstract interface that is inherited by the different proxy suppliers that can be instantiated within an event channel. It 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 proxy suppliers to administer a list of associated QoS properties. The latter inheritance enables proxy suppliers to administer a list of associated filter objects.

Locally, the `ProxySupplier` interface defines a read-only attribute that contains a reference to the `ConsumerAdmin` object that created it. In addition, the `ProxySupplier` interface defines attributes that associate two mapping filter objects with each proxy supplier, one for priority and one for lifetime. For more information on mapping filters refer to the *CORBA Notification Service Guide*.

Lastly, the ProxySupplier interface defines an operation to return the list of event types that a given proxy supplier can forward to its associated consumer, and an operation to determine which QoS properties can be set on a per-event basis.

ProxySupplier::priority_filter

```
attribute CosNotifyFilter::MappingFilter priority_filter;
```

Contains a reference to an object supporting the MappingFilter interface defined in the CosNotifyFilter module. Such an object encapsulates a list of constraint-value pairs, where each constraint is a boolean expression based on the type and contents of an event, and the value is a possible priority setting for the event.

Upon receipt of an event by a proxy supplier object whose priority_filter attribute contains a non-zero reference, the proxy supplier invokes the match operation supported by the mapping filter object. The mapping filter object then applies its encapsulated constraints to the event.

If the match operation returns TRUE, the proxy supplier changes the events priority to the value specified in the constraint-value pair that matched the event.

If the match operation returns FALSE, the proxy supplier checks if the events priority property is already set. If so, the filter does nothing. If the priority property is not set, the filter sets the priority property to its default value.

ProxySupplier::lifetime_filter

```
attribute CosNotifyFilter::MappingFilter lifetime_filter;
```

Contains a reference to an object supporting the MappingFilter interface defined in the CosNotifyFilter module. Such an object encapsulates a list of constraint-value pairs, where each constraint is a boolean expression based on the type and contents of an event, and the value is a possible lifetime setting for the event.

Upon receipt of each event by a proxy supplier object whose lifetime_filter attribute contains a non-zero reference, the proxy supplier invokes the match operation supported by the mapping filter object. The

mapping filter object then proceeds to apply its encapsulated constraints to the event.

If the `match` operation returns `TRUE`, the proxy supplier changes the events lifetime to the value specified in the constraint-value pair that matched the event.

If the `match` operation returns `FALSE`, the proxy supplier checks if the events lifetime property is already set. If so, the filter does nothing. If the lifetime property is not set, the filter sets the lifetime property to its default value.

ProxySupplier::obtain_offered_types()

```
CosNotification::EventTypeSeq obtain_offered_types(
                                in ObtainInfoMode mode );
```

Returns a list names of event types that the target proxy supplier can forward to its associated consumer.

This mechanism relies on event suppliers keeping the channel informed of the types of events they plan to supply by invoking the `offer_change` operation on their associated proxy consumer objects. The proxy consumers automatically share the information about supplied event types with the proxy suppliers associated with the channel. This enables consumers to discover the types of events that can be supplied to them by the channel by invoking the `obtain_offered_types` operation on their associated proxy supplier.

Parameters

`mode`	Specifies how to notify consumers of changes to the publication list.

ProxySupplier::validate_event_qos()

```
void validate_event_qos (
        in CosNotification::QoSProperties required_qos,
        out CosNotification::NamedPropertyRangeSeq available_qos)
    raises (CosNotification::UnsupportedQoS);
```

Checks whether the target proxy object will honor the setting of the specified QoS properties on a per-event basis. If all requested QoS property value settings can be satisfied by the target object, the operation returns successfully with an output parameter that contains a sequence of NamedPropertyRange data structures.

Parameters

`required_qos` A sequence of QoS property name-value pairs that specify a set of QoS settings that a client is interested in setting on an event

> **Note:** 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.

`available_qos` A sequence of NamedPropertyRange. Each element includes the name of a an additional QoS property whose setting is supported by the target object on a per-event basis. Each element also includes the range of values that are acceptable for each such property.

Exceptions

UnsupportedQoS Raised if any of the requested settings cannot be honored by the target object. 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, along with an error code and a range of settings for the property that could be satisfied.

CosNotifyChannelAdmin:: ProxyPushConsumer Interface

```
//IDL
interface ProxyPushConsumer :
        ProxyConsumer,
        CosEventComm::PushConsumer
{
    void connect_any_push_supplier (
            in CosEventComm::PushSupplier push_supplier)
        raises(CosEventChannelAdmin::AlreadyConnected);
};
```

The ProxyPushConsumer interface supports connections to the channel by suppliers that push events to the channel as Anys.

The ProxyPushConsumer extends the OMG event service push consumer interface by supporting event filtering and the configuration of various QoS properties. This interface enables OMG event service style untyped event suppliers to take advantage of these new features offered by the notification service.

Through inheritance of the ProxyConsumer interface, the ProxyPushConsumer interface supports administration of QoS properties, administration of a list of associated filter objects, and a read-only attribute containing a reference to the SupplierAdmin object that created it. In addition, this inheritance means that a ProxyPushConsumer instance supports an operation that returns the list of event types that consumers connected to the same channel are interested in receiving, and an operation that returns information about the instance's ability to accept a QoS request.

The ProxyPushConsumer interface also inherits from the PushConsumer interface defined within the CosEventComm module of the OMG event service. This interface supports the push operation which the supplier connected to a ProxyPushConsumer instance invokes to send an event to the channel in the form of an Any, and the operation to disconnect the ProxyPushConsumer from its associated supplier.

Finally, the ProxyPushConsumer interface defines the operation to establish the connection over which the push supplier sends events to the channel.

ProxyPushConsumer::connect_any_push_supplier()

```
void connect_any_push_supplier (
            in CosEventComm::PushSupplier push_supplier)
        raises(CosEventChannelAdmin::AlreadyConnected);
```

Establishes a connection between a push-style supplier of events in the form of an any and an event channel. Once established, the supplier can send events to the channel by invoking the push operation supported by the target `ProxyPushConsumer` instance.

Parameters

`push_supplier` The reference to an object supporting the `PushSupplier` interface defined within the `CosEventComm` module.

Exceptions

`AlreadyConnected` The target object of this operation is already connected to a push supplier object.

CosNotifyChannelAdmin::StructuredProxyPushConsumer Interface

```
//IDL
interface StructuredProxyPushConsumer :
        ProxyConsumer,
        CosNotifyComm::StructuredPushConsumer
{
    void connect_structured_push_supplier (
            in CosNotifyComm::StructuredPushSupplier push_supplier)
        raises(CosEventChannelAdmin::AlreadyConnected);
};
```

The StructuredProxyPushConsumer interface supports connections to the channel by suppliers that push events to the channel as structured events.

Through inheritance of the ProxyConsumer interface, the interface supports administration of QoS properties, administration of a list of associated filter objects, and a read-only attribute containing a reference to the SupplierAdmin object that created it. In addition, this inheritance means that a StructuredProxyPushConsumer instance supports an operation that returns the list of event types that consumers connected to the same channel are interested in receiving, and an operation that returns information about the instance's ability to accept a QoS request.

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 an operation to close down the connection from the supplier to the StructuredProxyPushConsumer. Since the StructuredPushConsumer interface inherits from the NotifyPublish interface, a supplier can inform the StructuredProxyPushConsumer to which it is connected whenever the list of event types it plans to supply to the channel changes.

Lastly, the StructuredProxyPushConsumer interface defines a method to establish a connection between the supplier and an event channel.

StructuredProxyPushConsumer:: connect_structured_push_supplier()

```
void connect_structured_push_supplier (
            in CosNotifyComm::StructuredPushSupplier push_supplier)
        raises(CosEventChannelAdmin::AlreadyConnected);
```

Establishes a connection between a push-style supplier of structured events and the event channel. Once the connection is established, the supplier can send events to the channel by invoking `push_structured_event` on its associated `StructuredProxyPushConsumer` instance.

Parameters

`push_supplier` A reference to an object supporting the `StructuredPushSupplier` interface defined within the `CosNotifyComm` module.

Exceptions

`AlreadyConnected` The proxy object is already connected to a push supplier object.

CosNotifyChannelAdmin::SequenceProxyPushConsumer Interface

```
//IDL
interface SequenceProxyPushConsumer :
        ProxyConsumer,
        CosNotifyComm::SequencePushConsumer
{
    void connect_sequence_push_supplier (
            in CosNotifyComm::SequencePushSupplier push_supplier)
        raises(CosEventChannelAdmin::AlreadyConnected);
};
```

The SequenceProxyPushConsumer interface supports connections to the channel by suppliers that push events to the channel as sequences of structured events.

Through inheritance of the ProxyConsumer interface, the interface supports administration of QoS properties, administration of a list of associated filter objects, and a read-only attribute containing a reference to the SupplierAdmin object that created it. In addition, this inheritance means that a SequenceProxyPushConsumer instance supports an operation that returns the list of event types that consumers connected to the same channel are interested in receiving, and an operation that returns information about the instance's ability to accept a QoS request.

The SequenceProxyPushConsumer interface also inherits from the SequencePushConsumer interface defined in the CosNotifyComm module. This interface supports the operation that enables a supplier of sequences of structured events to push them to a SequenceProxyPushConsumer, and also the operation to close down the connection from the supplier to the SequenceProxyPushConsumer. Since the SequencePushConsumer interface inherits from the NotifyPublish interface, a supplier can inform its associated SequenceProxyPushConsumer when the list of event types it supplies to the channel changes.

Lastly, the SequenceProxyPushConsumer interface defines a method to establish a connection between a supplier and an event channel.

SequenceProxyPushConsumer:: connect_sequence_push_supplier()

```
void connect_sequence_push_supplier (
            in CosNotifyComm::SequencePushSupplier push_supplier)
        raises(CosEventChannelAdmin::AlreadyConnected);
```

Establishes a connection between a push-style supplier of sequences of structured events and an event channel. Once the connection is established, the supplier can send events to the channel by invoking push_structured_events on its associated SequenceProxyPushConsumer.

Parameters

push_supplier	A reference to an object supporting the SequencePushSupplier interface defined within the CosNotifyComm module.

Exceptions

AlreadyConnected	The proxy is already connected to a push supplier object.

CosNotifyChannelAdmin:: ProxyPullSupplier Interface

```
//IDL
interface ProxyPullSupplier :
        ProxySupplier,
        CosEventComm::PullSupplier
{
    void connect_any_pull_consumer (
            in CosEventComm::PullConsumer pull_consumer)
        raises(CosEventChannelAdmin::AlreadyConnected);
};
```

The ProxyPullSupplier interface supports connections to the channel by consumers that pull events from the channel as Anys.

The ProxyPullSupplier interface extends the OMG event service pull-style consumers of untyped events by supporting event filtering and the configuration of QoS properties. This interface enables OMG event service style untyped event consumers to take advantage of the features offered by the notification service.

Through inheritance of the ProxySupplier interface, the ProxyPullSupplier interface supports administration of QoS properties, administration of a list of associated filter objects, mapping filters for event priority and lifetime, and a read-only attribute containing a reference to the ConsumerAdmin object that created it. This inheritance also means that a ProxyPullSupplier instance supports an operation that returns the list of event types that the proxy supplier will potentially supply, and an operation that returns information about the instance's ability to accept a QoS request.

The ProxyPullSupplier interface also inherits from the PullSupplier interface defined within the CosEventComm module of the OMG event service. This interface supports the pull and try_pull operations that the consumer connected to a ProxyPullSupplier instance invokes to receive an event from the channel in the form of an Any, and the operation to disconnect the ProxyPullSupplier from its associated consumer.

Finally, the ProxyPullSupplier interface defines the operation to establish a connection over which the pull consumer receives events from the channel.

ProxyPullSupplier::connect_any_pull_consumer()

```
void connect_any_pull_consumer (
            in CosEventComm::PullConsumer pull_consumer)
        raises(CosEventChannelAdmin::AlreadyConnected);
```

Establishes a connection between a pull consumer of events in the form of Anys and an event channel. Once established, the consumer can receive events from the channel by invoking `pull` or `try_pull` on its associated `ProxyPullSupplier`.

Parameters

`pull_consumer`	A reference to an object supporting the `PullConsumer` interface defined within the `CosEventComm` module of the OMG event service.

Exceptions

`AlreadyConnected`	The target object of this operation is already connected to a pull consumer object.

CosNotifyChannelAdmin::StructuredProxyPullSupplier Interface

```
//IDL
interface StructuredProxyPullSupplier :
        ProxySupplier,
        CosNotifyComm::StructuredPullSupplier
{
    void connect_structured_pull_consumer (
            in CosNotifyComm::StructuredPullConsumer pull_consumer)
        raises(CosEventChannelAdmin::AlreadyConnected);
};
```

The `StructuredProxyPullSupplier` interface supports connections to the channel by consumers that pull structured events from the channel.

Through inheritance of `ProxySupplier`, the `StructuredProxyPullSupplier` interface supports administration of QoS properties, administration of a list of associated filter objects, and a read-only attribute containing a reference to the `ConsumerAdmin` object that created it. In addition, this inheritance means that a `StructuredProxyPullSupplier` instance supports an operation that returns the list of event types that the proxy supplier can supply, and an operation that returns information about the instance's ability to accept a QoS request.

The `StructuredProxyPullSupplier` interface also inherits from the `StructuredPullSupplier` interface defined in `CosNotifyComm`. This interface supports the operations enabling a consumer of structured events to pull them from a `StructuredProxyPullSupplier`, and the operation to close the connection from the consumer to the `StructuredProxyPullSupplier`. Since the `StructuredPullSupplier` interface inherits from `NotifySubscribe`, a `StructuredProxyPullSupplier` can be notified whenever the list of event types that its associated consumer is interested in receiving changes.

Lastly, the `StructuredProxyPullSupplier` interface defines a method to establish a connection between the consumer and an event channel.

StructuredProxyPullSupplier:: connect_structured_pull_consumer()

```
void connect_structured_pull_consumer (
            in CosNotifyComm::StructuredPullSupplier pull_consumer)
        raises(CosEventChannelAdmin::AlreadyConnected);
```

Establishes a connection between a pull consumer of structured events and the event channel. Once established, the consumer can receive events from the channel by invoking `pull_structured_event` or `try_pull_structured_event` on its associated `StructuredProxyPullSupplier`.

Parameters

`pull_consumer`	A reference to an object supporting the `StructuredPullSupplier` interface defined in `CosNotifyComm`.

Exceptions

`AlreadyConnected`	The proxy is already connected to a pull consumer.

CosNotifyChannelAdmin::SequenceProxyPullSupplier Interface

```
//IDL
interface SequenceProxyPullSupplier :
        ProxySupplier,
        CosNotifyComm::SequencePullSupplier
{
        void connect_sequence_pull_consumer (
            in CosNotifyComm::SequencePullConsumer pull_consumer)
        raises(CosEventChannelAdmin::AlreadyConnected);
};
```

The SequenceProxyPullSupplier interface supports connections to the channel by consumers who pull sequences of structured events from an event channel.

Through inheritance of the ProxySupplier interface, the SequenceProxyPullSupplier interface supports administration of QoS properties, administration of a list of associated filter objects, and a read-only attribute containing a reference to the ConsumerAdmin object that created it. In addition, this inheritance implies that a SequenceProxyPullSupplier instance supports an operation that returns the list of event types that the proxy supplier can supply, and an operation that returns information about the instance's ability to accept a QoS request.

The SequenceProxyPullSupplier interface also inherits from the SequencePullSupplier interface defined in CosNotifyComm. This interface supports the operations enabling a consumer of sequences of structured events to pull them from the SequenceProxyPullSupplier, and also the operation to close the connection from the consumer to its associated SequenceProxyPullSupplier. Since the SequencePullSupplier interface inherits from the NotifySubscribe interface, a SequenceProxyPullSupplier can be notified whenever the list of event types that its associated consumer is interested in receiving changes.

The SequenceProxyPullSupplier interface also defines a method to establish a connection between the consumer and an event channel.

SequenceProxyPullSupplier:: connect_sequence_pull_consumer()

```
void connect_sequence_pull_consumer (
        in CosNotifyComm::SequencePullConsumer pull_consumer)
    raises(CosEventChannelAdmin::AlreadyConnected);
```

Establishes a connection between a pull-style consumer of sequences of structured events and the event channel. Once the connection is established, the consumer can proceed to receive events from the channel by invoking `pull_structured_events` or `try_pull_structured_events` on its associated `SequenceProxyPullSupplier`.

Parameters

`pull_consumer`	A reference to an object supporting the `SequencePullConsumer` interface defined in `CosNotifyComm`.

Exceptions

`AlreadyConnected`	The proxy is already connected to a pull consumer.

CosNotifyChannelAdmin:: ProxyPullConsumer Interface

```
//IDL
interface ProxyPullConsumer :
        ProxyConsumer,
       CosEventComm::PullConsumer
{
    void connect_any_pull_supplier (
            in CosEventComm::PullSupplier pull_supplier)
    raises(CosEventChannelAdmin::AlreadyConnected,
           CosEventChannelAdmin::TypeError);
};
```

The ProxyPullConsumer interface supports connections to the channel by suppliers who make events, packaged as Anys, available to the channel using the pull model.

The ProxyPullConsumer interface extends the OMG event service pull-style suppliers of untyped events by supporting event filtering and the configuration of QoS properties. This interface enables OMG event service style untyped event suppliers to take advantage of the features offered by the notification service.

Through inheritance of the ProxyConsumer interface, the ProxyPullConsumer interface supports administration of QoS properties, administration of a list of associated filter objects, and a read-only attribute containing a reference to the SupplierAdmin object that created it. In addition, this inheritance implies that a ProxyPullConsumer instance supports an operation that returns the list of event types that consumers connected to the same channel are interested in receiving, and an operation that returns information about the instance's ability to accept a QoS request.

The ProxyPullConsumer interface also inherits from the PullConsumer interface defined within CosEventComm. This interface supports the operation to disconnect the ProxyPullConsumer from its associated supplier. Finally, the ProxyPullConsumer interface defines the operation to establish the connection over which the pull supplier can send events to the channel.

ProxyPullConsumer::connect_any_pull_supplier()

```
void connect_any_pull_supplier (
                        in CosEventComm::PullSupplier pull_supplier)
    raises(CosEventChannelAdmin::AlreadyConnected,
           CosEventChannelAdmin::TypeError);
```

Establishes a connection between a pull-style supplier of events in the form of Anys, and the event channel. Once the connection is established, the proxy can proceed to receive events from the supplier by invoking `pull` or `try_pull` on the supplier (whether the proxy invokes `pull` or `try_pull`, and the frequency with which it performs such invocations, is a detail that is specific to the implementation of the channel).

Parameters

`pull_supplier`	A reference to an object supporting the `PullSupplier` interface defined within `CosEventComm`.

Exceptions

`AlreadyConnected`	Raised if the proxy is already connected to a pull supplier.
`TypeError`	An implementation of the `ProxyPullConsumer` interface may impose additional requirements on the interface supported by a pull supplier (for example, it may be designed to invoke some operation other than `pull` or `try_pull` in order to receive events). If the pull supplier being connected does not meet those requirements, this operation raises the `TypeError` exception.

CosNotifyChannelAdmin::StructuredProxyPullConsumer Interface

```
//IDL
interface StructuredProxyPullConsumer :
        ProxyConsumer,
        CosNotifyComm::StructuredPullConsumer
{
    void connect_structured_pull_supplier (
            in CosNotifyComm::StructuredPullSupplier pull_supplier)
        raises(CosEventChannelAdmin::AlreadyConnected,
               CosEventChannelAdmin::TypeError );
};
```

The `StructuredProxyPullConsumer` interface supports connections to the channel by suppliers that make structured events available to the channel using the pull model.

Through inheritance of `ProxyConsumer`, the `StructuredProxyPullConsumer` interface supports administration of QoS properties, administration of a list of associated filter objects, and a read-only attribute containing a reference to the `SupplierAdmin` object that created it. This inheritance also implies that a `StructuredProxyPullConsumer` instance supports an operation that returns the list of event types that consumers connected to the same channel are interested in receiving, and an operation that returns information about the instance's ability to accept a QoS request.

The `StructuredProxyPullConsumer` interface also inherits from the `StructuredPullConsumer` interface defined in `CosNotifyComm`. This interface supports the operation to close the connection from the supplier to the `StructuredProxyPullConsumer`. Since the `StructuredPullConsumer` interface inherits from `NotifyPublish`, a supplier can inform the `StructuredProxyPullConsumer` to which it is connected whenever the list of event types it plans to supply to the channel changes.

Lastly, the `StructuredProxyPullConsumer` interface defines a method to establish a connection between the supplier and an event channel.

StructuredProxyPullConsumer:: connect_structured_pull_supplier()

```
void connect_structured_pull_supplier (
        in CosNotifyComm::StructuredPullSupplier pull_supplier)
    raises(CosEventChannelAdmin::AlreadyConnected,
           CosEventChannelAdmin::TypeError );
```

Establishes a connection between a pull-style supplier of structured events and the event channel. Once the connection is established, the proxy can receive events from the supplier by invoking `pull_structured_event` or `try_pull_structured_event` on the supplier (whether the proxy invokes `pull_structured_event` or `try_pull_structured_event`, and the frequency with which it performs such invocations, is a detail specific to the implementation of the channel).

Parameters

`pull_supplier`	A reference to an object supporting the `StructuredPullSupplier` interface defined within `CosNotifyComm`.

Exceptions

`AlreadyConnected`	Raised if the proxy is already connected to a pull supplier.
`TypeError`	An implementation of the `StructuredProxyPullConsumer` interface may impose additional requirements on the interface supported by a pull supplier (for example, it may be designed to invoke some operation other than `pull_structured_event` or `try_pull_structured_event` in order to receive events). If the pull supplier being connected does not meet those requirements, this operation raises the `TypeError` exception.

CosNotifyChannelAdmin::SequenceProxyPullConsumer Interface

```
//IDL
interface SequenceProxyPullConsumer :
        ProxyConsumer,
        CosNotifyComm::SequencePullConsumer
{
    void connect_sequence_pull_supplier (
            in CosNotifyComm::SequencePullSupplier pull_supplier)
        raises(CosEventChannelAdmin::AlreadyConnected,
               CosEventChannelAdmin::TypeError );
};
```

The SequenceProxyPullConsumer interface supports connections to the channel by suppliers who make sequences of structured events available to the channel using the pull model.

Through inheritance of ProxyConsumer, the SequenceProxyPullConsumer interface supports administration of QoS properties, administration of a list of associated filter objects, and a read-only attribute containing a reference to the SupplierAdmin that created it. This inheritance also implies that a SequenceProxyPullConsumer supports an operation that returns the list of event types that consumers connected to the same channel are interested in receiving, and an operation that returns information about the instance's ability to accept a QoS request.

The SequenceProxyPullConsumer interface also inherits from the SequencePullConsumer interface defined in the CosNotifyComm module. This interface supports the operation to close the connection from the supplier to the SequenceProxyPullConsumer. Since the SequencePullConsumer interface inherits from NotifyPublish, a supplier can inform its associated SequenceProxyPullConsumer whenever the list of event types it plans to supply to the channel changes.

The SequenceProxyPullConsumer interface also defines a method to establish a connection between the supplier and an event channel.

SequenceProxyPullConsumer:: connect_sequence_pull_supplier()

```
void connect_sequence_pull_supplier (
        in CosNotifyComm::SequencePullSupplier pull_supplier)
    raises(CosEventChannelAdmin::AlreadyConnected,
           CosEventChannelAdmin::TypeError );
```

Establishes a connection between a pull-style supplier of sequences of structured events and the event channel. Once the connection is established, the proxy can receive events from the supplier by invoking `pull_structured_events` or `try_pull_structured_events` on the supplier (whether the proxy invokes `pull_structured_events` or `try_pull_structured_events`, and the frequency with which it performs such invocations, is a detail specific to the implementation of the channel).

Parameters

`pull_supplier`	A reference to an object supporting the `SequencePullSupplier` interface defined within `CosNotifyComm`.

Exceptions

`AlreadyConnected`	Raised if the proxy is already connected to a pull supplier.
`TypeError`	An implementation of the `SequenceProxyPullConsumer` interface may impose additional requirements on the interface supported by a pull supplier (for example, it may be designed to invoke some operation other than `pull_structured_events` or `try_pull_structured_events` in order to receive events). If the pull supplier being connected does not meet those requirements, this operation raises the `TypeError` exception.

CosNotifyChannelAdmin:: ProxyPushSupplier Interface

```
//IDL
interface ProxyPushSupplier :
        ProxySupplier,
        CosEventComm::PushSupplier
{
    void connect_any_push_consumer (
            in CosEventComm::PushConsumer push_consumer)
    raises(CosEventChannelAdmin::AlreadyConnected,
           CosEventChannelAdmin::TypeError );

    void suspend_connection()
    raises(CosEventChannel::ConnectionAlreadyInactive);

    void resume_connection()
    raises(CosEventChannelAdmin::ConnectionAlreadyActive);
};
```

The `ProxyPushSupplier` interface supports connections to the channel by consumers that receive events from the channel as untyped `Anys`.

The `ProxyPushSupplier` interface extends the OMG event service push-style consumers of untyped events by supporting event filtering and the configuration of QoS properties. Thus, this interface enables OMG event service push-style untyped event consumers to take advantage of the features offered by the notification service.

Through inheritance of `ProxySupplier`, the `ProxyPushSupplier` interface supports administration of QoS properties, administration of a list of associated filter objects, mapping filters for event priority and lifetime, and a read-only attribute containing a reference to the `ConsumerAdmin` that created it. This inheritance also implies that a `ProxyPushSupplier` instance supports an operation that returns the list of event types that the proxy supplier can supply, and an operation that returns information about the instance's ability to accept a QoS request.

The `ProxyPushSupplier` interface also inherits from the `PushSupplier` interface defined within `CosEventComm`. This interface supports the operation

to disconnect a ProxyPushSupplier from its associated consumer.

The ProxyPushSupplier interface defines the operation to establish the connection over which the push consumer can receive events from the channel. The ProxyPushSupplier interface also defines a pair of operations that can suspend and resume the connection between a ProxyPushSupplier and its associated PushConsumer. During the time a connection is suspended, the ProxyPushSupplier accumulates events destined for the consumer but does not transmit them until the connection is resumed.

ProxyPushSupplier::connect_any_push_consumer()

```
void connect_any_push_consumer (
        in CosEventComm::PushConsumer push_consumer)
raises(CosEventChannelAdmin::AlreadyConnected,
       CosEventChannelAdmin::TypeError );
```

Establishes a connection between a push-style consumer of events in the form of Anys, and the event channel. Once the connection is established, the ProxyPushSupplier sends events to its associated consumer by invoking push on the consumer.

Parameters

push_consumer	A reference to an object supporting the PushConsumer interface defined within CosEventComm

Exceptions

AlreadyConnected	Raised if the proxy is already connected to a push consumer.
TypeError	An implementation of the ProxyPushSupplier interface may impose additional requirements on the interface supported by a push consumer (for example, it may be designed to invoke some operation other than push in order to transmit events). If the push consumer being connected does not meet those requirements, this operation raises the TypeError exception.

ProxyPushSupplier::suspend_connection()

```
void suspend_connection()
    raises(ConnectionAlreadyInactive);
```

Causes the ProxyPushSupplier to stop sending events to the PushConsumer instance connected to it. The ProxyPushSupplier does not forward events to its associated PushConsumer until resume_connection() is invoked. During this time, the ProxyPushSupplier continues to queue events destined for the PushConsumer; however, events that time out prior to resumption of the connection are discarded. Upon resumption of the connection, all queued events are forwarded to the PushConsumer.

Exceptions

The ConnectionAlreadyInactive exception is raised if the connection is currently in a suspended state.

ProxyPushSupplier::resume_connection()

```
void resume_connection()
        raises(ConnectionAlreadyActive);
```

Causes the ProxyPushSupplier interface to resume sending events to the PushConsumer instance connected to it, including those events that have been queued while the connection was suspended and have not yet timed out.

Exceptions

ConnectionAlreadyActive The connection is not in a suspended state.

CosNotifyChannelAdmin::StructuredProxyPushSupplier Interface

```
//IDL
interface StructuredProxyPushSupplier :
        ProxySupplier,
        CosNotifyComm::StructuredPushSupplier
{

    void connect_structured_push_consumer (
           in CosNotifyComm::StructuredPushConsumer push_consumer)
        raises(CosEventChannelAdmin::AlreadyConnected,
               CosEventChannelAdmin::TypeError );

    void suspend_connection()
        raises(ConnectionAlreadyInactive);

    void resume_connection()
        raises(ConnectionAlreadyActive);
};
```

The `StructuredProxyPushSupplier` interface supports connections to the channel by consumers that receive structured events from the channel.

Through inheritance of `ProxySupplier`, the `StructuredProxyPushSupplier` interface supports administration of QoS properties, administration of a list of associated filter objects, and a read-only attribute containing a reference to the `ConsumerAdmin` that created it. This inheritance also implies that a `StructuredProxyPushSupplier` instance supports an operation that returns the list of event types that the proxy supplier can supply, and an operation that returns information about the instance's ability to accept a QoS request.

The `StructuredProxyPushSupplier` interface also inherits from the `StructuredPushSupplier` interface defined in `CosNotifyComm`. This interface supports the operation that to close the connection from the consumer to the `StructuredProxyPushSupplier`. Since `StructuredPushSupplier` inherits

from NotifySubscribe, a StructuredProxyPushSupplier can be notified whenever the list of event types that its associated consumer is interested in receiving changes.

Lastly, the StructuredProxyPushSupplier interface defines the operation to establish the connection over which the push consumer can receive events from the channel. The StructuredProxyPushSupplier interface also defines a pair of operations to suspend and resume the connection between a StructuredProxyPushSupplier and its associated StructuredPushConsumer. During the time such a connection is suspended, the StructuredProxyPushSupplier accumulates events destined for the consumer but does not transmit them until the connection is resumed.

StructuredProxyPushSupplier:: connect_structured_push_consumer()

```
void connect_structured_push_consumer (
        in CosNotifyComm::StructuredPushConsumer push_consumer)
    raises(CosEventChannelAdmin::AlreadyConnected,
           CosEventChannelAdmin::TypeError );
```

Establishes a connection between a push-style consumer of structured events and the event channel. Once the connection is established, the StructuredProxyPushSupplier sends events to the consumer by invoking push_structured_event.

Parameters

push_consumer	A reference to an object supporting the StructuredPushConsumer interface defined within CosNotifyComm

Exceptions

AlreadyConnected	Raised if the proxy is already connected to a push consumer.

TypeError	An implementation of the StructuredProxyPushSupplier interface may impose additional requirements on the interface supported by a push consumer (for example, it may be designed to invoke some operation other than push_structured_event to transmit events). If the push consumer being connected does not meet those requirements, this operation raises the TypeError exception.

StructuredProxyPushSupplier::suspend_connection()

```
void suspend_connection()
    raises(ConnectionAlreadyInactive);
```

Causes the StructuredProxyPushSupplier to stop sending events to the PushConsumer connected to it. The StructuredProxyPushSupplier does not forward events to its StructuredPushConsumer until resume_connection() is invoked. During this time, the StructuredProxyPushSupplier queues events destined for the StructuredPushConsumer; however, events that time out prior to resumption of the connection are discarded. Upon resumption of the connection, all queued events are forwarded to the StructuredPushConsumer.

Exceptions

ConnectionAlreadyInactive	The connection is already suspended.

StructuredProxyPushSupplier::resume_connection()

```
void resume_connection()
        raises(ConnectionAlreadyActive);
```

Causes causes the StructuredProxyPushSupplier to resume sending events to the StructuredPushConsumer connected to it, including those that have been queued while the connection was suspended and have not yet timed out.

Exceptions

ConnectionAlreadyActive	The connection is not currently suspended.

CosNotifyChannelAdmin:: SequenceProxyPushSupplier Interface

```
//IDL
interface SequenceProxyPushSupplier :
        ProxySupplier,
        CosNotifyComm::SequencePushSupplier
{
  void connect_sequence_push_consumer (
            in CosNotifyComm::SequencePushConsumer push_consumer)
        raises(CosEventChannelAdmin::AlreadyConnected,
               CosEventChannelAdmin::TypeError );

  void suspend_connection()
        raises(ConnectionAlreadyInactive);

  void resume_connection()
        raises(ConnectionAlreadyActive);
};
```

The SequenceProxyPushSupplier interface supports connections to the channel by consumers that receive sequences of structured events from the channel.

Through inheritance of ProxySupplier, the SequenceProxyPushSupplier interface supports administration of QoS properties, administration of a list of associated filter objects, and a read-only attribute containing a reference to the ConsumerAdmin that created it. This inheritance also implies that a SequenceProxyPushSupplier instance supports an operation that returns the list of event types that the proxy supplier can supply, and an operation that returns information about the instance's ability to accept a QoS request.

The SequenceProxyPushSupplier interface also inherits from the SequencePushSupplier interface defined in CosNotifyComm. This interface supports the operation to close the connection from the consumer to the SequenceProxyPushSupplier. Since the SequencePushSupplier interface inherits from the NotifySubscribe interface, a SequenceProxyPushSupplier can be notified whenever the list of event types that its associated consumer is interested in receiving changes.

Lastly, the SequenceProxyPushSupplier interface defines the operation to establish the connection over which the push consumer receives events from the channel. The SequenceProxyPushSupplier interface also defines a pair of operations to suspend and resume the connection between a SequenceProxyPushSupplier instance and its associated SequencePushConsumer. While a connection is suspended, the SequenceProxyPushSupplier accumulates events destined for the consumer but does not transmit them until the connection is resumed.

SequenceProxyPushSupplier:: connect_sequence_push_consumer()

```
void connect_sequence_push_consumer (
        in CosNotifyComm::SequencePushConsumer push_consumer)
    raises(CosEventChannelAdmin::AlreadyConnected,
           CosEventChannelAdmin::TypeError );
```

Establishes a connection between a push-style consumer of sequences of structured events and the event channel. Once the connection is established, the SequenceProxyPushSupplier sends events to its associated consumer by invoking push_structured_events.

Parameters

push_consumer	A reference to a SequencePushConsumer.

Exceptions

AlreadyConnected	Raised if the proxy is already connected to a push consumer.
TypeError	An implementation of the SequenceProxyPushSupplier interface may impose additional requirements on the interface supported by a push consumer (for example, it may be designed to invoke some operation other than push_structured_events in order to transmit events). If the push consumer being connected does not meet those requirements, this operation raises the TypeError exception.

SequenceProxyPushSupplier::suspend_connection()

```
void suspend_connection()
    raises(ConnectionAlreadyInactive);
```

Causes the SequenceProxyPushSupplier to stop sending events to the PushConsumer instance connected to it. The StructuredProxyPushSupplier does not forward events to its SequencePushConsumer until resume_connection() is invoked. During this time, the SequenceProxyPushSupplier continues to queue events destined for the SequencePushConsumer; however, events that time out prior to resumption of the connection are discarded. Upon resumption of the connection, all queued events are forwarded to the SequencePushConsumer.

Exceptions

ConnectionAlreadyInactive	The connection is already suspended.

SequenceProxyPushSupplier::resume_connection()

```
void resume_connection()
        raises(ConnectionAlreadyActive);
```

Causes the SequenceProxyPushSupplier to resume sending events to the SequencePushConsumer instance connected to it, including those that have been queued while the connection was suspended and have not yet timed out.

Exceptions

ConnectionAlreadyActive	The connection is not suspended.

CosNotifyChannelAdmin:: ConsumerAdmin Interface

```
//IDL
interface ConsumerAdmin :
        CosNotification::QoSAdmin,
        CosNotifyComm::NotifySubscribe,
        CosNotifyFilter::FilterAdmin,
        CosEventChannelAdmin::ConsumerAdmin
{
    readonly attribute AdminID MyID;
    readonly attribute EventChannel MyChannel;

    readonly attribute InterFilterGroupOperator MyOperator;

    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)
        raises ( AdminLimitExceeded );

    ProxySupplier obtain_notification_push_supplier (
            in ClientType ctype,
            out ProxyID proxy_id)
        raises ( AdminLimitExceeded );

    ProxySupplier obtain_txn_notification_pull_supplier (
            in ClientType ctype,
            out ProxyID proxy_id)
        raises ( AdminLimitExceeded );
```

```
    void destroy();
};
```

The ConsumerAdmin interface defines the behavior of objects that create and manage lists of proxy supplier objects within an event channel. A event channel can have any number of ConsumerAdmin instances associated with it. Each 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 group proxy suppliers 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 CosNotification, enabling each ConsumerAdmin 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.The ConsumerAdmin interface also inherits from the FilterAdmin interface defined within CosNotifyFilter. This enables each ConsumerAdmin to maintain a list of filters. These filters encapsulate subscriptions that apply to all proxy supplier objects that have been created by a given ConsumerAdmin.

The ConsumerAdmin interface also inherits from the NotifySubscribe interface defined in CosNotifyComm. This inheritance enables a ConsumerAdmin to be registered as the callback object for notification of subscription changes made on filters. This optimizes the notification of a group of proxy suppliers that have been created by the same ConsumerAdmin of changes to these shared filters.

The ConsumerAdmin interface also inherits from CosEventChannelAdmin::ConsumerAdmin. This inheritance enables clients to use the ConsumerAdmin interface to create pure OMG event service style proxy supplier objects. Proxy supplier objects created in this manner do not support configuration of QoS properties, and do not have associated filters. Proxy suppliers created through the inherited CosEventChannelAdmin::ConsumerAdmin interface do not have unique identifiers associated with them, whereas proxy suppliers created by operations supported by the ConsumerAdmin interface do have unique identifiers.

The ConsumerAdmin interface supports a read-only attribute that maintains a reference to the EventChannel instance that created it. The ConsumerAdmin interface also supports a read-only attribute that contains a unique numeric

identifier which is assigned event channel upon creation of a ConsumerAdmin instance. This identifier is unique among all ConsumerAdmin instances created by a given channel.

As described above, a ConsumerAdmin can maintain a list of filters that are applied to all proxy suppliers it creates. Each proxy supplier can also support a list of filters that apply only to the proxy. When combining these two lists during the evaluation of a given event, either AND or OR semantics may be applied. The choice is determined by an input flag when creating of the ConsumerAdmin, and the operator that is used for this purpose by a given ConsumerAdmin is maintained in a read-only attribute.

The ConsumerAdmin interface also supports attributes that maintain references to priority and lifetime mapping filter objects. These mapping filter objects are applied to all proxy supplier objects created by a given ConsumerAdmin.

Each ConsumerAdmin assigns a unique numeric identifier to each proxy supplier 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. The ConsumerAdmin interface also supports an operation that, given the unique identifier of a proxy supplier, returns the object reference of that proxy supplier. Finally, the ConsumerAdmin interface supports operations that create the various styles of proxy supplier objects supported by the event channel.

ConsumerAdmin::MyID

```
readonly attribute AdminID MyID;
```

Maintains the unique identifier of the target ConsumerAdmin instance that is assigned to it upon creation by the event channel.

ConsumerAdmin::MyChannel

```
readonly attribute EventChannel MyChannel
```

Maintains the object reference of the event channel that created a given ConsumerAdmin instance.

ConsumerAdmin::MyOperator

```
readonly attribute InterFilterGroupOperator MyOperator;
```

Maintains the information regarding whether AND or OR semantics are used during the evaluation of a given event when combining the filter objects associated with the target ConsumerAdmin and those defined locally on a given proxy supplier.

ConsumerAdmin::priority_filter

```
attribute CosNotifyFilter::MappingFilter priority_filter;
```

Maintains a reference to a mapping filter object that affects how each proxy supplier created by the target ConsumerAdmin treats events with respect to priority.

Each proxy supplier also has an associated attribute which maintains a reference to a mapping filter object for the priority property. This local mapping filter object is only used by the proxy supplier in the event that the priority_filter attribute of the ConsumerAdmin instance that created it is set to OBJECT_NIL.

ConsumerAdmin::lifetime_filter

```
attribute CosNotifyFilter::MappingFilter lifetime_filter;
```

Maintains a reference to a mapping filter that affects how each proxy supplier created by the target ConsumerAdmin treats events with respect to lifetime.

Each proxy supplier object also has an associated attribute that maintains a reference to a mapping filter object for the lifetime property. This local mapping filter object is only used by the proxy supplier in the event that the lifetime_filter attribute of the ConsumerAdmin instance that created it is set to OBJECT_NIL.

ConsumerAdmin::pull_suppliers

```
readonly attribute ProxyIDSeq pull_suppliers;
```

Contains the list of unique identifiers that have been assigned by a ConsumerAdmin instance to each pull-style proxy supplier it has created.

ConsumerAdmin::push_suppliers

```
readonly attribute ProxyIDSeq push_suppliers;
```

Contains the list of unique identifiers that have been assigned by a ConsumerAdmin instance to each push-style proxy supplier it has created.

ConsumerAdmin::get_proxy_supplier()

```
ProxySupplier get_proxy_supplier (in ProxyID proxy_id)
    raises ( ProxyNotFound );
```

Returns an object reference to the proxy supplier whose unique id was passed to the method.

Parameters

proxy_id	A numeric identifier associated with one of the proxy suppliers that created by the target ConsumerAdmin.

Exceptions

ProxyNotFound	The input parameter does not correspond to the unique identifier of a proxy supplier object created by the target ConsumerAdmin.

ConsumerAdmin::obtain_notification_pull_supplier()

```
ProxySupplier obtain_notification_pull_supplier (
        in ClientType ctype,
        out ProxyID proxy_id)
    raises ( AdminLimitExceeded );
```

Creates instances of the pull-style proxy suppliers defined in CosNotifyChannelAdmin and returns an object reference to the new proxy.

Three varieties of pull-style proxy suppliers are defined in this module:

- The ProxyPullSupplier interface supports connections to pull consumers that receive events as Anys.
- The StructuredProxyPullSupplier interface supports connections to pull consumers that receive structured events.
- The SequenceProxyPullSupplier interface support connections to pull consumers that receive sequences of structured events.

The input parameter flag indicates which type of pull style proxy instance to create.

The target ConsumerAdmin creates the new pull-style proxy supplier and assigns a numeric identifier to it that is unique among all proxy suppliers the ConsumerAdmin has created.

Parameters

ctype	A flag that indicates which style of pull-style proxy supplier to create.
proxy_id	The unique identifier of the new proxy supplier.

Exceptions

AdminLimitExceeded	The number of consumers currently connected to the channel with which the target ConsumerAdmin is associated exceeds the value of the MaxConsumers administrative property.

ConsumerAdmin::obtain_notification_push_supplier()

```
ProxySupplier obtain_notification_push_supplier (
        in ClientType ctype,
        out ProxyID proxy_id)
    raises ( AdminLimitExceeded );
```

Creates instances of the push-style proxy supplier objects defined in CosNotifyChannelAdmin and returns an object reference to the new proxy.

Three varieties of push-style proxy suppliers are defined in this module:

- The ProxyPushSupplier interface supports connections to push consumers that receive events as Anys.

- The StructuredProxyPushSupplier interface supports connections to push consumers that receive structured events.
- The SequenceProxyPushSupplier interface supports connections to push consumers that receive sequences of structured events.

The input parameter flag indicates which type of push-style proxy to create.

The target ConsumerAdmin creates the new push-style proxy supplier and assigns a numeric identifier to it that is unique among all proxy suppliers the ConsumerAdmin has created.

Parameters

ctype	A flag indicating which style of push-style proxy supplier to create.
proxy_id	The unique identifier of the new proxy supplier.

Exceptions

AdminLimitExceeded	The number of consumers currently connected to the channel with which the target ConsumerAdmin is associated exceeds the value of the MaxConsumers administrative property.

ConsumerAdmin::destroy()

```
void destroy();
```

Destroys all proxies under the administration of the target object, and then destroys the target object itself. When destroying each object, it frees any storage associated with the object in question, and then invalidates the object's IOR.

CosNotifyChannelAdmin:: SupplierAdmin Interface

```
//IDL
interface SupplierAdmin :
        CosNotification::QoSAdmin,
        CosNotifyComm::NotifyPublish,
        CosNotifyFilter::FilterAdmin,
        CosEventChannelAdmin::SupplierAdmin
{
    readonly attribute AdminID MyID;
    readonly attribute EventChannel MyChannel;

    readonly attribute InterFilterGroupOperator MyOperator;

    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)
        raises ( AdminLimitExceeded );

    ProxyConsumer obtain_notification_push_consumer (
            in ClientType ctype,
            out ProxyID proxy_id)
        raises ( AdminLimitExceeded );

    ProxyConsumer obtain_txn_notification_push_consumer (
            in ClientType ctype,
            out ProxyID proxy_id)
        raises ( AdminLimitExceeded );

void destroy();
};
```

The SupplierAdmin interface defines the behavior of objects that create and manage lists of proxy consumers within an event channel. A event channel can have any number of SupplierAdmin instances associated with it. Each instance is responsible for creating and managing a list of proxy consumers that share a common set of QoS property settings, and a common set of filters. This feature enables clients to 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 event forwarding decisions using a common set of filters.

The SupplierAdmin interface inherits QoSAdmin. This enables each SupplierAdmin to manage a set of QoS property settings. These QoS property settings are assigned as the default QoS property settings for any proxy consumer created by a SupplierAdmin.

The SupplierAdmin interface inherits from the FilterAdmin interface defined in CosNotifyFilter, enabling each SupplierAdmin to maintain a list of filters. These filters encapsulate subscriptions that apply to all proxy consumer objects that have been created by a given SupplierAdmin instance.

The SupplierAdmin interface also inherits from the NotifyPublish interface defined in CosNotifyComm. This inheritance enables a SupplierAdmin to be the target of an offer_change request made by a supplier, and for the change in event types being offered to be shared by all proxy consumer that were created by the target SupplierAdmin. This optimizes the notification of a group of proxy consumers that have been created by the same SupplierAdmin of changes to the types of events being offered by suppliers.

The SupplierAdmin interface also inherits from CosEventChannelAdmin::SupplierAdmin. This inheritance enables clients to use the SupplierAdmin interface to create pure OMG event service style proxy consumer objects. Proxy consumer objects created in this manner do not support configuration of QoS properties, and do not have associated filters. Proxy consumer objects created through the inherited CosEventChannelAdmin::SupplierAdmin interface do not have unique identifiers associated with them, whereas proxy consumers created by invoking the operations supported by the SupplierAdmin interface do.

The SupplierAdmin interface supports a read-only attribute that maintains a reference to the EventChannel that created a given SupplierAdmin. The SupplierAdmin interface also supports a read-only attribute that contains a

numeric identifier that is assigned to a SupplierAdmin the event channel that creates it. This identifier is unique among all SupplierAdmins created by a given channel.

A SupplierAdmin maintains a list of filters that are applied to all proxy consumers it creates. Each proxy consumer also supports a list of filters that apply only that proxy. When combining these two lists during the evaluation of an event, either AND or OR semantics can be applied. The choice is determined by an input flag upon creation of the SupplierAdmin, and the operator that is used for this purpose by a given SupplierAdmin is maintained in a read-only attribute.

Each SupplierAdmin assigns a unique numeric identifier to each proxy consumer 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. The SupplierAdmin interface also supports an operation which, when provided with the unique identifier of a proxy consumer, returns the object reference of that proxy consumer object. Finally, the SupplierAdmin interface supports operations that can create the various styles of proxy consumers supported by the event channel.

SupplierAdmin::MyID

```
readonly attribute AdminID MyID;
```

Maintains the unique identifier of the target SupplierAdmin. This ID is assigned to it upon creation by the event channel.

SupplierAdmin::MyChannel

```
readonly attribute EventChannel MyChannel;
```

Maintains an object reference to the event channel that created the SupplierAdmin.

SupplierAdmin::MyOperator

```
readonly attribute InterFilterGroupOperator MyOperator;;
```

Maintains the information regarding whether AND or OR semantics are used during the evaluation of events when combining the filters associated with the target SupplierAdmin and those defined on a given proxy consumer.

SupplierAdmin::pull_consumers

```
readonly attribute ProxyIDSeq pull_consumers;
```

Contains the list of unique identifiers assigned by a SupplierAdmin to each pull-style proxy consumer it has created.

SupplierAdmin::push_consumers

```
readonly attribute ProxyIDSeq push_consumers;
```

Contains the list of unique identifiers assigned by a SupplierAdmin to each push-style proxy consumer it has created.

SupplierAdmin::get_proxy_consumer()

```
ProxyConsumer get_proxy_consumer ( in ProxyID proxy_id )
raises ( ProxyNotFound );
```

Returns an object reference to the proxy consumer whose unique identifier was specified.

Parameters

proxy_id	The numeric identifier associated with one of the proxy consumers created by the target SupplierAdmin.

Exceptions

ProxyNotFound	The input parameter does not correspond to the unique identifier of a proxy consumer created by the target SupplierAdmin.

SupplierAdmin::obtain_notification_pull_consumer()

```
ProxyConsumer obtain_notification_pull_consumer (
            in ClientType ctype,
            out ProxyID proxy_id)
        raises ( AdminLimitExceeded );
```

Creates an instances of a pull-style proxy consumers and returns an object reference to the new proxy.

Three varieties of pull-style proxy consumers are defined:

- The `ProxyPullConsumer` interface supports connections to pull suppliers that send events as `Any`s.
- The `StructuredProxyPullConsumer` interface supports connections to pull suppliers that send structured events.
- The `SequenceProxyPullConsumer` interface supports connections to pull suppliers that send sequences of structured events.

The input parameter flag indicates which type of pull style proxy to create.

The target `SupplierAdmin` creates the new pull-style proxy consumer and assigns it a numeric identifier that is unique among all proxy consumers it has created.

Parameters

`ctype`	A flag indicating which style of pull-style proxy consumer to create.
`proxy_id`	The unique identifier of the new proxy consumer.

Exceptions

`AdminLimitExceeded`	The number of consumers currently connected to the channel that the target `SupplierAdmin` is associated with exceeds the value of the `MaxSuppliers` administrative property.

SupplierAdmin::obtain_notification_push_consumer()

```
ProxyConsumer obtain_notification_push_consumer (
        in ClientType ctype,
```

```
        out ProxyID proxy_id)
    raises ( AdminLimitExceeded );
```

Creates an instance of a push-style proxy supplier and returns an object reference to the new proxy.

Three varieties of push-style proxy consumer are defined:

- The ProxyPushConsumer interface supports connections to push consumers that receive events as Anys.
- The StructuredProxyPushConsumer interface supports connections to push consumers that receive structured events.
- The SequenceProxyPushConsumer interface supports connections to push consumers that receive sequences of structured events.

The input parameter flag indicates which type of push-style proxy to create.

The target SupplierAdmin creates the new push-style proxy consumer and assigns it a numeric identifier that is unique among all proxy suppliers it has created.

Parameters

ctype	A flag that indicates the type of push-style proxy consumer to create.
proxy_id	The unique identifier of the new proxy consumer.

Exceptions

AdminLimitExceeded	The number of consumers currently connected to the channel that the target SupplierAdmin is associated with exceeds the value of the MaxSuppliers administrative property.

SupplierAdmin::destroy()

```
void destroy();
```

Iteratively destroys each proxy under the administration of the target object, and finally destroys the target object itself. When destroying each object, it frees any storage associated with the object, and then invalidates the object's IOR.

CosNotifyChannelAdmin:: EventChannel Interface

```
//IDL
interface EventChannel :
        CosNotification::QoSAdmin,
        CosNotification::AdminPropertiesAdmin,
        CosEventChannelAdmin::EventChannel
{
    readonly attribute EventChannelFactory MyFactory;
    readonly attribute ConsumerAdmin default_consumer_admin;
    readonly attribute SupplierAdmin default_supplier_admin;
    readonly attribute CosNotifyFilter::FilterFactory
                                    default_filter_factory;

    ConsumerAdmin new_for_consumers(
            in InterFilterGroupOperator op,
            out AdminID id );

    SupplierAdmin new_for_suppliers(
            in InterFilterGroupOperator op,
            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();
};
```

The `EventChannel` interface defines the behavior of an event channel. This interface inherits from `CosEventChannelAdmin::EventChannel`; this makes an instance of the notification service `EventChannel` interface fully compatible with an OMG event service style untyped event channel.

Inheritance of CosEventChannelAdmin::EventChannel enables an instance of the EventChannel interface to create event service style ConsumerAdmin and SupplierAdmin instances. These instances can subsequently be used to create pure event service style proxies, which support connections to pure event service style suppliers and consumers.

While notification service style proxies and admin objects have unique identifiers associated with them, enabling their references to be obtained by invoking operations on the notification service style admin and event channel interfaces, event service style proxies and admin objects do not have associated unique identifiers, and cannot be returned by invoking an operation on the notification service style admin or event channel interfaces.

The EventChannel interface also inherits from the QoSAdmin and the AdminPropertiesAdmin interfaces defined in CosNotification. Inheritance of these interfaces enables a notification service style event channel to manage lists of QoS and administrative properties.

The EventChannel interface supports a read-only attribute that maintains a reference to the EventChannelFactory that created it. Each instance of the EventChannel interface has an associated default ConsumerAdmin and an associated default SupplierAdmin, both of which exist upon creation of the channel and that have the unique identifier of zero. Admin object identifiers must only be unique among a given type of admin, which means that the identifiers assigned to ConsumerAdmin objects can overlap those assigned to SupplierAdmin objects. The EventChannel interface supports read-only attributes that maintain references to these default admin objects.

The EventChannel interface supports operations that create new ConsumerAdmin and SupplierAdmin instances. The EventChannel interface also supports operations that, when provided with the unique identifier of an admin object, can return references to the ConsumerAdmin and SupplierAdmin instances associated with a given EventChannel. Finally, the EventChannel interface supports operations that return the sequence of unique identifiers of all ConsumerAdmin and SupplierAdmin instances associated with a given EventChannel.

EventChannel::MyFactory

```
readonly attribute EventChannelFactory MyFactory;
```

Maintains the object reference of the event channel factory that created a given EventChannel.

EventChannel::default_consumer_admin

```
readonly attribute ConsumerAdmin default_consumer_admin;
```

Maintains a reference to the default ConsumerAdmin associated with the target EventChannel. Each EventChannel instance has an associated default ConsumerAdmin, that exists upon creation of the channel and is assigned the unique identifier of zero. Clients can create additional event service style ConsumerAdmin by invoking the inherited for_consumers operation, and additional notification service style ConsumerAdmin by invoking the new_for_consumers operation defined by the EventChannel interface.

EventChannel::default_supplier_admin

```
readonly attribute SupplierAdmin default_supplier_admin;
```

Maintains a reference to the default SupplierAdmin associated with the target EventChannel. Each EventChannel has an associated default SupplierAdmin, that exists upon creation of the channel and is assigned the unique identifier of zero. Clients can create additional event service style SupplierAdmin by invoking the inherited for_suppliers operation, and additional notification service style SupplierAdmin by invoking the new_for_suppliers operation defined by the EventChannel interface.

EventChannel::default_filter_factory

```
readonly attribute CosNotifyFilter::FilterFactory
            default_filter_factory;
```

Maintains an object reference to the default factory to be used by its associated EventChannel for creating filters. If the target channel does not support a default filter factory, the attribute maintains the value of OBJECT_NIL.

EventChannel::new_for_consumers()

```
ConsumerAdmin new_for_consumers(
        in InterFilterGroupOperator op,
        out AdminID id );
```

Creates a notification service style ConsumerAdmin. The new instance is assigned a unique identifier by the target EventChannel that is unique among all ConsumerAdmins currently associated with the channel. Upon completion, the operation returns the reference to the new ConsumerAdmin, and the unique identifier assigned to the new ConsumerAdmin as the output parameter.

Parameters

op	A boolean flag indicating whether to use AND or OR semantics when the ConsumerAdmin's filters are combined with the filters associated with any supplier proxies the ConsumerAdmin creates.
id	The unique identifier assigned to the new ConsumerAdmin.

EventChannel::new_for_suppliers()

```
SupplierAdmin new_for_suppliers(
        in InterFilterGroupOperator op,
        out AdminID id );
```

Creates a notification service style SupplierAdmin. The new SupplierAdmin is assigned an identifier by the target EventChannel that is unique among all SupplierAdmins currently associated with the channel. Upon completion, the operation returns the reference to the new SupplierAdmin, and the unique identifier assigned to the new SupplierAdmin as the output parameter.

Parameters

op	A boolean flag indicating whether to use AND or OR semantics when the SupplierAdmin's filters are combined with the filters associated with any supplier proxies the SupplierAdmin creates.
id	The unique identifier assigned to the new SupplierAdmin.

EventChannel::get_consumeradmin()

```
ConsumerAdmin get_consumeradmin ( in AdminID id )
    raises (AdminNotFound);
```

Returns a reference to one of the ConsumerAdmins associated with the target EventChannel.

Note: While a notification service event channel can support both event service and notification service style ConsumerAdmins, only notification service style ConsumerAdmins have unique identifiers.

Parameters

id	A numeric value that is the unique identifier of one of the ConsumerAdmins associated with the target EventChannel.

Exceptions

AdminNotFound	The id is not the identifier of one of the ConsumerAdmins associated with the target EventChannel.

EventChannel::get_supplieradmin()

```
SupplierAdmin get_supplieradmin ( in AdminID id )
    raises (AdminNotFound);
```

Returns a reference to one of the SupplierAdmins associated with the target EventChannel.

Note: While a notification service style event channel can support both Event service and notification service style SupplierAdmins, only notification service style SupplierAdmins have unique identifiers.

Parameters

id	A numeric value that is the unique identifier of one of the SupplierAdmins associated with the target EventChannel.

Exceptions

AdminNotFound	The `id` is not the unique identifier of one of the `SupplierAdmins` associated with the target `EventChannel`.

EventChannel::get_all_consumeradmins()

```
AdminIDSeq get_all_consumeradmins();
```

Returns a sequence of unique identifiers assigned to all notification service style `ConsumerAdmins` created by the target `EventChannel`.

EventChannel::get_all_supplieradmins()

```
AdminIDSeq get_all_supplieradmins();
```

Returns a sequence of unique identifiers assigned to all notification service style `SupplierAdmins` created by the target `EventChannel`.

CosNotifyChannelAdmin:: EventChannelFactory Interface

```
//IDL
interface EventChannelFactory
{
    EventChannel create_channel (
            in CosNotification::QoSProperties initial_qos,
            in CosNotification::AdminProperties initial_admin,
            out ChannelID id)
        raises(CosNotification::UnsupportedQoS,
               CosNotification::UnsupportedAdmin );

    ChannelIDSeq get_all_channels();

    EventChannel get_event_channel ( in ChannelID id )
        raises (ChannelNotFound);
};
```

The `EventChannelFactory` interface defines operations for creating and managing event channels. It supports a routine that creates new instances of event channels and assigns unique numeric identifiers to them.

The `EventChannelFactory` interface supports a routine that returns the unique identifiers assigned to all event channels created by a given `EventChannelFactory`, and another routine that, given the unique identifier of an event channel, returns the object reference of that event channel.

EventChannelFactory::create_channel()

```
EventChannel create_channel (
            in CosNotification::QoSProperties initial_qos,
            in CosNotification::AdminProperties initial_admin,
            out ChannelID id)
        raises(CosNotification::UnsupportedQoS,
               CosNotification::UnsupportedAdmin );
```

Creates an instance of an event channel and returns an object reference to the new channel.

Parameters

`initial_qos`	A list of name-value pairs specifying the initial QoS property settings for the new channel.
`initial_admin`	A list of name-value pairs specifying the initial administrative property settings for the new channel.
`id`	A numeric identifier that is assigned to the new event channel and which is unique among all event channels created by the target object.

Exceptions

`UnsupportedQoS`	Raised if no implementation of the `EventChannel` interface exists that can support all of the requested QoS property settings. This exception contains a sequence of data structures which identifies the name of a QoS property in the input list whose requested setting could not be satisfied, along with an error code and a range of settings for the property that could be satisfied.
`UnsupportedAdmin`	Raised if no implementation of the `EventChannel` interface exists that can support all of the requested administrative property settings.This exception contains a sequence of data structures that identifies the name of an administrative property in the input list whose requested setting could not be satisfied, along with an error code and a range of settings for the property that could be satisfied.

EventChannelFactory::get_all_channels()

```
ChannelIDSeq get_all_channels();
```

Returns a sequence containing all of the unique numeric identifiers for the event channels which have been created by the target object.

EventChannelFactory::get_event_channel()

```
EventChannel get_event_channel ( in ChannelID id )
    raises (ChannelNotFound);
```

Returns the object reference of the event channel corresponding to the input identifier.

Parameters

`id`	A numeric value that is the unique identifier of an event channel that has been created by the target object.

Exceptions

`ChannelNotFound`	The `id` does not correspond to he unique identifier of an event channel that has been created by the target object.

IT_NotifyChannelAdmin::GroupProxyPushSupplier Interface

```
interface GroupProxyPushSupplier :
          CosNotifyChannelAdmin::ProxyPushSupplier
  {
    void connect_group_any_push_consumer(
         in IT_NotifyComm::GroupPushConsumer group_push_consumer)
    raises(CosEventChannelAdmin::AlreadyConnected,
           CosEventChannelAdmin::TypeError);
  };
```

The `GroupProxyPushSupplier` interface supports connections to the channel by endpoint groups receiving events from the channel as untyped `Any`s. Note that such endpoint groups are functionally similar to OMG Event Service push-style consumers of untyped events. The `GroupProxyPushSupplier` interface defined here, however, supports event filtering and configuration of QoS properties in addition to taking advantage of the IP/Multicast message transport.

Through inheritance of the `ProxyPushSupplier` interface, the `GroupProxyPushSupplier` interface supports administration of QoS properties, administration of a list of associated filter, mapping filters for event priority and lifetime, and a read-only attribute containing a reference to the `ConsumerAdmin` that created it. This inheritance implies that a `GroupProxyPushSupplier` instance supports an operation that returns the list of event types that the proxy supplier can supply, and an operation that returns information about the group's ability to accept a QoS request. The `GroupProxyPushSupplier` interface also inherits a pair of operations that suspend and resume the connection between a `GroupProxyPushSupplier` instance and its associated endpoint group. During the time a connection is suspended, the `GroupProxyPushSupplier` accumulates events destined for the endpoint group but does not transmit them until the connection is resumed.

The GroupProxyPushSupplier interface inherits the NotifySubscribe interface defined in CosNotifyComm, enabling it to be notified whenever its associated endpoint group changes the list of event types it is interested in receiving.

The GroupProxyPushSupplier interface also inherits from the PushSupplier interface defined in CosEventComm. This interface supports the operation to disconnect the GroupProxyPushSupplier from its associated endpoint group.

The GroupProxyPushSupplier interface defines the operation to establish the connection over which the consumer's endpoint group receives events from the channel.

GroupProxyPushSupplier:: connect_group_any_push_consumer()

```
void connect_group_any_push_consumer(
     in IT_NotifyComm::GroupPushConsumer group_push_consumer)
raises(CosEventChannelAdmin::AlreadyConnected,
       CosEventChannelAdmin::TypeError);
```

Establishes a connection between an endpoint group of consumers expecting events in the form of Anys, and an event. Once the connection is established, the GroupProxyPushSupplier sends events to the endpoint group by invoking push() on the connected consumer.

Parameters

group_push_consumer	The reference to an object supporting the GroupPushConsumer interface defined in IT_NotifyComm. This reference is that of a consumer connecting to the channel for the members of an end-point group.

Exceptions

`AlreadyConnected`	Raised if the target object of this operation is already connected to a push consumer object.
`TypeError`	An implementation of the `GroupProxyPushSupplier` interface may impose additional requirements on the interface supported by the push consumers in a group (for example, it may be designed to invoke some operation other than push in order to transmit events). If the consumers in the group being connected do not meet those requirements, this operation raises the `TypeError` exception.

IT_NotifyChannelAdmin::GroupStructuredProxyPushSupplier Interface

```
interface GroupStructuredProxyPushSupplier :
          CosNotifyChannelAdmin::StructuredProxyPushSupplier
  {
    void connect_group_structured_push_consumer(
         in IT_NotifyComm::GroupStructuredPushConsumer
         group_push_consumer)
    raises(CosEventChannelAdmin::AlreadyConnected,
           CosEventChannelAdmin::TypeError);
  };
```

The `GroupStructuredProxyPushSupplier` interface supports connections to the channel by endpoint groups that receive structured events from the channel.

Through inheritance of `StructuredProxyPushSupplier`, the `GroupStructuredProxyPushSupplier` interface supports administration of QoS properties, administration of a list of associated filters, mapping filters for event priority and lifetime, and a read-only attribute containing a reference to the `ConsumerAdmin` that created it. This inheritance implies that a `GroupStructuredProxyPushSupplier` instance supports an operation that returns the list of event types that the proxy supplier can supply, and an operation that returns information about the group's ability to accept a QoS request. The `GroupStructuredProxyPushSupplier` interface also inherits a pair of operations to suspend and resume the connection between a `GroupStructuredProxyPushSupplier` instance and its associated endpoint group. During the time a connection is suspended, the `GroupStructuredProxyPushSupplier` accumulates events destined for the endpoint group but does not transmit them until the connection is resumed.

The `GroupStructuredProxyPushSupplier` interface also inherits from the `StructuredPushSupplier` interface defined in `CosNotifyComm`. This interface defines the operation to disconnect the `GroupStructuredProxyPushSupplier` from its associated endpoint group. In addition, the

GroupStructuredProxySupplier interface inherits from NotifySubscribe, enabling it to be notified whenever its associated endpoint group changes the list of event types it is interested in receiving.

The GroupStructuredProxyPushSupplier interface defines the operation to establish the connection over which the consumer's endpoint group receives events from the channel.

GroupStructuredProxyPushSupplier:: connect_group_structured_push_consumer()

```
void connect_group_group_structured_push_consumer(
     in IT_NotifyComm::GroupStructuredPushConsumer
        group_push_consumer)
raises(CosEventChannelAdmin::AlreadyConnected,
       CosEventChannelAdmin::TypeError );
```

Establishes a connection between an endpoint group of consumers expecting structured events and an event channel. Once the connection is established, the GroupStructuredProxyPushSupplier sends events to the endpoint group invoking push_structured_event() on the connected consumer.

Parameters

group_push_consumer	A reference to an object supporting the GroupStructuredPushConsumer interface defined in IT_NotifyComm. This reference is that of a consumer connecting to the channel for the members of an end-point group.

Exceptions

AlreadyConnected	Raised if the target object of this operation is already connected to a push consumer.

`TypeError`	An implementation of the `GroupStructuredProxyPushSupplier` interface may impose additional requirements on the interface supported by an endpoint group (for example, it may be designed to invoke some operation other than `push_structured_event` to transmit events). If the members of the endpoint group being connected do not meet those requirements, this operation raises the `TypeError` exception.

IT_NotifyChannelAdmin: GroupSequenceProxyPushSupplier Interface

```
interface GroupSequenceProxyPushSupplier :
          CosNotifyChannelAdmin::SequenceProxyPushSupplier
  {
    void connect_group_sequence_push_consumer(
         in IT_NotifyComm::GroupSequencePushConsumer
            group_push_consumer)
    raises(CosEventChannelAdmin::AlreadyConnected,
           CosEventChannelAdmin::TypeError);
  };
```

The GroupSequenceProxyPushSupplier interface supports connections to the channel by endpoint groups that receive sequences of structured events from the channel.

Through inheritance of SequenceProxyPushSupplier, the GroupSequenceProxyPushSupplier interface supports administration of QoS properties, administration of a list of associated filter objects, and a read-only attribute containing a reference to the ConsumerAdmin that created it. This inheritance also implies that a GroupSequenceProxyPushSupplier instance supports an operation that returns the list of event types that the proxy supplier can supply, and an operation that returns information about the endpoint group's ability to accept a QoS request. The GroupSequenceProxyPushSupplier interface also inherits a pair of operations which suspend and resume the connection between a GroupSequenceProxyPushSupplier instance and its associated endpoint group. During the time a connection is suspended, the GroupSequenceProxyPushSupplier accumulates events destined for the endpoint group but does not transmit them until the connection is resumed.

The GroupSequenceProxyPushSupplier interface also inherits from the SequencePushSupplier interface defined in CosNotifyComm. This interface supports the operation to close the connection from the endpoint group to the GroupSequenceProxyPushSupplier. Since the SequencePushSupplier

interface inherits from NotifySubscribe, a GroupSequenceProxyPushSupplier can be notified whenever the list of event types that its associated endpoint group is interested in receiving changes.

The GroupSequenceProxyPushSupplier interface defines the operation to establish the connection over which the endpoint group receives events from the channel.

GroupSequenceProxyPushSupplier:: connect_group_sequence_push_consumer()

```
void connect_group_sequence_push_consumer(
     in IT_NotifyComm::GroupSequencePushConsumer
        group_push_consumer)
raises(CosEventChannelAdmin::AlreadyConnected,
       CosEventChannelAdmin::TypeError);
```

Establishes a connection between an endpoint group of consumers expecting sequences of structured events and an event channel. Once the connection is established, the GroupSequenceProxyPushSupplier sends events to its endpoint group by invoking push_structured_events() on the connected consumer.

Parameters

group_push_consumer	A reference to an object supporting the GroupSequencePushConsumer interface defined in IT_NotifyComm. This reference is that of a consumer connecting to the channel for the members of an endpoint group.

Exceptions

AlreadyConnected	Raised if the target object of this operation is already connected to a push consumer.

`TypeError`	An implementation of the `GroupSequenceProxyPushSupplier` interface may impose additional requirements on the interface supported by an endpoint group (for example, it may be designed to invoke some operation other than `push_structured_events` in order to transmit events). If the members of the endpoint group being connected do not meet those requirements, this operation raises the `TypeError` exception.

CosNotifyComm Module

CosNotifyComm specifies the following interfaces to instantiate notification service clients:

PushConsumer	PushSupplier
PullConsumer	PullSupplier
StructuredPushConsumer	StructuredPushSupplier
StructuredPullConsumer	StructuredPullSupplier
SequencePushConsumer	SequencePushSupplier
SequencePullConsumer	SequencePullSupplier

The module also specifies the NotifyPublish and NotifySubscribe interfaces to facilitate informing notification clients about subscription and publication changes.

CosNotifyComm Exceptions

CosNotifyComm::InvalidEventType Exception

```
exception InvalidEventType{ CosNotification::EventType type };
```

Raised when the specified EventType is not syntactically correct. It returns the name of the invalid event type.

Note: The Orbix notification service does not throw this exception.

CosNotifyComm::NotifyPublish Interface

```
interface NotifyPublish {
    void offer_change (
            in CosNotification::EventTypeSeq added,
            in CosNotification::EventTypeSeq removed )
        raises ( InvalidEventType );
};
```

The `NotifyPublish` interface supports an operation that allows a supplier to announce, or publish, the names of the event types it supplies. It is an abstract interface which is inherited by all notification service consumer interfaces, and it enables suppliers to inform consumers supporting this interface of the types of events they intend to supply.

NotifyPublish::offer_change()

```
void offer_change (
        in CosNotification::EventTypeSeq added,
        in CosNotification::EventTypeSeq removed )
    raises ( InvalidEventType );
```

Allows a supplier of notifications to announce, or publish, the names of the types of events it supplies.

Note: Each event type name consists of two components: the name of the domain in which the event type has meaning, and the name of the actual event type. Either component of a type name may specify a complete domain/event type name, a domain/event type name containing the wildcard '*' character, or the special event type name "%ALL".

Parameters

`added`	A sequence of event type names specifying those event types which the event supplier plans to supply.
`removed`	Sequence of event type names specifying those event types which the client no longer plans to supply.

Exceptions

`InvalidEventType`	One of the event type names supplied in either input parameter is syntactically invalid. In this case, the invalid name is returned in the type field of the exception.

CosNotifyComm::NotifySubscribe Interface

```
interface NotifySubscribe {
    void subscription_change(
            in CosNotification::EventTypeSeq added,
            in CosNotification::EventTypeSeq removed )
        raises ( InvalidEventType );
};
```

The `NotifySubscribe` interface supports an operation allowing a consumer to inform suppliers of the event types it wishes to receive. It is an abstract interface that is inherited by all notification service supplier interfaces. Its main purpose is to enable consumers to inform suppliers of the event types they are interested in, ultimately enabling the suppliers to avoid supplying events that are not of interest to any consumer.

NotifySubscribe::subscription_change()

```
void subscription_change(
        in CosNotification::EventTypeSeq added,
        in CosNotification::EventTypeSeq removed )
    raises ( InvalidEventType );
```

Allows a consumer to inform suppliers of the event types it wishes to receive.

Note: Each event type name is comprised of two components: the name of the domain in which the event type has meaning, and the name of the actual event type. Also note that either component of a type name may specify a complete domain/event type name, a domain/event type name containing the wildcard '*' character, or the special event type name "%ALL".

Parameters

`added`	A sequence of event type names specifying the event types the consumer wants to add to its subscription list.
`removed`	A sequence of event type names specifying the event types the consumer wants to remove from its subscription list.

Exceptions

`InvalidEventType`	One of the event type names supplied in either input parameter is syntactically invalid. The invalid name is returned in the type field of the exception.

CosNotifyComm::PushConsumer Interface

```
interface PushConsumer :
  NotifyPublish,
  CosEventComm::PushConsumer
{
};
```

The PushConsumer interface inherits all the operations of CosEventComm::PushConsumer. In addition, the PushConsumer interface inherits the NotifyPublish interface described above, which enables a supplier to inform an instance supporting this interface whenever there is a change to the types of events it intends to produce.

Note: An object supporting PushConsumer can receive all events that were supplied to its associated channel. How events supplied to the channel in other forms are internally mapped for delivery to a PushConsumer is summarized in the *CORBA Notification Service Guide*.

CosNotifyComm::PullConsumer Interface

```
interface PullConsumer :
  NotifyPublish,
  CosEventComm::PullConsumer
{
};
```

The `PullConsumer` interface inherits all the operations of `CosEventComm::PullConsumer`. In addition, the `PullConsumer` interface inherits the `NotifyPublish` interface described above, which enables a supplier to inform an instance supporting this interface whenever there is a change to the types of events it intends to produce.

Note: An object supporting `PullConsumer` can receive all events that were supplied to its associated channel. How events supplied to the channel in other forms are internally mapped for delivery to a `PullConsumer` is summarized in the *CORBA Notification Service Guide*.

CosNotifyComm::PullSupplier Interface

```
interface PullSupplier :
  NotifySubscribe,
  CosEventComm::PullSupplier
{
};
```

The `PullSupplier` interface inherits all the operations of `CosEventComm::PullSupplier`. In addition, the `PullSupplier` interface inherits the `NotifySubscribe` interface described above, which enables a consumer to inform an instance supporting this interface whenever there is a change to the types of events it wishes to receive.

Note: An object supporting the `PullSupplier` interface can transmit events that can potentially be received by any consumer connected to the channel. How events supplied to the channel in other forms are translated is summarized in the *CORBA Notification Service Guide*

CosNotifyComm::PushSupplier Interface

```
interface PushSupplier :
  NotifySubscribe,
  CosEventComm::PushSupplier
{
};
```

The PushSupplier interface inherits all the operations of CosEventComm::PushSupplier. In addition, the PushSupplier interface inherits the NotifySubscribe interface described above, which enables a consumer to inform an instance supporting this interface whenever there is a change to the types of events it wishes to receive.

Note: An object supporting the PushSupplier interface can transmit events that can potentially be received by any consumer connected to the channel. How events supplied to the channel in other forms are translated is summarized in the *CORBA Notification Service Guide*

CosNotifyComm::
StructuredPushConsumer Interface

```
interface StructuredPushConsumer : NotifyPublish {
    void push_structured_event(
            in CosNotification::StructuredEvent notification)
        raises(CosEventComm::Disconnected);
    void disconnect_structured_push_consumer();
};
```

The StructuredPushConsumer interface supports an operation enabling consumers to receive structured events by the push model. It also defines an operation to disconnect the push consumer from its associated proxy supplier. In addition, the StructuredPushConsumer interface inherits the NotifyPublish interface described above, which enables a supplier to inform an instance supporting this interface whenever there is a change to the types of events it intends to produce.

Note: An object supporting the StructuredPushConsumer interface can receive all events that were supplied to its associated channel, including events supplied in a form other than a structured event. How events supplied to the channel in other forms are internally mapped into a structured event for delivery to a StructuredPushConsumer is summarized in the *CORBA Notification Service Guide*.

StructuredPushConsumer::push_structured_event()

```
void push_structured_event(
        in CosNotification::StructuredEvent notification)
    raises(CosEventComm::Disconnected);
```

Enables consumers to receive structured events by the push model.

Parameters

`notification`	A parameter of type `StructuredEvent` as defined in the `CosNotification` module. When the method returns this parameter contains a structured event being delivered to the consumer by its proxy supplier.

Exceptions

`Disconnected`	This operation was invoked on a `StructuredPushConsumer` instance that is not currently connected to a proxy supplier.

StructuredPushConsumer:: disconnect_structured_push_consumer()

```
void disconnect_structured_push_consumer();
```

Terminates a connection between the target `StructuredPushConsumer` and its associated proxy supplier. That the target `StructuredPushConsumer` releases all resources allocated to support the connection, and disposes of its own object reference.

CosNotifyComm::
StructuredPullConsumer Interface

```
interface StructuredPullConsumer : NotifyPublish
{
    void disconnect_structured_pull_consumer();
};
```

The `StructuredPullConsumer` defines an operation that can be invoked to disconnect the pull consumer from its associated supplier. In addition, the `StructuredPullConsumer` interface inherits the `NotifyPublish` interface, which enables a supplier to inform an instance supporting this interface whenever there is a change to the types of events it intends to produce.

Note: An object supporting the `StructuredPullConsumer` interface can receive all events that were supplied to its associated channel, including events supplied in a form other than a structured event. How events supplied to the channel in other forms are internally mapped into a structured event for delivery to a `StructuredPullConsumer` is summarized in the *CORBA Notification Service Guide*.

StructuredPullConsumer:: disconnect_structured_pull_consumer()

```
void disconnect_structured_pull_consumer();
```

Terminates a connection between the target `StructuredPullConsumer`, and its associated supplier proxy. The target `StructuredPullConsumer` releases all resources allocated to support the connection, and disposes of its own object reference.

CosNotifyComm::StructuredPullSupplier Interface

```
interface StructuredPullSupplier : 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();
};
```

The `StructuredPullSupplier` interface supports operations that enable suppliers to transmit structured events by the pull model. It also defines an operation to disconnect the pull supplier from its associated consumer proxy. In addition, the `StructuredPullSupplier` interface inherits the `NotifySubscribe` interface, which enables a consumer to inform an instance supporting this interface whenever there is a change to the types of events it is interested in receiving.

Note: An object supporting the `StructuredPullSupplier` interface can transmit events that can potentially be received by any consumer connected to the channel, including those which consume events in a form other than a structured event. How events supplied to the channel in other forms are translated is summarized in the *CORBA Notification Service Guide*

StructuredPullSupplier::pull_structured_event()

```
CosNotification::StructuredEvent pull_structured_event()
        raises(CosEventComm::Disconnected);
```

Blocks until an event is available for transmission, at which time it returns an instance of a structured event containing the event being delivered to its connected consumer proxy.

Exceptions

`Disconnected`	The operation was invoked on a `StructuredPullSupplier` that is not currently connected to a consumer proxy.

StructuredPullSupplier::try_pull_structured_event()

```
CosNotification::StructuredEvent try_pull_structured_event(
        out boolean has_event)
    raises(CosEventComm::Disconnected);
```

If an event is available for delivery at the time the operation was invoked, the method returns a structured event that contains the event being delivered to its connected consumer and the output parameter of the operation is set to `TRUE`. If no event is available to return upon invocation, the operation returns immediately with the value of the output parameter set to `FALSE`. In this case, the return value does not contain a valid event.

Parameters

`has_event`	An output parameter of type `boolean` that indicates whether or not the return value actually contains an event.

Exceptions

`Disconnected`	The operation was invoked on a `StructuredPullSupplier` that is not currently connected to a consumer proxy.

StructuredPullSupplier::disconnect_structured_pull_supplier()

```
void disconnect_structured_pull_supplier();
```

Terminates a connection between the target `StructuredPullSupplier` and its associated consumer. The target `StructuredPullSupplier` releases all resources allocated to support the connection, and disposes of its own object reference.

CosNotifyComm:: StructuredPushSupplier Interface

```
interface StructuredPushSupplier : NotifySubscribe {
    void 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 proxy. In addition, the `StructuredPushSupplier` interface inherits `NotifySubscribe`, which enables a consumer to inform an instance supporting this interface whenever there is a change to the types of events it is interested in receiving.

Note: An object supporting the `StructuredPushSupplier` interface can transmit events which can potentially be received by any consumer connected to the channel, including those which consume events in a form other than a structured event. How events supplied to the channel are translated is summarized in the *CORBA Notification Service Guide*.

StructuredPushSupplier:: disconnect_structured_push_supplier()

```
void disconnect_structured_push_supplier();
```

Terminates a connection between the target `StructuredPushSupplier`, and its associated consumer. The target `StructuredPushSupplier` releases all resources allocated to support the connection, and disposes of its own object reference.

CosNotifyComm:: SequencePushConsumer Interface

```
interface SequencePushConsumer : NotifyPublish {
    void push_structured_events(
            in CosNotification::EventBatch notifications)
        raises(CosEventComm::Disconnected);
    void disconnect_sequence_push_consumer();
};
```

The SequencePushConsumer interface supports an operation that enables consumers to receive sequences of structured events using the push model. It also defines an operation to disconnect the push consumer from its associated supplier. The SequencePushConsumer interface inherits NotifyPublish, which enables a supplier to inform an instance supporting this interface whenever there is a change to the types of events it intends to produce.

Note: An object supporting the SequencePushConsumer interface can receive all events which are supplied to its associated channel, including events supplied in a form other than a sequence of structured events. How events supplied to the channel in other forms are internally mapped into a sequence of structured events for delivery to a SequencePushConsumer is summarized in the *CORBA Notification Service Guide*.

SequencePushConsumer::push_structured_events()

```
void push_structured_events(
        in CosNotification::EventBatch notifications)
    raises(CosEventComm::Disconnected);
```

Enables consumers to receive sequences of structured events by the push model.

The maximum number of events that are transmitted within a single invocation of this operation, along with the amount of time a supplier of sequences of structured events packs individual events into the sequence before invoking this operation, are controlled by QoS property settings as described in the *CORBA Notification Service Guide*.

Parameters

`notifications`	A parameter of type `EventBatch` as defined in the `CosNotification` module. Upon invocation, this parameter contains a sequence of structured events being delivered to the consumer by its associated supplier proxy.

Exceptions

`Disconnected`	The operation was invoked on a `SequencePushConsumer` instance that is not currently connected to a supplier proxy.

SequencePushConsumer:: disconnect_sequence_push_consumer()

```
void disconnect_sequence_push_consumer();
```

Terminates a connection between the target `SequencePushConsumer` and its associated supplier proxy. The target `SequencePushConsumer` releases all resources allocated to support the connection, and disposes of its own object reference.

CosNotifyComm::
SequencePullConsumer Interface

```
interface SequencePullConsumer : NotifyPublish {
    void disconnect_sequence_pull_consumer();
};
```

The SequencePullConsumer interface defines an operation to disconnect the pull consumer from its associated supplier. The SequencePullConsumer interface inherits NotifyPublish, which enables a supplier to inform an instance supporting this interface whenever there is a change to the types of events it intends to produce.

Note: An object supporting the SequencePullConsumer interface can receive all events that were supplied to its associated channel, including events supplied in a form other than a sequence of structured events. How events supplied to the channel in other forms are internally mapped into a sequence of structured events for delivery to a SequencePullConsumer is summarized in the *CORBA Notification Service Guide*.

SequencePullConsumer:: disconnect_sequence_pull_consumer()

```
void disconnect_sequence_pull_consumer();
```

Terminates a connection between the target SequencePullConsumer and its associated supplier. The target SequencePullConsumer releases all resources allocated to support the connection, and disposes of its own object reference.

CosNotifyComm:: SequencePullSupplier Interface

```
interface SequencePullSupplier : NotifySubscribe
{
    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();
};
```

The SequencePullSupplier interface supports operations that enable suppliers to transmit sequences of structured events using the pull model. It also defines an operation to disconnect the pull supplier from its associated consumer. The SequencePullSupplier interface inherits NotifySubscribe, which enables a consumer to inform an instance supporting this interface whenever there is a change to the types of events it is interested in receiving.

Note: An object supporting the SequencePullSupplier interface can transmit events that can be received by any consumer connected to the channel, including those which consume events in a form other than a sequence of structured events. How events supplied to the channel in the form of a sequence of structured events are internally mapped into different forms for delivery to consumers that receive events in a form other than the a sequence of structured events is summarized in the *CORBA Notification Service Guide*.

SequencePullSupplier::pull_structured_events()

```
CosNotification::EventBatch pull_structured_events(
        in long max_number )
    raises(CosEventComm::Disconnected);
```

Blocks until a sequence of structured events is available for transmission, at which time it returns the sequence containing events to be delivered to its connected consumer proxy.

The amount of time the supplier packs events into the sequence before transmitting it, along with the maximum size of any sequence it transmits (regardless of the input parameter), are controlled by QoS property settings as described in the *CORBA Notification Service Guide*.

Parameters

`max_number`	The maximum length of the sequence returned.

Exceptions

`Disconnected`	The operation was invoked on a `SequencePullSupplier` that is not currently connected to a consumer proxy.

SequencePullSupplier::try_pull_structured_events()

```
CosNotification::StructuredEvent try_pull_structured_events(
        in long max_number,
        out boolean has_event)
    raises(CosEventComm::Disconnected);
```

Returns a sequence of a structured events that contains events being delivered to its connected consumer, if such a sequence is available for delivery at the time the operation was invoked:

- If an event sequence is available for delivery and is returned as the result, the output parameter `has_event` is set to `TRUE`.
- If no event sequence is available to return upon invocation, the operation returns immediately with the value of the output parameter set to `FALSE`. In this case, the return value does not contain a valid event sequence.

Parameters

`max_number`	The maximum length of the sequence returned.
`has_event`	An output parameter of type boolean that indicates whether or not the return value actually contains a sequence of events.

Exceptions

`Disconnected`	This operation was invoked on a `SequencePullSupplier` that is not currently connected to a consumer proxy.

SequencePullSupplier::disconnect_sequence_pull_supplier()

```
void disconnect_sequence_pull_supplier();
```

Terminates a connection between the target `SequencePullSupplier` and its associated consumer. The target `SequencePullSupplier` releases all resources allocated to support the connection, and disposes of its own object reference.

CosNotifyComm:: SequencePushSupplier Interface

```
interface SequencePushSupplier : NotifySubscribe
{
    void disconnect_sequence_push_supplier();
};
```

The SequencePushSupplier interface defines an operation that to disconnect the push supplier from its associated consumer proxy. In addition, the SequencePushSupplier interface inherits NotifySubscribe, which enables a consumer to inform an instance supporting this interface whenever there is a change to the types of events it is interested in receiving.

Note: An object supporting the SequencePushSupplier interface can transmit events that can be received by any consumer connected to the channel, including those which consume events in a form other than a sequence of structured events. How events supplied to the channel in the form of a sequence of structured events are internally mapped into different forms for delivery to consumers which receive events in a form other than a sequence of structured events is summarized in the *CORBA Notification Service Guide*.

SequencePushSupplier::disconnect_sequence_push_supplier()

```
void disconnect_sequence_push_supplier();
```

Terminates a connection between the target SequencePushSupplier and its associated consumer. The target SequencePushSupplier releases all resources allocated to support the connection, and disposes of its own object reference.

IT_NotifyComm::GroupNotifyPublish Interface

```
interface GroupNotifyPublish
  {
    oneway void offer_change(
            in CosNotification::EventTypeSeq added,
            in CosNotification::EventTypeSeq removed);
  };
```

The `GroupNotifyPublish` interface supports an operation allowing a supplier to announce, or publish, the names of the types of events it supplies. It is an abstract interface which is inherited by all group consumer interfaces, and enables suppliers to inform consumers supporting this interface of the types of events they intend to supply.

When implemented by a group consumer, it allows the consumer to modify its subscription list accordingly.

GroupNotifyPublish::offer_change()

```
oneway void offer_change(
        in CosNotification::EventTypeSeq added,
        in CosNotification::EventTypeSeq removed);
```

Allows a supplier of notifications to announce, or publish, the names of the types of events it supplies to consumers using IP/Multicast.

Note: Each event type name consists of two components: the name of the domain in which the event type has meaning, and the name of the actual event type. Either component of a type name may specify a complete domain/event type name, a domain/event type name containing the wildcard '*' character, or the special event type name "%ALL".

Parameters

`added`	Sequence of event type names specifying the event types the supplier is adding to the list of event types it plans to supply.
`removed`	Sequence of event type names specifying the event types which the supplier no longer plans to supply.

IT_NotifyComm::GroupPushConsumer Interface

```
interface GroupPushConsumer : GroupNotifyPublish
  {
    oneway void push(in any data);
    oneway void disconnect_push_consumer();
  };
```

The GroupPushConsumer interface supports an operation enabling group consumers to receive unstructured events by the push model. It also defines an operation to disconnect the consumer's endpoint group from its associated proxy supplier. In addition, the GroupPushConsumer interface inherits GroupNotifyPublish which enables a supplier to inform an instance supporting this interface whenever there is a change to the types of events it intends to produce.

Note: An object supporting the GroupPushConsumer interface can receive all events that are supplied to its associated channel. How events supplied to the channel in other forms are internally mapped into an unstructured event for delivery to a GroupPushConsumer is summarized in the *CORBA Notification Service Guide*.

GroupPushConsumer::push()

```
oneway void push(in any data);
```

Receives unstructured events by the push model. The implementation of push() is application specific, and is supplied by application developers.

Parameters

data	A parameter of type CORBA::Any. Upon invocation, this parameter contains an unstructured event being delivered to the group.

GroupPushConsumer::disconnect_push_consumer()

```
oneway void disconnect_push_consumer();
```

Terminates a connection between the target `GroupPushConsumer` and its associated group proxy supplier. The result of this operation is that the target `GroupPushConsumer` releases all resources allocated to support the connection and disposes of the groups object reference. It also disconnects all other members of the target `GroupPushConsumer`'s endpoint group.

IT_NotifyComm::GroupStructuredPushConsumer Interface

```
interface GroupStructuredPushConsumer : GroupNotifyPublish
  {
    oneway void push_structured_event(
                in CosNotification::StructuredEvent notification);
    oneway void disconnect_structured_push_consumer();
  };
```

The GroupStructuredPushConsumer interface supports an operation enabling group consumers to receive structured events by the push model. It also defines an operation to disconnect the push consumer's endpoint group from its associated proxy supplier. In addition, the GroupStructuredPushConsumer interface inherits GroupNotifyPublish which enables a supplier to inform an instance supporting this interface whenever there is a change to the types of events it intends to produce.

Note: An object supporting the GroupStructuredPushConsumer interface can receive all events that were supplied to its associated channel, including events supplied in a form other than a structured event. How events supplied to the channel in other forms are internally mapped into a structured event for delivery to a GroupStructuredPushConsumer is summarized in the *CORBA Notification Service Guide*.

GroupStructuredPushConsumer::push_structured_event();

```
oneway void push_structured_event(
       in CosNotification::StructuredEvent notification);
```

Receives structured events by the push model. The implementation of push_structured_event() is application specific, and is supplied by application developers.

Parameters

`notification`	A parameter of type `StructuredEvent` as defined in `CosNotification`. Upon invocation, this parameter contains a structured event being delivered to the group.

GroupStructuredPushConsumer:: disconnect_structured_push_consumer()

```
oneway void disconnect_structured_push_consumer();
```

Terminates a connection between the target `GroupStructuredPushConsumer` and its associated group proxy supplier. The result of this operation is that the target `GroupStructuredPushConsumer` releases all resources allocated to support the connection and disposes of the groups object reference. This also disconnects all other members of the target `GroupStructuredPushConsumer`'s endpoint group.

IT_NotifyComm::GroupSequencePushConsumer Interface

```
interface GroupSequencePushConsumer : GroupNotifyPublish
  {
    oneway void push_structured_events(
                in CosNotification::EventBatch notifications);

    oneway void disconnect_sequence_push_consumer();
  };
```

The `GroupSequencePushConsumer` interface supports an operation enabling group consumers to receive sequences of structured events using the push model. It also defines an operation to disconnect the consumer's endpoint group from its associated proxy supplier. The `GroupSequencePushConsumer` interface inherits `GroupNotifyPublish` which enabling a supplier to inform an instance supporting this interface whenever there is a change to the types of events it intends to produce.

Note: An object supporting the `GroupSequencePushConsumer` interface can receive all events which were supplied to its associated channel, including events supplied in a form other than a sequence of structured events. How events supplied to the channel in other forms are internally mapped into a sequence of structured events for delivery to a `GroupSequencePushConsumer` is summarized in the *CORBA Notification Service Guide*.

GroupSequencePushConsumer::push_structured_events()

```
oneway void push_structured_events(
       in CosNotification::EventBatch notifications);
```

Receive sequences of structured events by the push model. The implementation of `push_structured_events` is application specific, and is supplied by application developers.

The maximum number of events that are transmitted within a single invocation of this operation, along with the amount of time a supplier of a sequence of structured events accumulates individual events into the sequence before invoking this operation are controlled by QoS property settings as described in the *CORBA Notification Service Guide*.

Parameters

`notifications`	A parameter of type `EventBatch` as defined in `CosNotification`. Upon invocation, this parameter contains a sequence of structured events being delivered to the group.

GroupSequencePushConsumer:: disconnect_sequence_push_consumer()

```
oneway void disconnect_sequence_push_consumer();
```

Terminates a connection between the target `GroupSequencePushConsumer` and its associated group proxy supplier. The result of this operation is that the target `GroupSequencePushConsumer` releases all resources allocated to support the connection and disposes of the groups object reference. This also disconnects all other members of the target `GroupSequencesPushConsumer`'s endpoint group.

CosNotifyFilter Module

The CosNotifyFilterModule specifies the following interfaces to support event filtering:

Filter
FilterFactory
MappingFilter
FilterAdmin

In addition to these interfaces the module specifies several data types and exceptions related to event filtering.

CosNotifyFilter Data Types

CosNotifyFilter::ConstraintID Data Type

```
typedef long ConstraintID;
```

Identifies a constraint.

CosNotifyFilter::ConstraintExp Data Structure

```
struct ConstraintExp
{
  CosNotification::EventTypeSeq event_types;
  string                        constraint_expr;
};
```

Contains a constraint expression and a list of events to check against. The constraint_expr member is a string that conforms to the Trader constraint grammar. For more information on the constraint grammar, see the *CORBA Notification Service Guide*.

CosNotifyFilter::ContsraintIDSeq Data Type

```
typedef <ConstraintID> ContsraintIDSeq;
```

Contains a list of constraint ID.

CosNotifyFilter::ConstraintExpSeq Data Type

```
typedef sequence<ConstraintExp> ContsraintExpSeq;
```

Contains a list of constraint expressions.

CosNotifyFilter::ConstraintInfo Data Structure

```
struct ConstraintInfo
{
  ConstraintExp constraint_expression;
  ConstraintID constraint_id;
}
```

Specifies an instantiated constraint.

CosNotifyFilter::ConstraintInfoSeq Data Type

```
typedef sequence<ConstraintInfo> ConstraintInfoSeq;
```

Contains a list of instantiated constraints.

CosNotifyFilter::FilterID Data Type

```
typedef long FilterID;
```

Identifies an instantiated filter. It is unique to the object to which it is attached.

CosNotifyFilter::FilterIDSeq Data Type

```
typedef sequence<FilterID> FilterIDSeq;
```

Contains a list of FilterIds.

CosNotifyFilter::MappingConstraintPair Data Structure

```
struct MappingConstraintPair
{
  ConstraintExp constraint_expression;
  any           result_to_set;
}
```

Specifies a constraint expression and the value to set if the event matches the constraint expression.

CosNotifyFilter::MappingConstraintPairSeq Data Type

```
typedef sequence<MappingConstraintPair> MappingConstraintPairSeq
```

Contains a list of mapping filter constraint/value pairs.

CosNotifyFilter::MappingConstraintInfo Data Structure

```
struct MappingConstraintInfo
{
  ConstraintExp constraint_expression;
  ConstraintID  constraint_id;
  any           value;
}
```

Specifies a mapping constraint that has been instantiated.

CosNotifyFilter::MappingConstraintInfoSeq Data Types

```
typedef sequence<MappingConstraintInfo> MappingConstraintInfoSeq;
```

Contains a list of instantiated mapping filter constraint/value pairs.

CosNotifyFilter::CallbackID Data Type

```
typedef long CallbackID;
```

Holds an identifier for a callback registered with `attach_callback`.

CosNotifyFilter::CallbackIDSeq Data Type

```
typedef sequence<CallbackID> CallbackIDSeq;
```

Contains a list of callback IDs.

CosNotifyFilter Exceptions

CosNotifyFilter::UnsupportedFilterableData Exception

```
exception UnsupportedFilterableData {};
```

Raised if the input parameter contains data that the `match` operation is not designed to handle. For example, the filterable data contains a field whose name corresponds to a standard event field that has a numeric value, but the actual value associated with this field name within the event is a string.

CosNotifyFilter::InvalidGrammar Exception

```
exception InvalidGrammar {};
```

Raised when creating a filter. If the string passed to the filter factory specifies a grammar that is not supported, the factory will throw `InvalidGrammar`.

Note: Orbix notification service supports the `EXTENDED_TCL` grammar.

CosNotifyFilter::InvalidConstraint Exception

```
exception InvalidConstraint {ConstraintExp constr};
```

Raised during the creation of constraints. If the string specifying the constraint is syntactically incorrect, `InvalidConstraint` is thrown. It returns the invalid constraint.

CosNotifyFilter::ConstraintNotFound Exception

```
exception ConstraintNotFound {ConstraintID id};
```

Raised when a specified constraint ID cannot be resolved to a constraint attached to the target filter object. It returns the ID that cannot be resolved.

CosNotifyFilterFilter::CallbackNotFound Exception

```
exception CallbackNotFound {};
```

Raised when the specified callback ID cannot be resolved to a callback object attached to the target filter object.

CosNotifyFilter::InvalidValue Exception

```
exception InvalidValue {ConstraintExp constr; any value};
```

Raised when the `type_code` of the value associated with the mapping filter constraint does not match the `value_type` of the target mapping filter object.

CosNotifyFilter::FilterNotFound Exception

```
exception FilterNotFound {};
```

Raised if the specified filter ID cannot be resolved to a filter associated with the target object.

CosNotifyFilter::Filter Interface

```
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();
}; // Filter
```

The `Filter` interface defines the behaviors supported by filter objects. These objects encapsulate constraints that are used by the proxies and admins associated with an event channel. The proxies and admins use the constraint definitions to determine which events are forwarded, and which are discarded.

For more information on filters and the constraint language, see the *CORBA Notification Service Guide*.

The `Filter` interface supports operations to manage the constraints associated with a `Filter` instance, along with a read-only attribute to identify the constraint grammar used to evaluate the constraints associated with the instance. In addition, the `Filter` interface supports three variants of the `match` operation which are invoked by a proxy object upon receipt of an event—the specific variant selected depends upon whether the event is received as an `Any` or a structured event—to evaluate the object using the constraints associated with the filter object.

The `Filter` interface also supports operations enabling a client to associate any number of callbacks with the target filter object. The callbacks are notified each time there is a change to the list of event types the filer forwards through the event channel. Operations are also defined to support administration of this callback list by unique identifier.

Filter::constraint_grammar

```
readonly attribute string constraint_grammar;
```

`constraint_grammar` is a readonly attribute specifiying the particular grammar used to parse the constraint expressions encapsulated by the target filter. The value of this attribute is set upon creation of a filter object.

A filter's constraints must be expressed using a particular constraint grammar because its member `match` operations must be able to parse the constraints to determine whether or not a particular event satisfies one of them.

Orbix supports an implementation of the `Filter` interface which supports the default constraint grammar described in the *CORBA Notification Service Guide*. The `constraint_grammar` attribute is set to the value `EXTENDED_TCL` when the target filter object supports this default grammar.

Other implementations can provide additional implementations of the `Filter` interface that support different constraint grammars, and thus the `constraint_grammar` attribute must be set to a different value upon creation of such a filter object.

Filter::add_constraints()

```
ConstraintInfoSeq add_constraints(
                    in ConstraintExpSeq constraint_list)
    raises (InvalidConstraint);
```

Associates one or more new constraints with the target filter object. Upon successful processing of all input constraint expressions, `add_constraints()` returns a `ConstraintInfoSeq` containing all of the constraints and the identifiers assigned to them by the filter.

If one or more of the constraints passed into `add_constraints()` is invalid, none of the constraints are added to the target filter.

Note: Once `add_constraints()` is invoked by a client, the target filter is temporarily disabled from usage by any proxy or admin it may be associated with. Upon completion of the operation, the target filter is re-enabled and can once again be used by associated proxies and admins to make event forwarding decisions.

Parameters

`constraint_list`	A sequence of constraint data structures using the constraint grammar supported by the target object.

Exceptions

If any of the constraints in the input sequence is not a valid expression within the supported constraint grammar, the `InvalidConstraint` exception is raised. This exception contains as data the specific constraint expression that was determined to be invalid.

Filter::modify_constraints()

```
void modify_constraints (
        in ConstraintIDSeq del_list,
        in ConstraintInfoSeq modify_list)
    raises (InvalidConstraint, ConstraintNotFound);
```

Modifies the constraints associated with the target filter object. This operation can be used both to remove constraints currently associated with the target filter, and to modify the constraint expressions of constraints currently associated with the filter.

If an exception is raised during the operation, no changes are made to the filter's constraints.

Note: Once `modify_constraints` is invoked by a client, the target filter is temporarily disabled from use by any proxy or admin. Upon completion of the operation, the target filter is re-enabled and can once again be used by associated proxies and admins to make event forwarding decisions.

Parameters

`del_list`	A sequence of numeric identifiers each of which should be associated with one of the constraints currently encapsulated by the target filter object.
`modify_list`	A sequence containing constraint structures and an associated numeric value. The numeric value in each element of the sequence is the unique identifier of one of the constraints encapsulated by the target filter.

Exceptions

`ConstraintNotFound`	Raised if any of the numeric values in either input sequences does not correspond to the unique identifier associated with any constraint encapsulated by the target filter. This exception contains the specific identifier that did not correspond to the identifier of some constraint encapsulated by the target filter.

InvalidConstraint	Raised if any of the constraint expressions supplied in the second input sequence is not a valid expression in terms of the constraint grammar supported by the target object. This exception contains the specific constraint that was determined to be invalid.

Filter::get_constraints()

```
ConstraintInfoSeq get_constraints(in ConstraintIDSeq id_list)
    raises (ConstraintNotFound);
```

Returns a sequence of data structures containing the input identifiers along with their associated constraint.

Parameters

id_list	A sequence of numeric values corresponding to the unique identifiers of constraints encapsulated by the target object.

Exceptions

ConstraintNotFound	One of the input values does not correspond to the identifier of some encapsulated constraint. The exception contains that input value.

Filter::get_all_constraints()

```
ConstraintInfoSeq get_all_constraints();
```

Returns all of the constraints currently encapsulated by the target filter object.

Filter::remove_all_constraints()

```
void remove_all_constraints();
```

Removes all of the constraints currently encapsulated by the target filter. Upon completion, the target filter still exists but no constraints are associated with it.

Filter::destroy()

```
void destroy();
```

Destroys the target filter and invalidates its object reference.

Filter::match()

```
boolean match (in any filterable_data)
    raises (UnsupportedFilterableData);
```

Evaluates the filter constraints associated with the target filter against an event supplied to the channel in the form of a CORBA::Any. The operation returns TRUE if the input event satisfies one of the filter constraints, and FALSE otherwise.

The act of determining whether or not a given event passes a given filter constraint is specific to the type of grammar in which the filter constraint is specified.

Parameters

`filterable_data` A CORBA::Any which contains an event to be evaluated.

Exceptions

UnsupportedFilterableData	The input parameter contains data that the match operation is not designed to handle.

Filter::match_structured()

```
boolean match_structured(
          in CosNotification::StructuredEvent filterable_data )
    raises (UnsupportedFilterableData);
```

Evaluates the filter constraints associated with the target filter against a structured event. The operation returns TRUE if the input event satisfies one of the filter constraints, and FALSE otherwise.

The act of determining whether or not a given event passes a given filter constraint is specific to the type of grammar in which the filter constraint is specified.

Parameters

`filterable_data`	A `CosNotification::StructuredEvent` containing an event to be evaluated,

Exceptions

`UnsupportedFilterableData`	The input parameter contains data that the `match` operation is not designed to handle.

Filter::attach_callback()

```
CallbackID attach_callback (
            in CosNotifyComm::NotifySubscribe callback);
```

Associates an object supporting the `CosNotifyComm::NotifySubscribe` interface with the target filter. This operation returns a numeric value assigned to this callback that is unique to all such callbacks currently associated with the target filter.

After this operation has been successfully invoked on a filter, the filter invokes the `subscription_change()` method of all its associated callbacks each time the set of constraints associated with the filter is modified. This process informs suppliers in the filter's callback list of the change in the set of event types to which the filter's clients subscribe. With this information, suppliers can make intelligent decisions about which event types to produce.

Parameters

`callback`	The reference to an object supporting the `CosNotifyComm::NotifySubscribe` interface.

Filter::detach_callback()

```
void detach_callback(in CallbackID callback)
raises (CallbackNotFound);
```

Removes a callback object from the filter's callback list. Subsequent changes to the event type subscription list encapsulated by the target filter are no longer propagated to that callback object.

Parameters

`callback`	A unique identifiers associated with one of the callback objects attached to the target filter.

Exceptions

`CallbackNotFound`	The input value does not correspond to the unique identifier of a callback object currently attached to the target filter object.

Filter::get_callbacks()

```
CallbackIDSeq get_callbacks();
```

Returns all the unique identifiers for the callback objects attached to the target filter.

CosNotifyFilter::MappingFilter Interface

```
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);
}; // MappingFilter
```

The `MappingFilter` interface defines the behaviors of objects that encapsulate a sequence of constraint-value pairs (see the description of the Default Filter Constraint Language in the *CORBA Notification Service Guide*). These constraint-value pairs are used to evaluate events and adjust their lifetime/priority values according to the result. An object supporting the `MappingFilter` interface can effect either an events lifetime property or its priority property, but not both.

The `MappingFilter` interface supports the operations required to manage the constraint-value pairs associated with an object instance supporting the interface. In addition, the `MappingFilter` interface supports a read-only attribute that identifies the constraint grammar used to parse the constraints encapsulated by this object. The `MappingFilter` interface supports a read-only attribute that identifies the typecode associated with the datatype of the specific property value it is intended to affect. It also supports another read-only attribute which holds the default value which is returned as the result of a match operation in cases when the event in question is found to satisfy none of the constraints encapsulated by the mapping filter. Lastly, the `MappingFilter` interface supports three variants of the operation which are invoked by an associated proxy object upon receipt of an event, to determine how the property of the event which the target mapping filter object was designed to affect should be modified.

MappingFilter::constraint_grammar

```
readonly attribute string constraint_grammar;
```

Identifies the grammar used to parse the constraint expressions encapsulated by the target mapping filter. The value of this attribute is set upon creation of a mapping filter.

A filter object's constraints must be expressed using a particular constraint grammar because its member `match` operations must be able to parse the constraints to determine whether or not a particular event satisfies one of them.

Orbix supports an implementation of the MappingFilter object which supports the default constraint grammar described in the *CORBA Notification Service Guide*. constraint_grammar is set to the value EXTENDED_TCL when the target mapping filter supports this default grammar.

Users may provide additional implementations of the MappingFilter interface which support different constraint grammars, and thus set the constraint_grammar attribute to a different value when creating such a mapping filter.

MappingFilter::value_type

```
readonly attribute CORBA::TypeCode value_type;
```

Identifies the datatype of the property value that the target mapping filter is designed to affect. Note that the factory creation operation for mapping filters accepts as an input parameter the default_value to associate with the mapping filter instance. This default_value is a CORBA::Any. Upon creation of a mapping filter, the typecode associated with the default_value is abstracted from the CORBA::Any, and its value is assigned to this attribute.

MappingFilter::default_value

```
readonly attribute any default_value;
```

The value returned as the result of any match operation during which the input event does not satisfy any of the constraints encapsulated by the mapping filter. The value of this attribute is set upon creation of a mapping filter object instance.

MappingFilter::add_mapping_constraints()

```
MappingConstraintInfoSeq add_mapping_constraints (
        in MappingConstraintPairSeq pair_list)
    raises (InvalidConstraint, InvalidValue);
```

Returns a sequence of structures which contain one of the input constraint expressions, its corresponding value, and the unique identifier assigned to this constraint-value pair by the target filter.

If one or more of the constraints passed into add_mapping_constraints() is invalid, none of the constraints are added to the target mapping filter.

Note: Once add_mapping_constraints is invoked by a client, the target filter is temporarily disabled from use by any proxy it may be associated with. Upon completion of the operation, the target filter is re-enabled and can once again be used by associated proxies to make event property mapping decisions.

Parameters

pair_list	A sequence of constraint-value pairs. Each constraint in this sequence must be expressed in the constraint grammar supported by the target object, and each associated value must be of the data type indicated by the value_type attribute of the target object.

Exceptions

InvalidConstraint	Raised if any of the constraint expressions in the input sequence is not a valid expression. This exception contains the constraint that was determined to be invalid.
InvalidValue	Raised if any of the values supplied in the input sequence are not of the same datatype as that indicated by the target object's value_type attribute. This exception contains the invalid value and its corresponding constraint.

MappingFilter::modify_mapping_constraints()

```
void modify_mapping_constraints (
        in ConstraintIDSeq del_list,
        in MappingConstraintInfoSeq modify_list)
     raises(InvalidConstraint,
            InvalidValue,
            ConstraintNotFound);
```

Modifies the constraint-value pairs associated with the target mapping filter. This operation can remove constraint-value pairs currently associated with the target mapping filter, and to modify the constraints and/or values of constraint-value pairs currently associated with the target mapping filter.

If an exception is raised during the operation, no changes are made to the filter's constraints.

Note: Once `modify_mapping_constraints()` is invoked by a client, the target mapping filter is temporarily disabled from use by any proxy it may be associated with. Upon completion of the operation, the target mapping filter is re-enabled and can be used by associated proxies to make event property mapping decisions.

Parameters

`del_list`	A sequence of unique identifiers associated with one of the constraint-value pairs currently encapsulated by the target mapping filter. If all input values are valid, the specific constraint-value pairs identified by the values contained in this parameter are deleted from the mapping filter's list of constraint-value-pairs.
`modify_list`	A sequence of structures containing a constraint structure, an Any value, and a numeric identifier. The numeric identifier of each element is the unique identifier associated with one of the constraint-value pairs currently encapsulated by the target filter object. The constraint-value pairs identified are modified to the values specified in the input list.

Exceptions

`ConstraintNotFound`	Raised if any of the identifiers in either of the input sequences does not correspond to the unique identifier associated with a constraint-value pair encapsulated by the target mapping filter. This exception contains the identifier which did not correspond to a constraint-value pair encapsulated by the target object.

InvalidConstraint	Raised if any of the constraint expressions supplied in an element of the second input sequence is not valid. This exception contains the constraint that was determined to be invalid.
InvalidValue	Raised if any of the values in the second input sequence is not of the same datatype as that indicated by the mapping filter's value_type attribute. This exception contains the invalid value and its corresponding constraint expression.

MappingFilter::get_mapping_constraints()

```
MappingConstraintInfoSeq get_mapping_constraints (
            in ConstraintIDSeq id_list)
        raises (ConstraintNotFound);
```

Returns a sequence of constraint-value pairs associated with the target mapping filter.

Parameters

id_list	A sequence of unique identifiers for constraint-value pairs encapsulated by the target object.

Exceptions

ConstraintNotFound	One of the input values does not correspond to the identifier of an encapsulated constraint-value pair. The exception contains the identifier that did not correspond to a constraint-value pair.

MappingFilter::get_all_mapping_constraints()

```
MappingConstraintInfoSeq get_all_mapping_constraints();
```

Returns all of the constraint-value pairs encapsulated by the target mapping filter.

MappingFilter::remove_all_mapping_constraints

```
void remove_all_mapping_constraints();
```

Removes all of the constraint-value pairs currently encapsulated by the target mapping filter. Upon completion, the target mapping filter still exists but has no constraint-value pairs associated with it.

MappingFilter::destroy()

```
void destroy();
```

Destroys the target mapping filter, and invalidates its object reference.

MappingFilter::match()

```
boolean match(in any filterable_data, out any result_to_set)
    raises (UnsupportedFilterableData);
```

Determines how to modify some property value of an event in the form of a CORBA::Any.

The target mapping filter begins applying the its constraints according to each constraint's associated value, starting with the constraint with the best associated value for the specific property the mapping filter is designed to affect (for example, the highest priority, the longest lifetime, and so on), and ending with the constraint with the worst associated value.

Upon encountering a constraint which the event matches, the operation sets result_to_set to the value associated with the matched constraint, and returns with a value of TRUE. If the event does not satisfy any of the target mapping filter's constraints, the operation sets result_to_set to the value of the target mapping filter's default_value attribute and returns with a value of FALSE.

The act of determining whether or not a given event passes a given filter constraint is specific to the grammar used to parse the filter constraints.

Parameters

filterable_data An Any containing the event being evaluated

result_to_set	An Any containing the value and the property name to set when an event evaluates to TRUE.

Exceptions

UnsupportedFilterableData	The input parameter contains data that the match operation is not designed to handle.

MappingFilter::match_structured()

```
boolean match_structured (
        in CosNotification::StructuredEvent filterable_data,
        out any result_to_set)
    raises (UnsupportedFilterableData);
```

Determines how to modify some property value of a structured event.

The target mapping filter begins applying the its constraints according to each constraints associated value, starting with the constraint with the best associated value for the specific property the mapping filter is designed to affect (for example, the highest priority, the longest lifetime, and so on), and ending with the constraint with the worst associated value.

Upon encountering a constraint which the event matches, the operation sets result_to_set to the value associated with the matched constraint, and returns with a value of TRUE. If the event does not satisfy any of the target mapping filter's constraints, the operation sets result_to_set to the value of the target mapping filter's default_value attribute and returns with a value of FALSE.

The act of determining whether or not a given event passes a given filter constraint is specific to the grammar used to parse the filter constraints.

Parameters

filterable_data	A CosNotification::StructuredEvent containing the event being evaluated.
result_to_set	An Any containing the value and the property name to set when an event evaluates to TRUE.

Exceptions

`UnsupportedFilterableDat`	The input parameter contains data that `match_structured()` is not designed to handle.

CosNotifyFilter::FilterFactory Interface

```
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);
};
```

The `FilterFactory` interface defines operations for creating filter.

FilterFactory::create_filter()

```
Filter create_filter (in string constraint_grammar)
    raises (InvalidGrammar);
```

Creates a forwarding filter object and returns a reference to the new filter.

Parameters

`constraint_grammar` A string identifying the grammar used to parse constraints associated with this filter.

Exceptions

`InvalidGrammar` The client invoking this operation supplied the name of a grammar that is not supported by any forwarding filter implementation this factory is capable of creating.

FilterFactory::create_mapping_filter()

```
MappingFilter create_mapping_filter (
        in string constraint_grammar,
```

```
        in any default_value)
    raises(InvalidGrammar);
```

Creates a mapping filter object and returns a reference to the new mapping filter.

Parameters

`constraint_grammar`	A string parameter identifying the grammar used to parse constraints associated with this filter.
`default_value`	An `Any` specifying the `default_value` of the new mapping filter.

Exceptions

`InvalidGrammar`	The client invoking this operation supplied the name of a grammar that is not supported by any mapping filter implementation this factory is capable of creating.

CosNotifyFilter::FilterAdmin Interface

```
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();
};
```

The `FilterAdmin` interface defines operations enabling an object supporting this interface to manage a list of filters, each of which supports the `Filter` interface. This interface is an abstract interface which is inherited by all of the proxy and admin interfaces defined by the notification service.

FilterAdmin::add_filter()

```
FilterID add_filter(in Filter new_filter);
```

Appends a filter to the list of filters associated with the target object upon which the operation was invoked and returns an identifier for the filter.

Parameters

`new_filter`	A reference to an object supporting the `Filter` interface.

FilterAdmin::remove_filter()

```
void remove_filter(in FilterID filter)
    raises ( FilterNotFound );
```

Removes the specified filter from the target object's list of filters.

Parameters

`filter`	A numeric value identifying a filter associated with the target object

Exceptions

`FilterNotFound`	The identifier does not correspond to a filter associated with the target object.

FilterAdmin::get_filter()

```
Filter get_filter (in FilterID filter)
    raises ( FilterNotFound );
```

Returns the object reference to the specified filter.

Parameters

`filter`	A numeric value identifying a filter associated with the target object

Exceptions

`FilterNotFound`	The identifier does not correspond to a filter associated with the target object.

FilterAdmin::get_all_filters()

```
FilterIDSeq get_all_filters();
```

Returns the list of unique identifiers corresponding to all of the filters associated with the target object.

FilterAdmin::remove_all_filters()

```
void remove_all_filters();
```

Removes all filters from the filter list of the target object.

CosNotification Module

The `CosNotification` module defines the structured event data type, and a data type used for transmitting sequences of structured events. In addition, this module provides constant declarations for each of the standard quality of service (QoS) and administrative properties supported by the notification service. Some properties also have associated constant declarations to indicate their possible settings. Finally, administrative interfaces are defined for managing sets of QoS and administrative properties.

CosNotification Data Types

CosNotification::StructuredEvent Data Structure

```
//IDL
struct EventType {
    string domain_name;
    string type_name;
};

struct FixedEventHeader {
    EventType 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
```

The StructuredEvent data structure defines the fields which make up a structured event. A detailed description of structured events is provided in the *CORBA Notification Service Guide*.

CosNotification::EventTypeSeq Type

```
//IDL
struct EventType {
    string domain_name;
    string type_name;
};
typedef sequence <EventType> EventTypeSeq
```

CosNotification::EventBatch Type

The CosNotification module defines the EventBatch data type as a sequence of structured events. The CosNotifyComm module defines interfaces supporting the transmission and receipt the EventBatch data type.

QoS and Administrative Constant Declarations

The CosNotification module declares several constants related to QoS properties, and the administrative properties of event channels.

```
// IDL in CosNotification module
const string EventReliability = "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

const string StopTime = "StopTime";
// StopTime takes a value of type TimeBase::UtcT

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 StartTimeSupported = "StartTimeSupported";
// StartTimeSupported takes on a boolean value

const string StopTimeSupported = "StopTimeSupported";
// StopTimeSupported takes on a boolean value

const string MaxEventsPerConsumer = "MaxEventsPerConsumer";
// MaxEventsPerConsumer takes on a value of type long
```

QoS and Admin Data Types

The `CosNotification` module defines several data types related to QoS properties, and the administrative properties of event channels.

CosNotification::PropertyName Type

```
typedef string PropertyName;
```

PropertyName is a string holding the name of a QoS or an Admin property.

CosNotification::PropertyValue Type

```
typedef any PropertyValue;
```

PropertyValue is an any holding the setting of QoS or Admin properties.

CosNotification::PropertySeq Type

```
//IDL in CosNotification module
struct Property
  {
  PropertyName name;
  PropertyValue value;
  };
typedef sequence <Property> PropertySeq;
```

PropertySeq is a set of name-value pairs that encapsulate QoS or Admin properties and their values.

Members

name	A string identifying the QoS or Admin property.
value	An Any containing the setting of the QoS or Admin property.

CosNotification::QoSProperties Type

```
typedef PropertySeq QoSProperties;
```

QoSProperties is a name-value pair of PropertySeq used to specify QoS properties.

CosNotification::AdminProperties Type

```
typedef PropertySeq AdminProperties;
```

AdminProperties is a name-value pair of PropertySeq used to specify Admin properties.

CosNotification::QoSError_code Enum

```
enum QoSError_code
{
    UNSUPPORTED_PROPERTY,
    UNAVAILABLE_PROPERTY,
    UNSUPPORTED_VALUE,
    UNAVAILABLE_VALUE,
    BAD_PROPERTY,
    BAD_TYPE,
    BAD_VALUE
};
```

QoSError_code specifies the error codes for UnsupportedQoS and UnsupportedAdmin exceptions. The return codes are:

UNSUPPORTED_PROPERTY	Orbix does not support the property for this type of object
UNAVAILABLE_PROPERTY	This property cannot be combined with existing QoS properties.
UNSUPPORTED_VALUE	The value specified for this property is invalid for the target object.
UNAVAILABLE_VALUE	The value specified for this property is invalid in the context of other QoS properties currently in force.
BAD_PROPERTY	The property name is unknown.
BAD_TYPE	The type supplied for the value of this property is incorrect.
BAD_VALUE	The value specified for this property is illegal.

CosNotification::PropertyErrorSeq Type

```
// IDL from CosNotification module
  struct PropertyRange
    {
    PropertyValue low_val;
    PropertyValue high_val;
    };

  struct PropertyError
    {
    QoSError_code code;
    PropertyName name;
    PropertyRange available_range;
    };
  typedef sequence <PropertyError> PropertyErrorSeq;
```

A `PropertyErrorSeq` is returned when `UnsupportedQoS` or `UnsupportedAdmin` is raised. It specifies a sequence containing the reason for the exception, the property that caused it, and a range of valid settings for the property.

CosNotification::NamedPropertyRangeSeq Type

```
struct NamedPropertyRange
 {
 PropertyName name;
 PropertyRange range;
 };
typedef sequence <NamedPropertyRange> NamedPropertyRangeSeq;
```

Specifies a range of values for the named property.

QoS and Admin Exceptions

The `CosNotification` module defines two exceptions related to QoS properties, and the administrative properties of event channels.

CosNotification::UnsupprtedQoS

```
exception UnsupportedQoS { PropertyErrorSeq qos_err; };
```

Raised when setting QoS properties on notification channel objects, or when validating QoS properties. It returns with a `PropertyErrorSeq` specifying the reason for the exception, which property was invalid, and a list of valid settings for the QoS property.

CosNotification::UnsupportedAdmin

```
exception UnsupportedAdmin { PropertyErrorSeq admin_err; };
```

Raised when setting Admin properties on notification channels. It returns with a `PropertyErrorSeq` specifying the reason for the exception, which property was invalid, and a list of valid settings for the property.

CosNotification::QoSAdmin Interface

```
//IDL
interface QoSAdmin {
    QoSProperties get_qos();
    void set_qos ( in QoSProperties qos)
        raises ( UnsupportedQoS );
    void validate_qos (
        in QoSProperites required_qos,
        out NamedPropertyRangeSeq available_qos )
        raises ( UnsupportedQoS );
```

The QoSAdmin interface defines operations enabling clients to manage the values of QoS properties. It also defines an operation to 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. QoSAdmin is an abstract interface which is inherited by the proxy, admin, and event channel interfaces defined in the CosNotifyChannelAdmin module.

QoSAdmin::get_qos()

```
QoSProperites get_qos();
```

Returns a sequence of name-value pairs encapsulating the current quality of service settings for the target object (which could be an event channel, admin, or proxy object).

QoSAdmin::set_qos()

```
void set_qos ( in QoSProperites qos)
    raises ( UnsupportedQoS );
```

Sets the specified QoS properties on the target object (which could be an event channel, admin, or proxy object).

Parameters

qos	A sequence of name-value pairs encapsulating quality of service property settings

Exceptions

UnsupportedQoS	The implementation of the target object is incapable of supporting some of the requested quality of service settings, or one of the requested settings are in conflict with a QoS property defined at a higher level of the object hierarchy.

QoSAdmin::validate_qos()

```
void validate_qos (
    in QoSProperites required_qos,
    out NamedPropertyRangeSeq available_qos )
    raises ( UnsupportedQoS );
```

Enables a client to discover if the target object is capable of supporting a set of QoS settings. If all requested QoS property value settings can be satisfied by the target object, the operation returns successfully (without actually setting the QoS properties on the target object).

Parameters

`required_qos`	A sequence of QoS property name-value pairs specifying a set of QoS settings.
`available_qos`	An output parameter that contains a sequence of `NamedPropertyRange`. Each element in this sequence includes the name of a an additional QoS property supported by the target object which 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.

Exceptions

UnsupportedQoS	Raised if If any of the requested settings cannot be satisfied by the target object.

CosNotification:: AdminPropertiesAdmin Interface

```
//IDL
interface AdminPropertiesAdmin {
    AdminProperites get_admin();
    void set_admin (in AdminProperites admin)
        raises ( UnsupportedAdmin);
};
```

The AdminPropertiesAdmin interface defines operations enabling clients to manage the values of administrative properties. This interface is an abstract interface which is inherited by the Event Channel interfaces defined in the CosNotifyChannelAdmin module.

AdminPropertiesAdmin::get_admin()

```
AdminProperites get_admin();
```

Returns a sequence of name-value pairs encapsulating the current administrative settings for the target channel.

AdminPropertiesAdmin::set_admin()

```
void set_admin (in AdminProperites admin)
    raises ( UnsupportedAdmin);
```

Sets the specified administrative properties on the target object.

Parameters

admin	A sequence of name-value pairs encapsulating administrative property settings.

Exceptions

UnsupportedAdmin	Raised if If any of the requested settings cannot be satisfied by the target object.

JMS-Notification Bridge Service

JMS-Notification Bridge Service Overview

This part lists the IDL definitions in the Application Server Platform's JMS-Notification bridge service's modules.

The bridge service is composed of the following IONA specific interfaces, listed in `messaging_bridge.idl`, `messaging_bridge_admin.idl`, and `notify_bridge.idl`:

- "IT_MessagingBridge::Endpoint Interface"
- "IT_MessagingBridge::SinkEndpoint Interface"
- "IT_MessagingBridge::SourceEndpoint Interface"
- "IT_MessagingBridge::EndpointAdmin Interface"
- "IT_MessagingBridgeAdmin::Bridge Interface"
- "IT_MessagingBridgeAdmin::BridgeAdmin Interface"
- "IT_NotifyBridge::SinkEndpoint Interface"

These interfaces support the bridging of enterprise messaging services as described in the *Applicaiton Server Platform Enterprise Messaging Guide*.

IT_MessagingBridge Module

IT_MessagingBridge defines the data types, exceptions, and interfaces used to establish and manage the endpoints of a bridge. The following interfaces are defined in IT_MessagingBridge:

- IT_MessagingBridge::Endpoint Interface
- IT_MessagingBridge::SinkEndpoint Interface
- IT_MessagingBridge::SourceEndpoint Interface
- IT_MessagingBridge::EndpointAdmin Interface

IT_MessagingBridge Data Types

IT_MessagingBridge::BridgeName

```
typedef string BridgeName;
```

BridgeName specifies the unique identifier of a bridge.

IT_MessagingBridge::BridgeNameSeq

```
typedef sequence<BridgeName> BridgeNameSeq;
```

BridgeNameSeq contains a list of bridge names and is the type returned by IT_MessagingBridgeAdmin::BridgeAdmin::list_all_bridges().

IT_MessagingBridge::EndpointName

```
typedef string EndpointName;
```

EndpointName uniquely identifies the name of the messaging object with which the endpoint is associated. For example, the EndpointName could be the name of a notification channel, a JMS topis, or a JMS queue.

IT_MessagingBridge::EndpointType

```
typedef short EndpointType;

const EndpointType JMS_TOPIC = 1;
const EndpointType JMS_QUEUE = 2;
const EndpointType NOTIFY_CHANNEL = 3;
```

EndpointType specifys what type of messaging object to which the endpoint is going to connect. It can take one of three constant values:

Table 23: *EndpointTypes and the associated messaging objects*

EndpointType	Messaging Object
JMS_TOPIC	JMS Topic
JMS_QUEUE	JMS Queue
NOTIFY_CHANNEL	Notification Channel

IT_MessagingBridge::EndpointTypeSeq

```
typedef sequeunce<EndpointType> EndpointTypeSeq;
```

EndpointTypeSeq specifies a list of endpoint types.

IT_MessagingBridge::EndpointAdminName

```
typedef string EndpointAdminName;
```

EndpointAdminName specifies the unique identifier assigned to an endpoint admin object.

IT_MessagingBridge::InvalidEndpointCode

```
typedef short InvalidEndpointCode;

const InvalidEndpointCode INVALID_TYPE = 1;
const InvalidEndpointCode INVALID_NAME = 2;
```

```
const InvalidEndpointCode UNSUPPORTED_TYPE = 3;
const InvalidEndpointCode INCOMPATIBLE_TYPE = 4;
const InvalidEndpointCode SAME_AS_PEER = 5;
const InvalidEndpointCode DOES_NOT_EXIST = 6;
```

`InvalidEndpointCode` specifies the return code of the `InvalidEndpoint` exception.

IT_MessagingBridge Exceptions

IT_MessaingBridge::InvalidEndpoint

```
exception InvalidEndpoint {InvalidEndpointCode code;};
```

`InvalidEndpoint` is raised when an endpoint is incorrectly specified. Its return code specifies the reason the endpoint is invalid. The return code will be one of the following:

Table 24: *InvalidEndpoint return codes and their explanation*

InvalidEndpointCode	Explanation
`INVALID_TYPE`	The `EndpointType` was not recognized.
`INVALID_NAME`	The `EndpointName` is not valid for the specified `EndpointType`.
`UNSUPPORYED_TYPE`	The `EndpointAdmin` does not support the specified type of endpoint.
`INCOMPATIBLE_TYPE`	The `EndpointType` of the endpoints being connected are incompatible. For example a `JMS_TOPIC` cannot be connected to a `JMS_QUEUE`.
`SAME_AS_PEER`	The `EndpointType` of the endpoint being connected to is the same as the current endpoint.
`DOES_NOT_EXIST`	The endpoint specified by `EndpointName` does not exist.

IT_MessagingBridge::EndpointAlreadyConnected

```
exception EndpointAlreadyConnected {};
```

`EndpointAlreadyConnected` is raised when an attempt is made to connect an endpoint that is already connected to a peer.

IT_MessagingBridge::BridgeNameNotFound

```
exception BridgeNameNotFound {};
```

`BridgeNameNotFound` is raised when the bridge with the specified name is not found.

IT_MessagingBridge::BridgeNameAlreadyExists

```
exception BridgeNameAlreadyExists {};
```

`BridgeNameAlreadyExists` is raised when an attempt to create a bridge with a name already in use is made.

IT_MessagingBridge::Endpoint Interface

```
interface Endpoint
{
  readonly attribute BridgeName bridge_name;
  readonly attribute EndpointType type;
  readonly attribute EndpointName name;
  readonly attribute EndpointAdmin admin;
  readonly attribute Endpoint peer;
  readonly attribute boolean connected;

  void connect(in Endpoint peer)
  raises (InvalidEndpoint, EndpointAlreadyConnected);

  void destroy();
};
```

`Endpoint` is a generic interface used to specify a bridge endpoint. This is recomended interface for developers to use when working with bridge endpoints. Defines the attributes used to specify the type of endpoint, the bridge is is associated with, and if the endpoint is actively in use by a bridge. The interface also specifies an operation for connecting an endpoint to a peer endpoint and an operation for releasing the resources used by an endpoint. In general, the connection of endpoints to peers and the destructions of specific endpoints is handled by the bridge service when a bridge is created or detoryed.

Endpoint::bridge_name

```
readonly attribute BridgeName bridge_name;
```

`bridge_name` specifies the name of the bridge with which the bridge is associated.

Endpoint::type

```
readonly attribute EndpointType type;
```

type specifies the type of messaging object to which the endpoint is connected.

Endpoint::name

```
readonly attribute EndpointName name;
```

name specifies the unique identifier of the endpoint.

Endpoint::admin

```
readonly attribute EndpointAdmin admin;
```

admin is a reference to the EndpointAdmin associated with the endpoint.

Endpoint::peer

```
readonly attribute Endpoint peer;
```

peer is a reference to the endpoint on the other end of the bridge. If the endpoint is not connected to a peer, this reference is nul.

Endpoint::connected

```
readonly attribute boolean connected;
```

connected specifies if the endpoint is actively connected to a peer endpoint.

Endpoint::connect()

```
void connect(in Endpoint peer)
raises (InvalidEndpoint, EndpointAlreadyConnected);
```

connect() creates a connection between the current endpoint and the endpoint passed into the operation. This operation is called by the bridge service when a bridge is create.

Parameters

`peer`	Specifies the endpoint that is being connected to.

Exceptions

`InvalidEndpoint`	The specified endpoint is invalid. The return code provides the details explaining the reason.
`EndpointAlreadyConnected`	One of the endpoints is already connected to a peer endpoint.

Endpoint::destroy()

```
void destroy();
```

Destroys the endpoint and releases all resources used to support it.

IT_MessagingBridge::SinkEndpoint Interface

```
interface SinkEndpoint : Endpoint
{
};
```

SinkEndpoint is a specialization of the generic IT_MessagingBridge::Endpoint interface. Ii is used to specify an endpoint that recieves messages from the bridge and foward the messages onto the recieving service. It defines no specific operations.

IT_MessagingBridge::SourceEndpoint Interface

```
interface SourceEndpoint : Endpoint
{
  void start();

  void suspend();

  void stop();
};
```

`SourceEndpoint` is a specialization of the generic `IT_MessagingBridge::Endpoint` interface. It is used to specify an endpoint that takes messages from the forwarding service and passes the messages into the bridge. It defines three operations for controling the flow of messages through the endpoint.

SourceEndpoint::start()

```
void start();
```

`start()` begins the flow of messages to the bridge.

SourceEndpoint::suspend()

```
void suspend();
```

`susspend()` stops the flow of messages to the bridge, but causes the endpoint to queue any incomming messages for delivery. Once the flow of messages is restarted, the queued messages will be pass to the bridge.

SourceEndpoint::stop()

```
void stop();
```

`stop()` completely stops the flow of messages to the bridge.

IT_MessagingBridge::EndpointAdmin Interface

```
interface EndpointAdmin
{
  readonly attribute EndpointAdminName name;
  readonly attribute EndpointTypeSeq supported_types;

  SinkEndpoint create_sink_endpoint(in BridgeName   bridge_name,
                                    in EndpointType type,
                                    in EndpointName name)
  raises (InvalidEndpoint, BridgeNameAlreadyExists);

  SourceEndpoint create_source_endpoint(in BridgeName bridge_name,
                                        in EndpointType type,
                                        in EndpointName name)
  raises (InvalidEndpoint, BridgeNameAlreadyExists);

  SinkEndpoint get_sink_endpoint(in BridgeName bridge_name)
  raises (BridgeNameNotFound);

  SourceEndpoint get_source_endpoint(in BridgeName bridge_name)
  raises (BridgeNameNotFound);

  BridgeNameSeq get_all_sink_endpoints();

  BridgeNameSeq get_all_source_endpoints();
};
```

`EndpointAdmin` defines the factory operations to create and discover endpoints. There is one `EndpointAdmin` object for each messaging service that can participate in bridging.

EndpointAdmin::name

```
readonly attribute EndpointAdminName name;
```

`name` specifies the unique identifier of the endpoint admin object.

EndpointAdmin::supported_types

```
readonly attribute EndpointTypeSeq supported_types;
```

supported_types specifies the types of endpoint that the admin object can support. For example, the EndpointAdmin for JMS can support endpoints of type JMS_TOPIC and JMS_QUEUE.

EndpointAdmin::create_sink_endpoint()

```
SinkEndpoint create_sink_endpoint(in BridgeName   bridge_name,
                                  in EndpointType type,
                                  in EndpointName name)
raises (InvalidEndpoint, BridgeNameAlreadyExists);
```

create_sink_endpoint() creates a new SinkEndpoint of the specified type and associates it with the specified bridge name.

Parameters

bridge_name	The name of the bridge with which to associate the endpoint.
type	The EndpointType of the new endpoint.
name	The unique identifier to use for the endpoint.

Exceptions

InvalidEndpoint	The type or the name specified are incorrect. The return code will contain the details.
BridgeNameAlreadyExists	

EndpointAdmin::create_source_endpoint()

```
SourceEndpoint create_source_endpoint(in BridgeName bridge_name,
                                      in EndpointType type,
                                      in EndpointName name)
raises (InvalidEndpoint, BridgeNameAlreadyExists);
```

create_source_endpoint() creates a new SourceEndpoint of the specified type and associates it with the specified bridge name.

Parameters

`bridge_name`	The name of the bridge with which to associate the endpoint.
`type`	The `EndpointType` of the new endpoint.
`name`	The unique identifier to use for the endpoint.

Exceptions

`InvalidEndpoint`	The `type` or the `name` specified are incorrect. The return code will contain the details.
`BridgeNameAlreadyExists`	

EndpointAdmin::get_sink_endpoint()

```
SinkEndpoint get_sink_endpoint(in BridgeName bridge_name)
raises (BridgeNameNotFound);
```

`get_sink_endpoint()` returns a reference to the sink endpoint of the specified bridge.

Parameters

`bridge_name`	The name of the bridge from which to get the sink endpoint.

Exceptions

`BridgeNameNotFound`	No bridges with the specified name exist.

EndpointAdmin::get_source_endpoint()

```
SourceEndpoint get_source_endpoint(in BridgeName bridge_name)
raises (BridgeNameNotFound);
```

`get_source_endpoint()` returns a reference to the source endpoint of the specified bridge.

Parameters

`bridge_name`	The name of the bridge from which to get the source endpoint.

Exceptions

`BridgeNameNotFound`	No bridges with the specified name exist.

EndpointAdmin::get_all_sink_endpoints()

```
BridgeNameSeq get_all_sink_endpoints();
```

`get_all_sink_endpoints()` returns a list of the names of all bridges that have sink endpoints associated with them.

EndpointAdmin::get_all_source_endpoints()

```
BridgeNameSeq get_all_source_endpoints();
```

`get_all_source_endpoints()` returns a list of the names of all the bridges that have source endpoints associated with them.

IT_MessagingBridgeAdmin Module

IT_MessagingBridgeAdmin defines the data, exceptions, and interfaces to create and manage bridges. It defines the following interfaces:

- IT_MessagingBridgeAdmin::Bridge Interface
- IT_MessagingBridgeAdmin::BridgeAdmin Interface

IT_MessagingBridgeAdmin Data Types

IT_MessagingBridgeAdmin::BridgeName

```
typedef IT_MessagingBridge::BridgeName BridgeName;
```

BridgeName specifiys the uniqe identifier for a bridge object.

IT_MessagingBridgeAdmin::BridgeNameSeq

```
typedef IT_MessagingBridge::BridgeNameSeq BridgeNameSeq;
```

BridgeNameSeq contains a list of BridgeName. It is returned by IT_MessagingBridgeAdmin::BridgeAdmin::get_all_bridges().

IT_MessagingBridgeAdmin::InvalidEndpoitCode

```
typedef IT_MessagingBridge::InvalidEndpointCode
InvalidEndpointCode;
```

InvalidEndpointCode specifies the reason for an InvalidEndpoint exception.

IT_MessagingBridgeAdmin::EndpointInfo

```
struct EndpointInfo
{
```

```
  IT_MessagingBridge::EndpointAdmin admin;
  IT_MessagingBridge::EndpointType  type;
  IT_MessagingBridge::EndpointName  name;
};
```

`EndpointInfo` encapsulated the information needed to specify and endpoint to a bridge. It has the following fields:

`admin` A reference to the `EndpointAdmin` associated with the endpoint. For more information, see "IT_MessagingBridge::EndpointAdmin Interface" on page 1028.

`type` Specifies the endpoint's type. This correlates to the messaging service to which the endpoint is attached. For more information, see "IT_MessagingBridge::EndpointType" on page 1020.

`name` Specifies the unique identifier of the endpoint.

IT_MessagingBridgeAdmin Exceptions

IT_MessagingBridgeAdmin::CannotCreateBridge

```
exception CannotCreateBridge {};
```

`CannotCreateBridge` is raised when there is an error creating a bridge.

IT_MessagingBridgeAdmin::BridgeNotFound

```
exception BridgeNotFound {};
```

`BridgeNotFound` is raised when the bridge specified in either `get_bridge()` or `find_bridge()` does not exist.

IT_MessagingBridgeAdmin::BridgeAlreadyExists

```
exception BridgeAlreadyExists {BridgeName bridge_name;};
```

`BridgeAlreadyExists` if the endpoints specified in `create_bridge()` are already connected to form a bridge. It returns the name of the bridge connecting the endpoints.

IT_MessagingBridgeAdmin::BridgeNameAlreadyExists

```
exception BridgeNameAlreadyExists {};
```

`BridgeNameAlreadyExists` is raised when the bridge name specified in `create_bridge()` is already in use.

IT_MessagingBridgeAdmin::InvalidEndpoint

```
exception InvalidEndpoint
{
  EndpointInfo endpoint;
  InvalidEndpointCode code;
};
```

`InvalidEndpoint` is raised when one of the endpoints specified in `create_bridge()` is invalid. The first return value is a reference to the invalid endpoint and the second return value specifies why the endpoint is invalid.

IT_MessagingBridgeAdmin::Bridge Interface

```
interface Bridge
{
  readonly attribute BridgeName name;
  readonly attribute EndpointInfo source;
  readonly attribute EndpointInfo sink;

  void start();
  void suspend();
  void stop();
  void destroy();
};
```

`Bridge` specifies the attributes and operations of a uni-directional bridge between two endpoints. The bridge maintains a reference for each of its endpoints and provides the operations that control the flow of messages accross the bridge. It is recomended that developers use the operation defined on the bridge object as opposed to the operations specified by the IT_MessagingBridge::SourceEndpoint Interface.

Bridge::name

`readonly attribute BridgeName name;`

`name` specifies the identifyer for the bridge.

Bridge::source

`readonly attribute EndpointInfo source;`

`source` specifies the endpoint from which the bridge recieves messages.

Bridge::sink

```
readonly attribute EndpointInfo sink;
```

sink specifies the endpoint to which the bridge forwards messages.

Bridge::start()

```
void start();
```

start() signals the source endpoint to begin delivering messages to the bridge. Once the bridge begins recieving messages it fowards them the the sink endpoint.

Bridge::suspend()

```
void suspend();
```

suspend() signals the source endpoint to suspend the flow of messages. The bridge will not forward any messages while it is suspended, but the source endpoint will continue to queue messages for delievery to the bridge. Once start() has been called, the queued messages are forwarded.

Bridge::stop()

```
void stop();
```

stop() signals the source endpoint to completly halt the delivery of messages. No messages are queued for later delivery.

Bridge::destory()

```
void destroy();
```

destroy() destroys the bridge and cleans up all the resources associated with it, including the bridges endpoints.

IT_MessagingBridgeAdmin::BridgeAdmin Interface

```
interface BridgeAdmin
{
  Bridge create_bridge(in BridgeName   bridge_name,
                       in EndpointInfo source,
                       in EndpointInfo sink)
  raises (InvalidEndpoint, BridgeAlreadyExists,
          BridgeNameAlreadyExists, CannotCreateBridge);

  Bridge get_bridge(in BridgeName bridge_name)
  raises (BridgeNotFound);

  Bridge find_bridge(in  EndpointInfo source,
                     in  EndpointInfo sink,
                     out BridgeName bridge_name)
  raises (BridgeNotFound);

  BridgeNameSeq get_all_bridges();
};
```

`BridgeAdmin` defines the factory operation for `Bridge` objects. It also defines two operations to discover active bridges and one operation to list the bridges in the service. Developers get a reference to the `BridgeAdmin` by using the initial reference key `"IT_Messaging"`.

BridgeAdmin::create_bridge()

```
Bridge create_bridge(in BridgeName   bridge_name,
                     in EndpointInfo source,
                     in EndpointInfo sink)
raises (InvalidEndpoint, BridgeAlreadyExists,
        BridgeNameAlreadyExists, CannotCreateBridge);
```

`create_bridge()` creates a new uni-directional bridge between two endpoints and returns a reference to the bridge.

Parameters

`bridge_name`	Specifies the unique identifier for the bridge.
`source`	Specifies the endpoint from which the bridge will receive messages.
`sink`	Specifies the endpoint to which the bridge will forward messages.

Exceptions

`InvalidEndpoint`	One of the specified endpoints is not a valid endpoint for the new bridge.
`BridgeAlreadyExists`	A bridge connecting the two endpoints already exists.
`BridgeNameAlreadyExists`	The name specified for the bridge is already in use.
`CannotCreateBridge`	An unspecified error occurred while creating the bridge.

BridgeAdmin::get_bridge()

```
Bridge get_bridge(in BridgeName bridge_name)
raises (BridgeNotFound);
```

`get_bridge()` returns a reference to the specified bridge.

Parameters

`bridge_name`	Specifies the name of the bridge to get.

Exceptions

`BridgeNotFound`	The specified bridge does not exist.

BridgeAdmin::find_bridge()

```
Bridge find_bridge(in EndpointInfo source,
                   in  EndpointInfo sink,
                   out BridgeName bridge_name)
```

```
raises (BridgeNotFound);
```

`find_bridge()` returns a reference to the bridge linking the specified endpoints. The name of the bridge is returned as a parameter to the operation.

Parameters

`source`	Specifies the endpoint from which the bridge receives messages.
`sink`	Specifies the endpoint to which the bridge forwards messages.
`bridge_name`	Specifies the name of the returned bridge.

Exceptions

`BridgeNotFound`	The specified bridge does not exist.

BridgeAdmin::get_all_bridges()

```
BridgeNameSeq get_all_bridges();
```

`get_all_bridges()` returns a list containing the names of all existing bridges.

IT_NotifyBridge Module

IT_NotifyBridge defines an extenstion of IT_MessagingBridge::SinkEndpoint. This extension provides the method used by a bridge to forward notification events.

IT_NotifyBridge Exceptions

IT_NotifyBridge::MappingFailure

```
exception MappingFailure {};
```

MappingFailure is raised when the bridge is unable to properly map messages to a notification event.

IT_NotifyBridge::EndpointNotConnected

```
exception EndpointNotConnected {};
```

EndpointNotConnected is raised when an attempt to recieve messages through a SinkEndpoint that is not connected to a SourceEndpoint is made.

IT_NotifyBridge::SinkEndpoint Interface

```
interface SinkEndpoint : IT_MessagingBridge::SinkEndpoint
{
  void send_events(in CosNotification::EventBatch events)
  raises (MappingFailure, EndpointNotConnected);
};
```

`IT_NotifyBridge::SinkEndPoint` extends the functionality of `IT_MessagingBridge::SinkEndpoint` to include the ability to recieve notification style events. Due to the inheritance from `IT_MessagingBridge::SinkEndpoint`, it retains all of the functionality of a generic endpoint. `IT_NotifyBridge::SinkEndpoint` recieves a batch of notification events using the `CosNotificaiton::EventBatch` structure.

SinkEndpoint::send_events()

```
void send_events(in CosNotification::EventBatch events)
raises (MappingFailure, EndpointNotConnected);
```

`send_events()` revieves a batch of notification events from a bridge and passes them into the recieving messaging service.

Parameters

`events`	A group of notification events packaged into a `CosNotification::EventBatch`.

Exceptions

`MappingFailure`	The bridge encountered an error mapping the JMS messages to notification events.
`EndpointNotConnected`	The `SinkEndpoint` is not connected to a `SourceEndpoint`.

Object Transaction Service

CosTransactions Overview

The Object Management Group's (OMG) object transaction service (OTS) defines interfaces that integrate transactions into the distributed object paradigm. The OTS interface enables developers to manage transactions under two different models of transaction propagation, implicit and explicit:

- In the implicit model, the transaction context is associated with the client thread; when client requests are made on transactional objects, the transaction context associated with the thread is propagated to the object implicitly.
- In the explicit model, the transaction context must be passed explicitly when client requests are made on transactional objects in order to propagate the transaction context to the object.

Keep the following in mind:

- Applications must include the header file `CosTransactions.hh`.
- All of the OTS classes are nested within the `CosTransactions` namespace. Therefore, you must prefix `CosTransactions` to the OTS class and function names when using them in your application.
- All of the OTS class methods can throw the `CORBA::SystemException` exception if an object request broker (ORB) error occurs.

Overview of Classes

The OTS classes provide the following functionality:

- Managing transactions under the implicit model:
  `Current`
- Managing transactions under the explicit model:
  `TransactionFactory`
  `Control`
  `Coordinator`
  `Terminator`
- Managing resources in the CORBA environment:

RecoveryCoordinator
Resource
SubtransactionAwareResource
Synchronization

- Defining transactional interfaces in the CORBA environment:

  TransactionalObject

- Reporting system errors:

  HeuristicCommit
  HeuristicHazard
  HeuristicMixed
  HeuristicRollback
  Inactive
  InvalidControl
  INVALID_TRANSACTION
  NoTransaction
  NotPrepared
  NotSubtransaction
  SubtransactionsUnavailable
  TRANSACTION_MODE
  TRANSACTION_REQUIRED
  TRANSACTION_ROLLEDBACK
  TRANSACTION_UNAVAILABLE
  Unavailable

General Data Types

OTS defines enumerated data types to represent the status of a transaction object during its lifetime and to indicate a participant's vote on the outcome of a transaction.

Status Enumeration Type

```
enum Status{
    StatusActive,
    StatusMarkedRollback,
    StatusPrepared,
    StatusCommitted,
    StatusRolledBack,
```

```
    StatusUnknown,
    StatusNoTransaction,
    StatusPreparing,
    StatusCommitting,
    StatusRollingBack
};
```

The `Status` enumerated type defines values that are used to indicate the status of a transaction. Status values are used in both the implicit and explicit models of transaction demarcation defined by OTS. The `Current:: get_status()` operation can be called to return the transaction status if the implicit model is used. The `Coordinator:: get_status()` operation can be called to return the transaction status if the explicit model is used.

The `Status` values indicate the following:

`StatusActive`	Processing of a transaction is still in progress.
`StatusMarkedRollback`	A transaction is marked to be rolled back.
`StatusPrepared`	A transaction has been prepared but not completed.
`StatusCommitted`	A transaction has been committed and the effects of the transaction have been made permanent.
`StatusRolledBack`	A transaction has been rolled back.
`StatusUnknown`	The status of a transaction is unknown.
`StatusNoTransaction`	A transaction does not exist in the current transaction context.
`StatusPreparing`	A transaction is preparing to commit.
`StatusCommitting`	A transaction is in the process of committing.
`StatusRollingBack`	A transaction is in the process of rolling back.

See Also

`CosTransactions::Coordinator::get_status()`
`CosTransactions::Current::get_status()`

Vote Enumeration Type

```
enum Vote{
    VoteCommit,
    VoteRollback,
    VoteReadOnly
};
```

The `Vote` enumerated type defines values for the voting status of transaction participants. The participants in a transaction each vote on the outcome of a transaction during the two-phase commit process. In the prepare phase, a `Resource` object can vote whether to commit or abort a transaction. If a `Resource` has not modified any data as part of the transaction, it can vote `VoteReadOnly` to indicate that its participation does not affect the outcome of the transaction. The `Vote` values specify the following:

`VoteCommit`	The value used to indicate a vote to commit a transaction.
`VoteRollback`	The value used to indicate a vote to abort (rollback) a transaction.
`VoteReadOnly`	The value used to indicate no vote on the outcome of a transaction.

See Also `CosTransactions::Resource`

OTSPolicyValue Data Type

```
typedef unsigned short OTSPolicyValue;
const OTSPolicyValue REQUIRES = 1;
const OTSPolicyValue FORBIDS = 2;
const OTSPolicyValue ADAPTS = 3;
const CORBA::PolicyType OTS_POLICY_TYPE = 56;
```

The `OTSPolicyValue` data type is used to create POA policy objects that define behavior of objects during invocations, both with and without a current transaction.

The `CORBA::ORB::create_policy()` operation is used to create the policy objects (passing in the appropriate `OTSPolicyValue` value). The policy object is passed in the list of policy objects passed to `PortableServer::POA::create_POA()`.

The OTSPolicyValue values indicate the following:

REQUIRES	The target object depends on the presence of a transaction. If there is no current transaction, a TRANSACTION_REQUIRED system exception is raised.
FORBIDS	The target object depends on the absence of a transaction. If there is a current transaction, the INVALID_TRANSACTION system exception is raised. When there is no current transaction, the behavior of the FORBIDS policy is also affected by the NonTxTargetPolicy.
ADAPTS	The target object is invoked within the current transaction, whether there is one or not.

You cannot create a POA that mixes the OTSPolicyValue FORBIDS or ADAPTS values with the InvocationPolicyValue EITHER or UNSHARED values. Attempting to do so raises PortableServer::InvalidPolicy exception.

Examples

The following example shows the ADAPTS value:

```
//C++
CORBA::ORB_var orb = ...
CORBA::Any policy_val;
policy_val <<= CosTransactions::ADAPTS;
CORBA::Policy_var policy =
  orb->create_policy(CosTransactions::OTS_POLICY_TYPE,
                     policy_val);
```

See Also

CosTransactions::NonTxTargetPolicyValue
CosTransactions::TransactionalObject

InvocationPolicyValue Data Type

```
typedef unsigned short InvocationPolicyValue;
const InvocationPolicyValue EITHER = 0;
const InvocationPolicyValue SHARED = 1;
const InvocationPolicyValue UNSHARED = 2;
const CORBA::PolicyType INVOCATION_POLICY_TYPE = 55;
```

The InvocationPolicyValue data type is used to create POA policy objects that define the behavior of objects with respect to the *shared* and *unshared* transaction models.

The shared transaction model represents a standard end-to-end transaction that is shared between the client and the target object. The unshared transaction model uses asynchronous messaging where separate transactions are used along the invocation path. Hence, the client and the target object do not share the same transaction.

The CORBA::ORB::create_policy() operation is used to create the policy objects (passing in the appropriate InvocationPolicyValue). The policy object is passed in the list of policy objects passed to PortableServer::POA::create_POA().

The InvocationPolicyValue data type values indicate the following:

EITHER	The target object supports both shared and unshared invocations.
SHARED	The target object supports synchronous invocations and asynchronous includes that do not involve a routing element.
UNSHARED	The target object.

You cannot create a POA that mixes the InvocationPolicyValue EITHER or UNSHARED values with the OTSPolicyValue FORBIDS or ADAPTS values. Attempting to do this raises a PortableServer::InvalidPolicy exception.

If no InvocationPolicy object is passed to create_POA(), the InvocationPolicy defaults to SHARED.

Note: The unshared transaction model is not supported in this release.

Examples

The following example shows the SHARED value:

```
//C++
CORBA::ORB_var orb = ...
CORBA::Any policy_val;
policy_val <<= CosTransactions::SHARED;
CORBA::Policy_var policy =
  orb->create_policy(CosTransactions::INVOCATION_POLICY_TYPE,
                     policy_val);
```

See Also

CosTransactions::OTSPolicyValue
CosTransactions::NonTxTargetPolicyValue

NonTxTargetPolicyValue Data Type

```
typedef unsigned short NonTxTargetPolicyValue;
const NonTxTargetPolicyValue PREVENT = 0;
const NonTxTargetPolicyValue PERMIT = 1;
const CORBA::PolicyType NON_TX_TARGET_POLICY_TYPE = 57;
```

The NonTxTargetPolicyValue data type is used to create policy objects used by clients to affect the behavior of invocations on objects with an OTSPolicy of FORBIDS.

The CORBA::ORB::create_policy() operation creates the policy objects (passing the appropriate NonTxTargetPolicyValue). The policy object is passed in the list of policy objects passed to CORBA::PolicyManager::set_policy_overrides() and CORBA::PolicyCurrent::set_policy_overrides().

See the CORBA::PolicyCurrent and CORBA::PolicyManager classes for more details on setting policies.

The behavior of the NonTxTargetPolicy values apply to invocations where there is a current transaction and the target object has the OTSPolicyValue of FORBIDS. The NonTxTargetPolicy values indicate the following:

PREVENT	The invocation is prevented from proceeding and the system exception INVALID_TRANSACTION is raised.
PERMIT	The invocation proceeds but not in the context of the current transaction.

The default NonTxTargetPolicy is PREVENT.

Examples The following example shows the PERMIT value:

```
//C++
CORBA::ORB_var orb = ...
CORBA::Any policy_val;
policy_val <<= CosTransactions::PERMIT;
CORBA::Policy_var policy =
  orb->create_policy(CosTransactions::NON_TX_TARGET_POLICY_TYPE,
                     policy_val);
```

See Also CosTransactions::OTSPolicyValue
CosTransactions::InvocationPolicyValue

TransactionPolicyValue Data Type

```
typedef unsigned short TransactionPolicyValue;
const TransactionPolicyValue Allows_shared = 0;
const TransactionPolicyValue Allows_none = 1;
const TransactionPolicyValue Requires_shared = 2;
const TransactionPolicyValue Allows_unshared = 3;
const TransactionPolicyValue Allows_either = 4;
const TransactionPolicyValue Requires_unshared = 5;
const TransactionPolicyValue Requires_either = 6;
const CORBA::PolicyType TRANSACTION_POLICY_TYPE = 36;
```

The `TransactionalPolicyValue` data type has been deprecated and replaced with the `OTSPolicyValue` and `InvocationPolicyValue` types.

The `TransactionalPolicyValue` data type has been retained in this release for backward compatibility. See the *CORBA Programmer's Guide* for details of interoperability with previous Orbix releases.

General Exceptions

Errors are handled in OTS by using exceptions. Exceptions provide a way of returning error information back through multiple levels of procedure or method calls, propagating this information until a method or procedure is reached that can respond appropriately to the error.

Each of the following exceptions are implemented as classes. The exceptions are shown here in two tables: one for the OTS exceptions and another for the system exceptions.

Table 25: *OTS Exceptions*

Exception	Description
`HeuristicCommit`	This exception is thrown to report that a heuristic decision was made by one or more participants in a transaction and that all updates have been committed. See Also: `Resource class`

Table 25: *OTS Exceptions*

Exception	Description

`HeuristicHazard`	This exception is thrown to report that a heuristic decision has possibly been made by one or more participants in a transaction and the outcome of all participants in the transaction is unknown. See Also:
	
	`Current::commit()`
	`Resource` class
	`Terminator::commit()`

`HeuristicMixed`	This exception is thrown to report that a heuristic decision was made by one or more participants in a transaction and that some updates have been committed and others rolled back. See Also:
	
	`Current::commit()`
	`Resource` class
	`Terminator::commit()`

`HeuristicRollback`	This exception is thrown to report that a heuristic decision was made by one or more participants in a transaction and that all updates have been rolled back. See Also: `Resource` class

`Inactive`	This exception is thrown when a transactional operation is requested for a transaction, but that transaction is already prepared. See Also:
	
	`Coordinator::create_subtransaction()`
	`Coordinator::register_resource()`
	`Coordinator::register_subtran_aware()`
	`Coordinator::rollback_only()`

`InvalidControl`	This exception is thrown when an invalid `Control` object is used in an attempt to resume a suspended transaction. See Also: `Control` class `Current::resume()`

Table 25: *OTS Exceptions*

Exception	Description
`NotPrepared`	This exception is thrown when an operation (such as a `commit()`) is requested for a resource, but that resource is not prepared. See Also: `RecoveryCoordinator::replay_completion()` `Resource` class

`NoTransaction`	This exception is thrown when an operation is requested for the current transaction, but no transaction is associated with the client thread. See Also:
	`Current::commit()`
	`Current::rollback()`
	`Current::rollback_only()`

`NotSubtransaction`	This exception is thrown when an operation that requires a subtransaction is requested for a transaction that is not a subtransaction. See Also: `Coordinator::register_subtran_aware()`

`SubtransactionsUnavailable`	This exception is thrown when an attempt is made to create a subtransaction. See Also:
	`Coordinator::create_subtransaction()`
	`Current::begin()`

`Unavailable`	This exception is thrown when a `Terminator` or `Coordinator` object cannot be provided by a `Control` object due to environment restrictions. See Also:
	`Control::get_coordinator()`
	`Control::get_terminator()`

The following table shows the system exceptions that can be thrown:

Table 26: *System Exceptions*

Exception	Description
`INVALID_TRANSACTION`	This exception is raised when the transaction context is invalid for a request.
`TRANSACTION_MODE`	This exception is raised when there is a mismatch between the transaction policy in the target object's IOR and the current transaction mode (see Table 1).
`TRANSACTION_REQUIRED`	This exception is raised when an invocation on an object expecting a transaction is performed with no transaction (see Table 1).

`TRANSACTION_ROLLEDBACK`	This exception is raised when a transactional operation (such as `commit()`) is requested for a transaction that has been rolled back or marked for rollback. See Also:
	`Current::commit()`
	`Terminator::commit()`

`TRANSACTION_UNAVAILABLE`	This exception is raised when a transaction invocation is requested but the transaction service is not available.

CosTransactions::Control Class

The `Control` class enables explicit control of a factory-created transaction; the factory creates a transaction and returns a `Control` instance associated with the transaction. The `Control` object provides access to the `Coordinator` and `Terminator` objects used to manage and complete the transaction.

A `Control` object can be used to propagate a transaction context explicitly. By passing a `Control` object as an argument in a request, the transaction context can be propagated. `TransactionFactory::create()` can be used to create a transaction and return the `Control` object associated with it.

```
// C++
class Control {
public:
    Terminator_ptr get_terminator();
    Coordinator_ptr get_coordinator();
};
typedef Control *Control_ptr;
class Control_var;
```

See Also

`CosTransactions::Coordinator`
`CosTransactions::Current::get_control()`
`CosTransactions::Coordinator::get_status()`
`CosTransactions::Terminator`
`CosTransactions::TransactionFactory::create()`
`NoTransaction`
`NotSubtransaction`

Control::get_coordinator()

```
// C++
Coordinator_ptr get_coordinator()
    throw(CORBA::SystemException, Unavailable);
```

`get_coordinator()` returns the `Coordinator` object for the transaction with which the `Control` object is associated. The returned `Coordinator` object can be used to determine the status of the transaction, the relationship between

the associated transaction and other transactions, to create subtransactions, and so on.

Exceptions

Unavailable	The Coordinator associated with the Control object is not available.

See Also CosTransactions::Coordinator

Control::get_terminator()

```
// C++
Terminator_ptr get_terminator()
    throw(CORBA::SystemException, Unavailable);
```

get_terminator() returns the Terminator object for the transaction with which the Control object is associated. The returned Terminator object can be used to either commit or roll back the transaction.

Exceptions

Unavailable	The Terminator associated with the Control object is not available.

See Also CosTransactions::Terminator

CosTransactions::Coordinator Class

The Coordinator class enables explicit control of a factory-created transaction; the factory creates a transaction and returns a Control instance associated with the transaction. Control::get_coordinator() returns the Coordinator object used to manage the transaction.

The operations defined by the Coordinator class can be used by the participants in a transaction to determine the status of the transaction, determine the relationship of the transaction to other transactions, mark the transaction for rollback, and create subtransactions.

The Coordinator class also defines operations for registering resources as participants in a transaction and registering subtransaction-aware resources with a subtransaction.

```
// C++
class Coordinator {
    public:
    char *get_transaction_name();
    Status get_status();
    Status get_parent_status();
    Status get_top_level_status();
    CORBA::Boolean is_same_transaction(Coordinator_ptr);
    CORBA::Boolean is_related_transaction(Coordinator_ptr);
    CORBA::Boolean is_ancestor_transaction(Coordinator_ptr);
    CORBA::Boolean is_descendant_transaction(Coordinator_ptr);
    CORBA::Boolean is_top_level_transaction();
    unsigned long hash_transaction();
    unsigned long hash_top_level_tran();
    RecoveryCoordinator register_resource(Resource);
    void register_subtran_aware(SubtransactionAwareResource);
    Control_ptr create_subtransaction();
    void rollback_only();
    PropagationContext* get_txcontext()
};
typedef Coordinator *Coordinator_ptr;
class Coordinator_var;
```

See Also CosTransactions::Control

CosTransactions::Control::get_coordinator()
CosTransactions::Terminator

Coordinator::create_subtransaction()

```
// C++
Control_ptr create_subtransaction()
    throw(CORBA::SystemException, Inactive,
   SubtransactionsUnavailable);
```

`create_subtransaction()` returns the `Control` object associated with the new subtransaction.

`create_subtransaction()` creates a new subtransaction for the transaction associated with the `Coordinator` object. A subtransaction is one that is embedded within another transaction; the transaction within which the subtransaction is embedded is referred to as its parent. A transaction that has no parent is a top-level transaction. A subtransaction executes within the scope of its parent transaction and can be used to isolate failures; if a subtransaction fails, only the subtransaction is rolled back. If a subtransaction commits, the effects of the commit are not permanent until the parent transaction commits. If the parent transaction rolls back, the subtransaction is also rolled back.

Exceptions

SubtransactionsUnavailable	Subtransactions are not supported.
Inactive	The transaction is already prepared.

See Also CosTransactions::Control

Coordinator::get_parent_status()

```
// C++
Status get_parent_status()
    throw(CORBA::SystemException);
```

get_parent_status() returns the status of the parent of the transaction associated with the Coordinator object. For more information, see create_subtransaction().

The status returned indicates which phase of processing the transaction is in. See the reference page for the Status type for information about the possible status values. If the transaction associated with the Coordinator object is a subtransaction, the status of its parent transaction is returned. If there is no parent transaction, the status of the transaction associated with the Coordinator object itself is returned.

See Also

CosTransactions::Coordinator::create_subtransaction()
CosTransactions::Coordinator::get_status()
CosTransactions::Coordinator::get_top_level_status()
CosTransactions::Status

Coordinator::get_status()

```
// C++
Status get_status()
    throw(CORBA::SystemException);
```

get_status() returns the status of the transaction associated with the Coordinator object. The status returned indicates which phase of processing the transaction is in. See the reference page for the Status type for information about the possible status values.

See Also

CosTransactions::Coordinator::get_parent_status()
CosTransactions::Coordinator::get_top_level_status()
CosTransactions::Status

Coordinator::get_top_level_status()

```
// C++
Status get_top_level_status()
    throw(CORBA::SystemException);
```

get_top_level_status() returns the status of the top-level ancestor of the transaction associated with the Coordinator object. See Coordinator::create_subtransaction() for more information.

The status returned indicates which phase of processing the transaction is in. See the reference page for the Status type for information about the possible status values. If the transaction associated with the Coordinator object is the top-level transaction, its status is returned.

See Also

CosTransactions::Coordinator::create_subtransaction()
CosTransactions::Coordinator::get_status()
CosTransactions::Coordinator::get_parent_status()
CosTransactions::Status

Coordinator::get_transaction_name()

```
// C++
char *get_transaction_name();
```

get_transaction_name() returns the name of the transaction associated with the Coordinator object.

Coordinator::get_txcontext()

```
// C++
PropagationContext* Coordinator::get_txcontext()
    throw (CORBA::SystemException, Unavailable);
```

Returns the propagation context object which is used to export the current transaction to a new transaction service domain.

Exceptions

Unavailable	The propagation context is unavailable.

See Also

CosTransactions::TransactionFactory::recreate()

Coordinator::hash_top_level_tran()

```
// C++
unsigned long hash_top_level_tran()
    throw(CORBA::SystemException);
```

hash_top_level_tran() returns a hash code for the top-level ancestor of the transaction associated with the Coordinator object. If the transaction associ-

ated with the Coordinator object is the top-level transaction, its hash code is returned. See create_subtransaction() for more information. The returned hash code is typically used as an index into a table of Coordinator objects. The low-order bits of the hash code can be used to hash into a table with a size that is a power of two.

See Also

CosTransactions::Coordinator::create_subtransaction()
CosTransactions::Coordinator::hash_transaction()

Coordinator::hash_transaction()

```
// C++
unsigned long hash_transaction()
    throw(CORBA::SystemException);
```

hash_transaction() returns a hash code for the transaction associated with the Coordinator object.

See Also

CosTransactions::Coordinator::hash_top_level_tran()

Coordinator::is_ancestor_transaction()

```
// C++
CORBA::Boolean is_ancestor_transaction(
    Coordinator_ptr tc
)
    throw(CORBA::SystemException);
```

is_ancestor_transaction() returns true if the transaction is an ancestor or if the two transactions are the same; otherwise, the method returns false.

Parameters

tc	Specifies the coordinator of another transaction to compare with the Coordinator object.

is_ancestor_transaction() determines whether the transaction associated with the Coordinator object is an ancestor of the transaction associated with the coordinator specified in the tc parameter. See create_subtransaction() for more information.

See Also

CosTransactions::Coordinator::is_descendant_transaction()
CosTransactions::Coordinator::is_related_transaction()

CosTransactions::Coordinator::is_same_transaction()
CosTransactions::Coordinator::create_subtransaction()

Coordinator::is_descendant_transaction()

```
// C++
CORBA::Boolean is_descendant_transaction(Coordinator_ptr tc)
    throw(CORBA::SystemException);
```

is_descendant_transaction() returns true if the transaction is a descendant or if the two transactions are the same; otherwise, the method returns false.

Parameters

tc	Specifies the coordinator of another transaction to compare with the Coordinator object.

is_descendant_transaction() determines whether the transaction associated with the Coordinator object is a descendant of the transaction associated with the coordinator specified in the tc parameter. See Coordinator::create_subtransaction() for more information.

See Also

CosTransactions::Coordinator::is_descendant_transaction()
CosTransactions::Coordinator::is_related_transaction()
CosTransactions::Coordinator::is_same_transaction()
CosTransactions::Coordinator::is_top_level_transaction()
CosTransactions::Coordinator::create_subtransaction()

Coordinator::is_related_transaction()

```
// C++
CORBA::Boolean is_related_transaction(
    Coordinator_ptr tc
)
    throw(CORBA::SystemException);
```

is_related_transaction() returns true if both transactions are descendants of the same transaction; otherwise, the method returns false.

Parameters

`tc`	Specifies the coordinator of another transaction to compare with the `Coordinator` object.

`is_related_transaction()` determines whether the transaction associated with the `Coordinator` object and the transaction associated with the coordinator specified in the `tc` parameter have a common ancestor. See `create_subtransaction()` for more information.

See Also

`CosTransactions::Coordinator::is_descendant_transaction()`
`CosTransactions::Coordinator::is_ancestor_transaction()`
`CosTransactions::Coordinator::is_same_transaction()`
`CosTransactions::Coordinator::is_top_level_transaction()`
`CosTransactions::Coordinator::create_subtransaction()`

Coordinator::is_same_transaction()

```
// C++
CORBA::Boolean is_same_transaction(
    Coordinator_ptr tc
)
    throw(CORBA::SystemException);
```

`is_same_transaction()` returns true if the transactions associated with the two `Coordinator` objects are the same transaction; otherwise, the method returns false.

Parameters

`tc`	Specifies the coordinator of another transaction to compare with the `Coordinator` object.

`is_same_transaction()` determines whether the transaction associated with the `Coordinator` object and the transaction associated with the coordinator specified in the `tc` parameter are the same transaction.

See Also

`CosTransactions::Coordinator::is_descendant_transaction()`
`CosTransactions::Coordinator::is_related_transaction()`
`CosTransactions::Coordinator::is_ancestor_transaction()`
`CosTransactions::Coordinator::is_top_level_transaction()`

Coordinator::is_top_level_transaction()

```
// C++
CORBA::Boolean is_top_level_transaction()
    throw(CORBA::SystemException);
```

is_top_level_transaction() returns true if the transaction is a top-level transaction; otherwise, the method returns false. is_top_level_transaction() determines whether the transaction associated with a Coordinator object is a top-level transaction. See create_subtransaction() for more information.

See Also

CosTransactions::Coordinator::is_descendant_transaction()
CosTransactions::Coordinator::is_related_transaction()
CosTransactions::Coordinator::is_same_transaction()
CosTransactions::Coordinator::is_ancestor_transaction()
CosTransactions::Coordinator::create_subtransaction()

Coordinator::register_resource()

```
// C++
RecoveryCoordinator register_resource(
    Resource resource
)
    throw(CORBA::SystemException, Inactive);
```

register_resource() registers a specified resource as a participant in the transaction associated with a Coordinator object. When the transaction ends, the registered resource must commit or roll back changes made as part of the transaction. Only server applications can register resources. See Resource class for more information. register_resource() returns a RecoveryCoordinator object that the registered Resource object can use during recovery.

Parameters

resource	The resource to register as a participant.

Exceptions

CORBA::TRANSACTION_ROLLEDBACK	The transaction is marked for rollback only.

See Also

CosTransactions::RecoveryCoordinator
CosTransactions::Resource

Coordinator::register_subtran_aware()

```
// C++
void register_subtran_aware(
    SubtransactionAwareResource resource
)
    throw(CORBA::SystemException, NotSubtransaction, Inactive);
```

`register_subtran_aware()` registers a specified resource with the subtransaction associated with a `Coordinator` object. The resource is registered with the subtransaction only, not as a participant in the top-level transaction. (`register_resource()` can be used to register the resource as a participant in the top-level transaction.) Only server applications can register resources.

Parameters

`resource`	The resource to register.

When the transaction ends, the registered resource must commit or roll back changes made as part of the subtransaction. See the reference page for the `SubtransactionAwareResource` class for more information.

Exceptions

`NotSubtransaction`	The transaction associated with the `Coordinator` object is not a subtransaction
`Inactive`	The subtransaction or any ancestor of the subtransaction has ended.
`CORBA::TRANSACTION_ROLLEDBACK`	The transaction is marked for rollback only.

See Also

`CosTransactions::RecoveryCoordinator`
`CosTransactions::SubtransactionAwareResource`

Coordinator::register_synchronization()

```
// C++
void register_synchronization(
    Synchronization sync
);
    throw(CORBA::SystemException, Inactive);
```

register_synchronization() registers a specified synchronization object for the transaction associated with a Coordinator object. See the reference page for the Synchronization class for more information.

Parameters

sync	The synchronization object to register.

Exceptions

Inactive	The transaction is already prepared.
CORBA::TRANSACTION_ROLLEDBACK	The transaction is marked for rollback only.

See Also CosTransactions::RecoveryCoordinator
CosTransactions::Synchronization

Coordinator::rollback_only()

```
// C++
void rollback_only()
    throw(CORBA::SystemException, Inactive);
```

rollback_only() marks the transaction associated with the Coordinator object so that the only possible outcome for the transaction is to roll back. The transaction is not rolled back until the participant that created the transaction either commits or aborts the transaction.

OTS allows Terminator::rollback() to be called instead of rollback_only(). Calling Terminator::rollback() rolls back the transaction immediately, preventing unnecessary work from being done between the time the transaction is marked for rollback and the time the transaction is actually rolled back.

Exceptions

Inactive	The transaction is already prepared.

See Also CosTransactions::Terminator::rollback()

CosTransactions::Current Class

The `Current` class represents a transaction that is associated with the calling thread; the thread defines the transaction context. The transaction context is propagated implicitly when the client issues requests.

This class defines member methods for beginning, committing, and aborting a transaction using the implicit model of transaction control. It also defines member methods for suspending and resuming a transaction and retrieving information about a transaction.

```
// C++
class Current {
  public:
    void begin();
    void commit(CORBA::Boolean);
    void rollback();
    void rollback_only();
    Status get_status();
    char *get_transaction_name();
    void set_timeout(unsigned long);
    unsigned long get_timeout();
    Control_ptr get_control();
    Control_ptr suspend();
    void resume(Control_ptr);
};
typedef Current *Current_ptr;
class Current_var;
```

See Also CosTransactions::Control
CosTransactions::Status

Current::begin()

```
// C++
void begin()
    throw(CORBA::SystemException,SubtransactionsUnavailable);
```

begin() creates a new transaction and modifies the transaction context of the calling thread to associate the thread with the new transaction. If subtransactions are not available, an attempt to create a nested transaction throws the SubtransactionsUnavailable exception.

See Also

CosTransactions::Current::commit()
CosTransactions::Current::rollback()
CosTransactions::Current::rollback_only()

Current::commit()

```
// C++
void commit(
    CORBA::Boolean report_heuristics
)
    throw(CORBA::SystemException,
            NoTransaction,
            HeuristicHazard,
            TRANSACTION_ROLLEDBACK);
```

commit() attempts to commit the transaction associated with the calling thread.

Parameters

report_heuristics	specifies whether to report heuristic decisions for the transaction associated with the calling thread.

Exceptions

NoTransaction exception	No transaction is associated with the calling thread.
HeuristicMixed	The report_heuristics parameter is true and a heuristic decision causes inconsistent outcomes
HeuristicHazard	The report_heuristics parameter is true and a heuristic decision might have caused inconsistent outcomes.
TRANSACTION_ROLLEDBACK	Not all the transaction participants commit.

See Also

CosTransactions::Current::begin()
CosTransactions::Current::rollback()
CosTransactions::Current::rollback_only()

Current::get_control()

```
// C++
Control_ptr get_control()
    throw(CORBA::SystemException);
```

`get_control()` returns the `Control` object for the transaction associated with the calling thread. If no transaction is associated with the calling thread, a null object reference is returned.

See Also `CosTransactions::Current::resume()`

Current::get_status()

```
// C++
Status get_status()
    throw(CORBA::SystemException);
```

`get_status()` returns the status of the transaction associated with the calling thread. If no transaction is associated with the calling thread, the `StatusNoTransaction` value is returned.

The status returned indicates the processing phase of the transaction. See the `Status` type for information about the possible status values.

See Also `CosTransactions::Status` `Enumeration Type`

Current::get_timeout()

```
// C++
unsigned long get_timeout()
    throw(CORBA::SystemException)
```

Returns the timeout in seconds for transactions created using the `begin()` operation.

See Also `CosTransactions::Current`
`CosTransactions::Current::begin()`
`CosTransactions::Current::set_timeout()`

Current::get_transaction_name()

```
// C++
char *get_transaction_name();
```

`get_transaction_name()` returns the name of the transaction associated with the calling thread. If no transaction is associated with the calling thread, a null string is returned.

See Also `CosTransactions::Current`

Current::resume()

```
// C++
void resume(
    Control_ptr which
)
    throw(CORBA::SystemException, InvalidControl);
```

`resume()` resumes the suspended transaction identified by the which parameter and associated with the calling thread. If the value of the which parameter is a null object reference, the calling thread disassociates from the transaction. If the control object is invalid, the `InvalidControl` exception is thrown.

Parameters

`which`	Specifies a `Control` object that represents the transaction context associated with the calling thread.

See Also `CosTransactions::Current`
`CosTransactions::Current::get_control()`
`CosTransactions::Current::suspend()`

Current::rollback()

```
// C++
void rollback()
    throw(CORBA::SystemException, NoTransaction);
```

`rollback()` rolls back the transaction associated with the calling thread. If the transaction was started with `begin()`, the transaction context for the thread is restored to its state before the transaction was started; otherwise, the transaction context is set to null.

Exceptions

NoTransaction No transaction is associated with the calling thread.

See Also

CosTransactions::Current
CosTransactions::Current::begin()
CosTransactions::Current::rollback_only()

Current::rollback_only()

```
// C++
void rollback_only()
    throw(CORBA::SystemException, NoTransaction);
```

`rollback_only()` marks the transaction associated with the calling thread for rollback. The transaction is modified so that the only possible outcome is to roll back the transaction. Any participant in the transaction can mark the transaction for rollback. The transaction is not rolled back until the participant that created the transaction either commits or aborts the transaction.

OTS allows `Current::rollback()` to be called instead of `rollback_only()`. Calling `Current::rollback()` rolls back the transaction immediately, preventing unnecessary work from being done between the time the transaction is marked for rollback and the time the transaction is actually rolled back.

Exceptions

NoTransaction No transaction is associated with the calling thread.

See Also

CosTransactions::Current
CosTransactions::Current::rollback()

Current::set_timeout()

```
// C++
void set_timeout(
    unsigned long seconds
)
    throw(CORBA::SystemException);
```

set_timeout() sets a timeout period for the transaction associated with the calling thread. The timeout affects only those transactions begun with begin() after the timeout is set. The seconds parameter sets the number of seconds from the time the transaction is begun that it waits for completion before being rolled back; if the seconds parameter is zero, no timeout is set for the transaction.

Parameters

seconds	The number of seconds that the transaction waits for completion before rolling back.

See Also

CosTransactions::Current
CosTransactions::Current::begin()
CosTransactions::Current::get_timeout()

Current::suspend()

```
// C++
Control_ptr suspend()
    throw(CORBA::SystemException);
```

suspend() suspends the transaction associated with the calling thread. An identifier for the suspended transaction is returned by the method. This identifier can be passed to resume() to resume the suspended transaction.

See Also

CosTransactions::Current
CosTransactions::Current::resume()

CosTransactions:: RecoveryCoordinator Class

The RecoveryCoordinator class enables a recoverable object to control the recovery process for an associated resource. A RecoveryCoordinator object can be obtained for a recoverable object via the Coordinator object associated with the recoverable object. Coordinator::register_resource() returns a RecoveryCoordinator object.

```
// C++
class RecoveryCoordinator {
    public:
    Status replay_completion(Resource_ptr);
};
typedef RecoveryCoordinator *RecoveryCoordinator_ptr;
class RecoveryCoordinator_var;
```

See Also CosTransactions::Resource

RecoveryCoordinator::replay_completion()

```
// C++
Status replay_completion(
    Resource_ptr resource
)
    throw(CORBA::SystemException, NotPrepared);
```

replay_completion() notifies the recovery coordinator that the commit() or rollback() operations have not been performed for the associated resource. Notifying the coordinator that the resource has not completed causes completion to be retried, which is useful in certain failure cases. The method returns the current status of the transaction.

Parameters

`resource`	The resource associated with the recovery coordinator.

Exceptions

`NotPrepared`	The resource is not in the prepared state.

See Also `CosTransactions::Resource`
`CosTransactions::Status`

CosTransactions::Resource Class

The Resource class represents a recoverable resource, that is, a transaction participant that manages data subject to change within a transaction. The Resource class specifies the protocol that must be defined for a recoverable resource. Interfaces that inherit from this class must implement each of the member methods to manage the data appropriately for the recoverable object based on the outcome of the transaction. These methods are invoked by the Transaction Service to execute two-phase commit; the requirements of these methods are described in the following sections.

To become a participant in a transaction, a Resource object must be registered with that transaction. Coordinator::register_resource() can be used to register a resource for the transaction associated with the Coordinator object.

The full name for the class is CosTransactions::Resource.

```
// C++
class Resource {
    public:
    virtual Vote prepare();
    virtual void rollback();
    virtual void commit();
    virtual void commit_one_phase();
    virtual void forget();
};
typedef Resource *Resource_ptr;
class Resource_var;
```

See Also

CosTransactions::Synchronization
CosTransactions::RecoveryCoordinator
CosTransactions::Vote

Two-phase Commit

The two-phase commit requires methods prepare() and commit().

`prepare()` must be defined to vote on the outcome of the transaction with which the resource is registered. The transaction service invokes this method as the first phase of a two-phase commit; the return value controls the second phase:

- Returns `VoteReadOnly` if the resource's data is not modified by the transaction. The transaction service does not invoke any other methods on the resource, and the resource can forget all knowledge of the transaction.
- Returns `VoteCommit` if the resource's data is written to stable storage by the transaction and the transaction is prepared. Based on the outcome of other participants in the transaction, the transaction service calls either `commit()` or `rollback()` for the resource. The resource should store a reference to the `RecoveryCoordinator` object in stable storage to support recovery of the resource.
- Returns `VoteRollback` for all other situations. The transaction service calls `rollback()` for the resource, and the resource can forget all knowledge of the transaction.

`commit()` must be defined to commit all changes made to the resource as part of the transaction. If `forget()` has already been called, no changes need to be committed. If the resource has not been prepared, the `NotPrepared` exception must be thrown.

Use the heuristic outcome exceptions to report heuristic decisions related to the resource. The resource must remember heuristic outcomes until `forget()` is called, so that the same outcome can be returned if the transaction service calls `commit()` again.

One-phase Commit

`commit_one_phase()` must be defined to commit all changes made to the resource as part of the transaction. The transaction service may invoke this method if the resource is the only participant in the transaction. Unlike `commit()`, `commit_one_phase()` does not require that the resource be prepared first. Use the heuristic outcome exceptions to report heuristic decisions related to the resource. The resource must remember heuristic outcomes until `forget()` is called, so that the same outcome can be returned if the transaction service calls `commit_one_phase()` again.

Rollback Transaction

`rollback()` must be defined to undo all changes made to the resource as part of the transaction. If `forget()` has been called, no changes need to be undone. Use the heuristic outcome exceptions to report heuristic decisions related to the resource. The resource must remember heuristic outcomes until `forget()` is called, so that the same outcome can be returned if the transaction service calls `rollback()` again.

Forget Transaction

`forget()` must be defined to cause the resource to forget all knowledge of the transaction. The transaction service invokes this method if the resource throws a heuristic outcome exception in response to `commit()` or `rollback()`.

CosTransactions:: SubtransactionAwareResource Class

Note: This class is not supported in this release of OTS for Orbix. The information in this section therefore does not apply to this release.

The `SubtransactionAwareResource` class represents a recoverable resource that makes use of nested transactions. This specialized resource object allows the resource to be notified when a subtransaction for which it is registered either commits or rolls back.

The `SubtransactionAwareResource` class specifies the protocol that must be defined for this type of recoverable resource. Interfaces that inherit from this class must implement each of the member methods to manage the recoverable object's data appropriately based on the outcome of the subtransaction. These methods are invoked by the transaction service; the requirements of these methods are described below.

`Coordinator::register_subtran_aware()` can be used to register a resource with the subtransaction associated with the `Coordinator` object. The resource can also register with the top-level transaction by using `Coordinator::register_resource()` as well. In this case, the protocol for the `Resource` class must be defined in addition to the protocol for `SubtransactionAwareResource`. See the reference page for the `Resource` class for more information.

```
// C++
class SubtransactionAwareResource : Resource {
    public:
        virtual void commit_subtransaction(Coordinator);
        virtual void rollback_subtransaction();
};
typedef SubtransactionAwareResource
   *SubtransactionAwareResource_ptr;
class SubtransactionAwareResource_var;
```

See Also

CosTransactions::Coordinator
CosTransactions::Resource
CosTransactions::Status

Commit Subtransaction

`commit_subtransaction()` must be defined to commit all changes made to the resource as part of the subtransaction. If an ancestor transaction rolls back, the subtransaction's changes are rolled back. The transaction service invokes this method if the resource is registered with a subtransaction and it is committed.

The method must be defined to take a `Coordinator` object as its only argument. When the transaction service invokes this method, it passes the `Coordinator` object associated with the parent transaction.

Rollback Subtransaction

`rollback_subtransaction()` must be defined to undo all changes made to the resource as part of the subtransaction. The transaction service invokes this method if the resource is registered with a subtransaction and it is rolled back.

CosTransactions::Synchronization Class

The Synchronization class represents a non-recoverable object that maintains transient state data and is dependent on a recoverable object to ensure that the data is persistent. To make data persistent, a synchronization object moves its data to one or more resources before the transaction completes.

The Synchronization class specifies a protocol that must be defined for this type of object. A synchronization object must be implemented as a class derived from the Synchronization class. The derived class must implement each of the member methods to ensure that the data maintained by the nonrecoverable object is made recoverable. The transaction service invokes these methods before and after the registered resources commit; the specific requirements of these methods are described in the following sections.

Coordinator::register_synchronization() can be used to register a synchronization object with the transaction associated with the Coordinator object.

```
// C++
class Synchronization : TransactionalObject {
    public:
        virtual void before_completion();
        virtual void after_completion(Status);
};
```

Before Completion

before_completion() must be defined to move the synchronization object's data to a recoverable object. The transaction service invokes this method prior to the prepare phase of the transaction. The method is invoked only if the synchronization object is registered with a transaction and the transaction attempts to commit.

The only exceptions this method can throw are `CORBA::SystemException` exceptions. Throwing other exceptions can cause the transaction to be marked for rollback only.

After Completion

`after_completion()` must be defined to do any necessary processing required by the synchronization object; for example, the method could be used to release locks held by the transaction. The transaction service invokes this method after the outcome of the transaction is complete. The method is invoked only if the synchronization object is registered with a transaction and the transaction has either committed or rolled back.

The method must be defined to take a `Status` value as its only argument. When the transaction service invokes this method, it passes the status of the transaction with which the synchronization object is registered.

The only exceptions this method can throw are `CORBA::SystemException` exceptions. Any exceptions that are thrown have no effect on the commitment of the transaction.

See Also

`CosTransactions::Coordinator`
`CosTransactions::Coordinator::register_synchronization()`
`CosTransactions::Resource`
`CosTransactions::Status`

CosTransactions::Terminator Class

The `Terminator` class enables explicit termination of a factory-created transaction. The transaction with which the `Terminator` object is associated can be either committed or rolled back. `Control::get_terminator()` can be used to return the `Terminator` object associated with a transaction.

```
// C++
class Terminator {
public:
    void commit(CORBA::Boolean);
    void rollback();
};
typedef Terminator *Terminator_ptr;
class Terminator_var;
```

See Also

`CosTransactions::Coordinator`
`CosTransactions::Control::get_terminator()`
`CosTransactions::Control`
`CosTransactions::Status`

Terminator::commit()

```
// C++
void commit(
    CORBA::Boolean report_heuristics
)
    throw(CORBA::SystemException,
            HeuristicHazard,
            TRANSACTION_ROLLEDBACK);
```

`commit()` attempts to commit the transaction associated with the `Terminator` object. If the `report_heuristics` parameter is true, the `HeuristicHazard` exception is thrown when the participants report that a heuristic decision has possibly been made.

Parameters

`report_heuristics`	Specifies whether to report heuristic decisions for the commit.

Exceptions

`CORBA::TRANSACTION_ROLLEDBACK`	The transaction has been marked as rollback-only, or all participants in the transaction do not agree to commit.

See Also

`CosTransactions::Coordinator`
`CosTransactions::Terminator`
`CosTransactions::Terminator::rollback()`
`CosTransactions::Control`

Terminator::rollback()

```
// C++
void rollback();
```

`rollback()` rolls back the transaction associated with the `Terminator` object.

See Also

`CosTransactions::Coordinator`
`CosTransactions::Terminator`
`CosTransactions::Terminator::commit()`

CosTransactions::TransactionalObject Class

The TransactionalObject interface has been deprecated and replaced with transactional policies (see "OTSPolicyValue Data Type" on page 1048). Backward compatibility with existing OTS implementations is provided for outbound requests only and only if the target object does not have a transactional policy in its IOR.

See the *CORBA Programmer's Guide* for details of interoperability with existing OTS implementations.

```
// C++
class TransactionalObject {};
typedef TransactionalObject *TransactionalObject_ptr;
class TransactionalObject_var;
```

CosTransactions::TransactionFactory Class

The `TransactionFactory` class represents a transaction factory that allows the originator of transactions to begin a new transaction for use with the explicit model of transaction demarcation. Servers provide a default instance of this class. Clients can bind to the default instance by using the standard binding mechanism for the object request broker.

```
// C++
class TransactionFactory {
    public:
        Control_ptr create(unsigned long timeout);
        Control_ptr recreate(const PropagationContext& ctx);
};
typedef TransactionFactory *TransactionFactory_ptr;
class TransactionFactory_var;
```

See Also `CosTransactions::Control`

TransactionFactory::create()

```
// C++
Control_ptr create(unsigned long timeout)
    throw(CORBA::SystemException);
```

`create()` creates a new top-level transaction for use with the explicit model of transaction demarcation. A `Control` object is returned for the transaction. The `Control` object can be used to propagate the transaction context. See the reference page for the `Control` class for more information.

Parameters

`timeout`	Specifies the number of seconds that the transaction waits to complete before rolling back. If the `timeout` parameter is zero, no timeout is set for the transaction.

See Also CosTransactions::TransactionFactory
CosTransactions::Control

TransactionFactory::recreate()

```
// C++
Control_ptr TransactionFactory::recreate(
      const PropagationContext& ctx);
```

Creates a new representation for an existing transaction defined in the propagation context `ctx`. This is used to import a transaction from another domain. The method returns a control object for the new transaction representation.

See Also CosTransactions::Coordinator::get_txcontext()

Persistent State Service

CosPersistentState Overview

The persistent state service (PSS) is a CORBA-friendly object-oriented database. PSS storage objects can hold any kind of IDL type. The Orbix implementation of PSS is organized into three modules and an object factory class:

- "CosPersistentState Overview"

  The `CosPersistentState` module is the standard OMG service for persistent objects.
- "IT_PSS Overview"

  The `IT_PSS` module provides various proprietary useful features such as queries.
- The IT_PSS_DB Module Overview

  The Orbix implementation of PSS is targeted at relational and relational-like database back-ends. It is not restricted to any particular database system.

The CosPersistentState module's features are listed in Table 27:

Table 27: *The CosPersistentState Module*

Common Data Types	Interfaces
AccessMode Type ForUpdate Enumeration IsolationLevel Type NotFound Exception Parameter Structure ParameterList Sequence Pid Type ShortPid Type TransactionalSessionList Sequence TypeId Type YieldRef Enumeration	CatalogBase Connector EndOfAssociationCallback Session StorageHomeBase TransactionalSession
Native Types and Helper Classes	
CosPersistentState_Factory StorageHomeFactory StorageObjectBase StorageObjectFactory StorageObjectRef	

The rest of this chapter describes the common data types for the module.

CosPersistentState::AccessMode Type

```
// PSDL Code
typedef short AccessMode;
```

The mode of access for a storage object. Valid values include:

```
READ_ONLY
READ_WRITE
```

The AccessMode READ_WRITE is higher than READ_ONLY.

CosPersistentState::ForUpdate Enumeration

```
// PSDL Code
enum ForUpdate { FOR_UPDATE };
```

Used in the language mapping to define an overloaded accessor method that can update the state member.

Examples

For example, a state member whose type is an abstract storagetype is mapped to a read-only accessor, a read-write (update) accessor, and a modifier:

```
// PSDL
abstract storagetype A {};
abstract storagetype B {
    state A embedded;
};
```

This PSDL code maps to:

```
// C++
class B : public virtual StorageObject {
public:
    virtual const A& embedded() const = 0;
    virtual A& embedded(CosPersistentState::ForUpdate) = 0;
    virtual void embedded(const A&) = 0; // copies
};
```

CosPersistentState::IsolationLevel Type

```
// PSDL Code
typedef short IsolationLevel;
const IsolationLevel READ_UNCOMMITTED  = 0;
const IsolationLevel READ_COMMITTED    = 1;
const IsolationLevel REPEATABLE_READ   = 2;
const IsolationLevel SERIALIZABLE      = 3;
```

When data is accessed through a transactional session actively associated with a resource, undesirable phenomena such as dirty reads or non-repeatable reads may occur. An isolation level controls user access to these kinds of phenomenon during a transactional session.

Valid `IsolationLevel` values include the following:

`READ_UNCOMMITTED` — When a resource has this isolation level, its user may experience the dirty reads and the non-repeatable reads phenomena.

`READ_COMMITTED` — When a resource has this isolation level, its user may experience the non-repeatable reads phenomenon, but not the dirty reads phenomenon.

SERIALIZABLE	When a resource has this isolation level, its user is protected from both the dirty reads and the non-repeatable reads phenomena
REPEATABLE_READ	This isolation level is reserved for future use.

A dirty read occurs when a resource is used to read the uncommitted state of a storage object. For example, suppose a storage object is updated using resource 1. The updated storage object's state is read using resource 2 before resource 1 is committed. If resource 1 is rolled back, the data read with resource 2 is considered never to have existed.

A non-repeatable read occurs when a resource is used to read the same data twice but different data is returned by each read. For example, suppose resource 1 is used to read the state of a storage object. Resource 2 is used to update the state of this storage object and resource 2 is committed. If resource 1 is used to reread the storage object's state, different data is returned.

See Also CosPersistentState::TransactionalSession

CosPersistentState::NotFound Exception

```
// PSDL Code
exception NotFound {};
```

An exception that indicates that a storage object or registry connector cannot be found.

CosPersistentState::Parameter Structure

```
// PSDL Code
struct Parameter {
    string name;
    any val;
};
```

A parameter in a list of parameters when creating a session.

Parameters

`name`	The parameter's name.
`val`	The value in the parameter.

See Also `CosPersistentState::ParameterList`
`CosPersistentState::Connector::create_basic_session()`
`CosPersistentState::Connector::create_transactional_session()`

CosPersistentState::ParameterList Sequence

```
// PSDL Code
typedef sequence<Parameter> ParameterList;
```

A sequence of `Parameter` structures.

See Also `CosPersistentState::Parameter`

CosPersistentState::Pid Type

```
// PSDL Code
typedef CORBA::OctetSeq Pid;
```

A global persistent object identifier that storage objects use. The scope of the `Pid` is all storage objects that can be accessed through the same catalog.

See Also `CosPersistentState::ShortPid`

CosPersistentState::ShortPid Type

```
// PSDL Code
typedef CORBA::OctetSeq ShortPid;
```

A storage object identifier that is unique within a storage home family.

See Also `CosPersistentState::Pid`

CosPersistentState::TransactionalSessionList Sequence

```
// PSDL Code
```

```
typedef sequence<TransactionalSession> TransactionalSessionList;
```

A list of transactional sessions.

See Also CosPersistentState::TransactionalSession
CosPersistentState::Connector::sessions()

CosPersistentState::TypeId Type

```
// PSDL Code
typedef string TypeId;
```

A string that identifies a PSDL type. The format of a PSDL type id is the same as the IDL format of repository ids, except that the prefix is `PSDL`, not `IDL`.

See Also CORBA::RepositoryId
CosPersistentState::Connector

CosPersistentState::YieldRef Enumeration

```
// PSDL Code
enum YieldRef { YIELD_REF };
```

Used in the language mapping to define overloaded methods that yield incarnations and references as parameters.

Examples For example, a state member whose type is a reference to an abstract storagetype is mapped to two accessors and two modifier methods:

```
// PSDL
abstract storagetype Bank;
abstract storagetype Account {
    state ref<Bank> my_bank;
};
```

The mapping shows that one of the accessor methods takes no parameter and returns a storage object incarnation, and the other takes a `YieldRef` parameter and returns a reference:

```
// C++
class Account : public virtual StorageObject {
public:
    virtual Bank* my_bank() const= 0;
    virtual const BankRef* my_bank(
```

```
            CosPersistentState::Yield-Ref yr
        ) const = 0;

        virtual void my_bank(Bank* b) = 0;
        virtual void my_bank(const BankRef* b) = 0;
    };
```

CosPersistentState::CatalogBase Interface

The CatalogBase interface is the base interface for the implementation of a local catalog object.

```
// PSDL in module CosPersistentState
local interface CatalogBase {

    readonly attribute AccessMode access_mode;

    StorageHomeBase find_storage_home(
        in string storage_home_type_id
    )
        raises (NotFound);

    StorageObjectBase find_by_pid(
        in Pid the_pid
    )
        raises (NotFound);

    void flush();
    void refresh();
    void free_all();
    void close();
};
```

CatalogBase::access_mode Attribute

```
// PSDL code
readonly attribute AccessMode access_mode;
```

Returns the access mode of this catalog. When the access mode is READ_ONLY, the storage object incarnations obtained through storage home instances provided by this catalog are read-only.

CatalogBase::close()

```
// PSDL code
void close();
```

Terminates the catalog. If the catalog is associated with one or more transactions when `close()` is called, these transactions are marked as roll-back only. When closed, the catalog is also flushed for a non-transactional session.

CatalogBase::find_by_pid()

```
// PSDL code
StorageObjectBase find_by_pid(
    in Pid the_pid
)
    raises (NotFound);
```

Attempts to locate a storage object and returns an incarnation of it.

Parameters

`the_pid`	The operation uses the given `Pid` to find the storage object in the storage homes provided by the target catalog.

Exceptions

`NotFound exception`	The operation cannot find a storage object with this `Pid`.

CatalogBase::find_storage_home()

```
// PSDL code
StorageHomeBase find_storage_home(
    in string storage_home_type_id
)
    raises (NotFound);
```

Returns a storage home instance.

Parameters

`storage_home_type_id`	The operation looks up a PSDL-defined storage home with this Id in the catalog's default data-store. The format of this parameter is mostly implementation-defined. In the case of type-specific catalogs (declared in PSDL), the provided declarations define valid values for this parameter. The operation can also interpret Ids that have the form of a PSDL type Id. For example: `PSDL:com/acme/PersonStoreImpl:1.0`

Exceptions

`NotFound`	The operation cannot find a storage home that matches the given storage home Id.

CatalogBase::flush()

```
// PSDL code
void flush();
```

Writes to disk any cached modifications of storage object incarnations managed by this catalog. PSS can cache some *dirty* data, thus, when an application creates a new storage object or updates a storage object, the modification is not written directly to disk.

CatalogBase::free_all()

```
// PSDL code
void free_all();
```

Instructs the catalog implementation to set the reference count of all its PSDL storage object instances to 0.

CatalogBase::refresh()

```
// PSDL code
void refresh();
```

Refreshes any cached storage object incarnations accessed (read) by this catalog. In addition to caching write data, PSS can cache data read from datastores.

Note: This operation can invalidate any direct reference to a storage object incarnation's data member. Most applications do not use `refresh()`, so calling it is unusual.

CosPersistentState::Connector Interface

A connector is a local object that represents a given PSS implementation. Sessions are created by connectors. You obtain a connector of a given ORB by calling CORBA::ORB::resolve_initial_references() with the ObjectId of PSS.

```
// PSDL code in module CosPersistentState
local interface Connector {

    readonly attribute string implementation_id;

    Pid get_pid(
        in StorageObjectBase obj
    );

    ShortPid get_short_pid(
        in StorageObjectBase obj
    );

    Session create_basic_session(
        in AccessMode access_mode,
        in TypeId catalog_type_name,
        in ParameterList additional_parameters
    );

    TransactionalSession create_transactional_session(
        in AccessMode access_mode,
        in IsolationLevel default_isolation_level,
        in EndOfAssociationCallback callback,
        in TypeId catalog_type_name,
        in ParameterList additional_parameters
    );

    TransactionalSession current_session();

    TransactionalSessionList sessions(
```

```
        in CosTransactions::Coordinator transaction
    );

    StorageObjectFactory register_storage_object_factory(
        in TypeId storage_type_name,
        in StorageObjectFactory storage_object_factory
    );

    StorageHomeFactory register_storage_home_factory(
        in TypeId storage_home_type_name,
        in StorageHomeFactory storage_home_factory
    );

    SessionFactory register_session_factory(
        in TypeId catalog_type_name,
        in SessionFactory session_factory
    );

    SessionPoolFactory register_session_pool_factory(
        in TypeId catalog_type_name,
        in SessionPoolFactory session_pool_factory
    );

};
```

Connector::create_basic_session()

```
// PSDL code
Session create_basic_session(
    in AccessMode access_mode,
    in TypeId catalog_type_name,
    in ParameterList additional_parameters
);
```

Creates a basic, non-transactional session and returns a reference to the session.

Parameters

`access_mode`	The access can be read-only or both read and write.
`catalog_type_name`	The value is either an empty string or the PSDL type id of a catalog. For example: `PSDL:com/acme/People:1.0`.
`additional_parameters`	See Table 28.

Table 28: *Additional PSS Session Creation Parameters*

Parameter Name	Type	Description
`to`	`string`	This is a required parameter. Some string that identifies what you connect to. For example with PSS/DB, it will be an environment name; with PSS/ODBC a datasource name; with PSS/Oracle, an Oracle database name.
`concurrent`	`boolean`	Will this session be used by multiple concurrent threads? This parameter is not required. The default value is false.
`single writer`	`boolean`	Is this session the only session that writes to this database? When true, there is no risk of deadlock and the cache can be kept as-is after a commit. This parameter is not required. The default value is false.
Additional Relational Parameters		
`pessimistic locking`	`boolean`	Does this session acquire a write lock before updating an object in its cache? The default value is true. This parameter is not required.
`incarnation map size`	`long`	The size of the per-session hash map in which PSS/R keeps incarnations. The given value is rounded up to the closest power of 2. The default value is `1024`. This parameter is not required.

Exceptions

PERSIST_STORE	A session cannot be provided with the desired (or higher) access mode.

See Also CosPersistentState::Connector::create_transactional_session()

Connector::create_transactional_session()

```
// PSDL code
TransactionalSession create_transactional_session(
    in AccessMode access_mode,
    in IsolationLevel default_isolation_level,
    in EndOfAssociationCallback callback,
    in TypeId catalog_type_name,
    in ParameterList additional_parameters
);
```

Creates a new transactional session and returns a reference to the session.

Parameters

access_mode	The access can be read-only or both read and write.
default_isolation_level	The isolation level of resources created by this transactional session.
callback	Your application can be notified when a session is released by PSS by passing in an EndOfAssociationCallback local object.
catalog_type_name	The value is either an empty string or the PSDL type id of a catalog. For example: PSDL:com/acme/People:1.0.
additional_parameters	See Table 29.

Table 29: *Additional PSS TransactionalSession Creation Parameters*

Parameter Name	Type	Description
`to`	`string`	This is a required parameter. Some string that identifies what you connect to. For example with PSS/DB, it will be an environment name; with PSS/ODBC a datasource name; with PSS/Oracle, an Oracle database name.
`concurrent`	`boolean`	Will this session be used by multiple concurrent threads? This parameter is not required. The default value is false.
`single writer`	`boolean`	Is this session the only session that writes to this database? When true, there is no risk of deadlock and the cache can be kept as-is after a commit. This parameter is not required. The default value is false.

Exceptions

`PERSIST_STORE` Raised if:

- The session cannot be provided with the desired (or higher) access mode.
- The implementation cannot provide the desired default isolation level.

See Also `CosPersistentState::Connector::create_basic_session()`

Connector::current_session()

```
// PSDL code
TransactionalSession current_session();
```

Returns the current transactional session. The operation logically calls `sessions()` with the transaction associated with the calling thread.

Exceptions

`PERSIST_STORE` A single session cannot be returned.

See Also CosPersistentState::Connector::sessions()

Connector::get_pid()

```
// PSDL code
Pid get_pid(
    in StorageObjectBase obj
);
```

Returns the Pid of the given storage object.

See Also CosPersistentState::Connector::get_short_pid()

Connector::get_short_pid()

```
// PSDL code
ShortPid get_short_pid(
    in StorageObjectBase obj
);
```

Returns the ShortPid of the given storage object.

See Also CosPersistentState::Connector::get_pid()

Connector::implementation_id Attribute

```
// PSDL code
readonly attribute string implementation_id;
```

Returns the Id of this implementation.

Connector::register_session_factory()

```
// PSDL code
SessionFactory register_session_factory(
     in TypeId catalog_type_name,
     in SessionFactory session_factory
);
```

Registers a session factory and returns the factory previously registered with the given name. The operation returns NULL when there is no previously registered factory.

See Also

CosPersistentState::Connector::register_storage_object_factory()
CosPersistentState::Connector::register_storage_home_factory()
CosPersistentState::Connector::register_session_pool_factory()

Connector::register_session_pool_factory()

```
// PSDL code
SessionPoolFactory register_session_pool_factory(
     in TypeId catalog_type_name,
     in SessionPoolFactory session_pool_factory
);
```

Registers session pool factories and returns the factory previously registered with the given name. The operation returns NULL when there is no previously registered factory.

See Also

CosPersistentState::Connector::register_storage_object_factory()
CosPersistentState::Connector::register_storage_home_factory()
CosPersistentState::Connector::register_session_factory()

Connector::register_storage_home_factory()

```
// PSDL code
StorageHomeFactory register_storage_home_factory(
     in TypeId storage_home_type_name,
     in StorageHomeFactory storage_home_factory
);
```

Registers storage home factories and returns the factory previously registered with the given name. The operation returns NULL when there is no previously registered factory.

See Also

CosPersistentState::Connector::register_storage_object_factory()
CosPersistentState::Connector::register_session_factory()
CosPersistentState::Connector::register_session_pool_factory()

Connector::register_storage_object_factory()

```
// PSDL code
StorageObjectFactory register_storage_object_factory(
     in TypeId storage_type_name,
     in StorageObjectFactory storage_object_factory
);
```

Registers storage object factories and returns the factory previously registered with the given name. The operation returns NULL when there is no previously registered factory.

See Also

CosPersistentState::Connector::register_storage_home_factory()
CosPersistentState::Connector::register_session_factory()
CosPersistentState::Connector::register_session_pool_factory()

Connector::sessions()

```
// PSDL code
TransactionalSessionList sessions(
    in CosTransactions::Coordinator transaction
);
```

Returns all the transactional sessions created by this connector that are associated with resources registered with the given transaction. Very often `sessions()` returns a single session.

See Also

CosPersistentState::Connector::current_session()

CosPersistentState:: EndOfAssociationCallback Interface

The `EndOfAssociationCallback` interface is implemented by the developer of the application. When a session-resource association has ended, the session may not become available immediately. For example, if the session is implemented using an ODBC or JDBC connection, PSS needs this connection until the resource (ODBC/JDBC transaction) is committed or rolled back.

```
// PSDL code in module CosPersistentState
local interface EndOfAssociationCallback {
    void released(in TransactionalSession session);
};
```

See Also `CosPersistentState::Connector::create_transactional_session()`

CosPersistentState_Factory Template

The `CosPersistentState_Factory` class is a helper template you use to build `StorageHomeFactory` and `StorageObjectFactory` objects. The class contains the following virtual methods.

```
template <class T>
class CosPersistentState_Factory {
  public:

    virtual T* create()
        throw(CORBA::SystemException) = 0;

    virtual void _add_ref() {}

    virtual void _remove_ref() {}

    virtual ~CosPersistentState_Factory() {}
};
```

CosPersistentState::Session Interface

A PSS session is a logical connection between a process and one or more datastores. There are two kinds of sessions:

- Basic sessions for file-like access.
- Transactional sessions for transactional access. (See the `TransactionalSession` interface.)

You create a basic session by calling `Connector::create_basic_session()`. A basic session is a local object that supports the following interface:

```
// PSDL Code in module CosPersistentState
local interface Session : CatalogBase {};
```

See Also `IT_PSS::Session`

CosPersistentState:: StorageHomeBase Interface

A storage home can have behavior that is described by operations on its abstract storage home(s). An abstract storage home can also define any number of keys; each key declaration implicitly declares a pair of finder operations. All storage home instances implement the local interface `StorageHomeBase`:

```
// PSDL in module CosPersistentState
local interface StorageHomeBase {

    StorageObjectBase find_by_short_pid(
        in ShortPid short_pid
    )
    raises (NotFound);

    CatalogBase get_catalog();
};
```

StorageHomeBase::find_by_short_pid()

```
// PSDL code
StorageObjectBase find_by_short_pid(
    in ShortPid short_pid
)
    raises (NotFound);
```

Returns a storage object for the given short pid.

Parameters

`short_pid`	The short pid in the target storage home.

Exceptions

CosPersistentState::NotFound	The object is not found.

StorageHomeBase::get_catalog()

```
// PSDL code
    CatalogBase get_catalog();
```

Returns the catalog that manages the target storage home instance.

CosPersistentState:: StorageHomeFactory Native Type

The `StorageHomeFactory` is a native PSDL type.

```
// PSDL in module CosPersistentState
native StorageHomeFactory;
```

The C++ mapping of this native type is as follows:

```
// C++
typedef CosPersistentState_Factory<StorageHomeBase>
   StorageHomeFactory;
```

The application developer derives a class from this `StorageHomeFactory` type to provide an implementation.

See Also

`IT_PSS_StorageHomeFactory`
`CosPersistentState::CosPersistentState_Factory`

CosPersistentState::StorageObject Interface

The `StorageObject` interface supports a PSS storage object.

```
// PSDL in module CosPersistentState
abstract storagetype StorageObject {
    void destroy_object();
    boolean object_exists();
    Pid get_pid();
    ShortPid get_short_pid();
    StorageHomeBase get_storage_home();
};
```

StorageObject::destroy_object()

```
// PSDL code
void destroy_object();
```

When called on an incarnation, the operation destroys the associated storage object (but does not destroy any of its incarnation).

Exceptions

`PERSIST_STORE` The operation is called on the instance of an embedded storage object.

StorageObject::get_pid()

```
// PSDL code
Pid get_pid();
```

Returns the `Pid` of the associated storage object when called on an incarnation.

Exceptions

PERSIST_STORE The operation is called on the instance of an embedded storage object.

See Also CosPersistentState::StorageObject::get_short_pid()

StorageObject::get_short_pid()

```
// PSDL code
ShortPid get_short_pid();
```

Returns the ShortPid of the associated storage object when called on an incarnation.

Exceptions

PERSIST_STORE The operation is called on the instance of an embedded storage object.

See Also CosPersistentState::StorageObject::get_pid()

StorageObject::get_storage_home()

```
// PSDL code
StorageHomeBase get_storage_home();
```

Returns the storage home instance that manages the target storage object instance.

StorageObject::object_exists()

```
// PSDL code
boolean object_exists();
```

Returns true if the target incarnation represents an actual storage object and false if it does not.

CosPersistentState:: StorageObjectBase Native Type

A storage object can have both state and behavior. The visible part of its state is described by state members on its abstract storage type(s). Similarly, its behavior is described by operations on its abstract storage type(s).

All storage object instances are derived from this common base, `StorageObjectBase`:

```
// PSDL in module CosPersistentState
    native StorageObjectBase;
```

The C++ mapping of this native type is as follows:

```
class StorageObjectBase {
protected:
    virtual ~StorageObjectBase() {}
};
```

CosPersistentState:: StorageObjectFactory Native Type

`StorageObjectFactory` is a native type.

```
// in module CosPersistentState
native StorageObjectFactory;
```

The C++ mapping of this native type is as follows:

```
// C++
typedef CosPersistentState_Factory<StorageObject>
   StorageObjectFactory;
```

The application developer derives a class from this `StorageObjectFactory` type to provide an implementation.

See Also

`IT_PSS_StorageObjectFactory`
`CosPersistentState::CosPersistentState_Factory`

CosPersistentState::StorageObjectRef Class

The StorageObjectRef class is a standard C++ base class mapping for a StorageObject reference.

```
class StorageObjectRef {
  public:
    typedef StorageObject _target_type;

    static CORBA::TypeCode_ptr _static_type();

    StorageObjectRef(
        StorageObject*  obj       = 0,
        CatalogBase_ptr catalog   = 0,
        void*           impl_data = 0
    );
    StorageObjectRef(
        const StorageObjectRef& ref
    );

    StorageObjectRef&  operator=(
        const StorageObjectRef& ref
    );

    StorageObjectRef&  operator=(
        StorageObject* obj
    );

    void release();

    StorageObject* operator->(); // not const!

    CORBA::Boolean same_ref(
        StorageObjectRef
    ) const;

    void destroy_object() const;
```

```
    Pid* get_pid() const;

    ShortPid* get_short_pid() const;

    CORBA::Boolean is_null() const;

    StorageHomeBase_ptr get_storage_home() const;


    // read-only access to data members

    void* _impl_data() const;

    CosPersistentState::CatalogBase_ptr _catalog() const;

    StorageObject* _target() const;

  protected:
    CosPersistentState::CatalogBase_ptr m_catalog;
    void*                               m_impl_data;
    StorageObject*                      m_target;
};
```

StorageObjectRef::_catalog()

`CosPersistentState::CatalogBase_ptr _catalog() const;`

Returns the catalog of the object.

StorageObjectRef::destroy_object()

`void destroy_object() const;`

Destroys the target object.

StorageObjectRef::get_pid()

```
Pid* get_pid() const;
```

Returns the Pid of the target object.

StorageObjectRef::get_short_pid()

```
ShortPid* get_short_pid() const;
```

Returns the short pid of the target object.

StorageObjectRef::get_storage_home()

```
StorageHomeBase_ptr get_storage_home() const;
```

Returns the storage home of the target object.

StorageObjectRef::_impl_data()

```
void* _impl_data() const;
```

StorageObjectRef::is_null()

```
CORBA::Boolean is_null() const;
```

Returns true if and only if this reference is null.

StorageObjectRef::operator=()

```
StorageObjectRef&  operator=(
    const StorageObjectRef& ref
);
```

An assignment operator that takes an incarnation of the target abstract storage type.

```
StorageObjectRef&  operator=(
    StorageObject* obj
);
```

An assignment operator.

StorageObjectRef::operator->()

```
StorageObject* operator->(); // not const!
```

A de-reference operator that de-references this reference and returns the target object. The caller is not supposed to release this incarnation.

StorageObjectRef::release()

```
void release();
```

Releases the reference.

StorageObjectRef::same_ref()

```
CORBA::Boolean same_ref(
    StorageObjectRef
) const;
```

Returns true if the input storage object reference is the same as this one.

StorageObjectRef::_static_type()

```
static CORBA::TypeCode_ptr _static_type();
```

Returns a TypeCode reference.

StorageObjectRef::StorageObjectRef() Constructors

```
StorageObjectRef(
    StorageObject*  obj       = 0,
    CatalogBase_ptr catalog   = 0,
```

```
    void*           impl_data = 0
);
```

The default constructor creates a null reference.

```
StorageObjectRef(
    const StorageObjectRef& ref
);
```

A non-explicit constructor that takes an incarnation of the target abstract storage type.

StorageObjectRef::_target_type

```
typedef StorageObject _target_type;
```

A type definition to the target type. This is useful for programming with templates.

StorageObjectRef::_target()

```
StorageObject* _target() const;
```

Returns the target object.

CosPersistentState:: TransactionalSession Interface

A transactional session is a specialized session that provides transactional access to storage objects. A transactional session supports the local interface `TransactionalSession`.

At a given time, a transactional session can be associated with one resource object (a datastore transaction), or with no resource at all. The session-resource association can be active, suspended, or ending. The state members of an incarnation managed by a transactional session can be used only when this session has an active association with a resource.

Typically, a resource is associated with a single session for its entire lifetime. However, with some advanced database products, the same resource may be associated with several sessions, possibly at the same time.

You create a transaction session by calling `create_transactional_session()`.

```
// PSDL Code in module CosPersistentState
local interface TransactionalSession : Session {

    readonly attribute IsolationLevel default_isolation_level;

    typedef short AssociationStatus;
    const AssociationStatus NO_ASSOCIATION = 0;
    const AssociationStatus ACTIVE         = 1;
    const AssociationStatus SUSPENDED      = 2;
    const AssociationStatus ENDING         = 3;

    void start(in CosTransactions::Coordinator transaction);
    void suspend(in CosTransactions::Coordinator transaction);
    void end(
        in CosTransactions::Coordinator transaction,
        in boolean success
    );
```

```
    AssociationStatus get_association_status();

    CosTransactions::Coordinator transaction();
    };
```

See Also IT_PSS::TransactionalSession
CosPersistentState::Session

TransactionalSession::AssociationStatus Type

```
// PSDL Code
typedef short AssociationStatus;
const AssociationStatus NO_ASSOCIATION = 0;
const AssociationStatus ACTIVE         = 1;
const AssociationStatus SUSPENDED      = 2;
const AssociationStatus ENDING         = 3;
```

The association status of a resource with a session. Valid values include:

NO_ASSOCIATION
ACTIVE
SUSPENDED
ENDING

See Also CosPersistentState::TransactionalSession::
get_association_status()

TransactionalSession::default_isolation_level Attribute

```
// PSDL Code
readonly attribute IsolationLevel default_isolation_level;
```

Returns the default isolation level of resources created for this transactional session.

TransactionalSession::end()

```
// PSDL Code
void end(
    in CosTransactions::Coordinator transaction,
    in boolean success
```

```
);
```

Terminates a session-transaction association.

Parameters

`transaction`	The transaction of the resource.
`success`	If the success parameter is FALSE, the resource is rolled back immediately. Like `refresh()`, `end()` invalidates direct references to incarnations' data members.

A resource can be prepared or committed in one phase only when it is not actively associated with any session. The resource will rollback if it is asked to prepare or commit in one phase when still in use. A resource ends any session-resource association in which it is involved when it is prepared, committed in one phase, or rolled back.

Exceptions

`PERSIST_STORE`	No associated resource.
`INVALID_TRANSACTION`	The given transaction does not match the transaction of the resource associated with this session.

The standard exception is raised if

See Also `CosPersistentState::TransactionalSession::start()`
`CosPersistentState::TransactionalSession::suspend()`

TransactionalSession::get_association_status()

```
// PSDL Code
AssociationStatus get_association_status();
```

Returns the status of the association (if any) with this session.

TransactionalSession::start()

```
// PSDL Code
void start(
    in CosTransactions::Coordinator transaction
);
```

Starts the transaction.

Parameters

`transaction` The transaction to start.

This operation does one of three things depending on the association of the transaction:

1. When `transaction` matches the transaction of the suspended (or ending) association, `start()` re-activates a suspended (or ending) session-resource association.
2. If a resource compatible with this session is already associated with the given transaction, `start()` associates this resource with this session, and makes the association active.
3. If the session creates a new resource and registers it with the given transaction. The session also associates itself with this resource and makes the association active.

Exceptions

`INVALID_TRANSACTION` There is a suspended (or ending) association but the transactions do not match.

See Also `CosPersistentState::TransactionalSession::suspend()`
`CosPersistentState::TransactionalSession::end()`

TransactionalSession::suspend()

```
// PSDL Code
void suspend(
    in CosTransactions::Coordinator transaction
);
```

Suspends a session-resource association.

Parameters

`transaction` The transaction to suspend.

Exceptions

`PERSIST_STORE` No active association.

The standard exception `INVALID_TRANSACTION` is raised if the given transaction does not match the transaction of the resource actively associated with this session.

See Also `CosPersistentState::TransactionalSession::start()`
`CosPersistentState::TransactionalSession::end()`

TransactionalSession::transaction()

```
// PSDL Code
CosTransactions::Coordinator transaction();
```

Returns the coordinator of the transaction with which the resource associated with this session is registered. The operation returns a nil object reference when the session is not associated with a resource.

IT_PSS Overview

The IT_PSS interfaces consist of:

CatalogBase
Connector
PreparedStatement
Master
Replica
ResultSet
Session
SessionManager
Statement
StorageObject
TransactionalSession

This module also has the following helper classes:

IT_PSS_StorageHomeFactory
IT_PSS_StorageObjectFactory
TxSessionAssociation

IT_PSS::CatalogBase Interface

PSS provides simple JDBC-like queries. You use `CatalogBase` to create a `Statement` or `PreparedStatement`.The query language is a subset of SQL that currently only supports the following form of select query:

```
select ref(h) from home_type_id h
```

The PSDL code is as follows:

```
// PSDL Code in Module IT_PSS
local interface CatalogBase :
CosPersistentState::CatalogBase {

    Statement it_create_statement();

    Statement it_create_statement_with_type_and_concurrency(
        in ResultSet::Type type,
        in ResultSet::Concurrency concurrency
    );

    PreparedStatement it_prepare_statement(
        in string pssql
    );

    PreparedStatement
    it_prepare_statement_with_type_and_concurrency(
        in string pssql,
        in ResultSet::Type type,
        in ResultSet::Concurrency concurrency
    );

    void it_discard_flush_list();

    void it_discard_all(
        in boolean clear_non_id_refs
    );
};
```

Enhancement This is an Orbix enhancement.

See Also

CosPersistentState::CatalogBase
IT_PSS::PreparedStatement
IT_PSS::Statement
IT_PSS::ResultSet

CatalogBase::it_create_statement()

```
// PSDL Code
Statement it_create_statement();
```

Creates and returns a JDBC-like Statement.

Enhancement This is an Orbix enhancement.

CatalogBase:: it_create_statement_with_type_and_concurrency()

```
// PSDL Code
Statement it_create_statement_with_type_and_concurrency(
    in ResultSet::Type type,
    in ResultSet::Concurrency concurrency
);
```

Creates and returns a JDBC-like Statement with a specific ResultSet type. The concurrency setting can be either read-only or updateable. Only one ResultSet per Statement can be open at any point in time. All statement execute methods implicitly close a statement's current ResultSet if an open one exists.

Enhancement This is an Orbix enhancement.

CatalogBase::it_discard_all()

```
// PSDL Code
void it_discard_all(
    in boolean clear_non_id_refs
);
```

Discards all cached objects.

Parameters

`clear_non_id_refs`	If this parameter is set to true, any references that an object might have to another object are removed. This removes the possibility of circular references between objects.

Enhancement This is an Orbix enhancement.

CatalogBase::it_discard_flush_list()

```
// PSDL Code
void it_discard_flush_list();
```

Discards all modified objects in the catalog.

Enhancement This is an Orbix enhancement.

CatalogBase::it_prepare_statement()

```
// PSDL Code
PreparedStatement it_prepare_statement(
    in string pssql
);
```

Creates and returns a JDBC-like `PreparedStatement` with the given query.

Enhancement This is an Orbix enhancement.

CatalogBase:: it_prepare_statement_with_type_and_concurrency()

```
// PSDL Code
PreparedStatement  it_prepare_statement_with_type_and_concurrency(
    in string pssql,
    in ResultSet::Type type,
    in ResultSet::Concurrency concurrency
);
```

Creates and returns a JDBC-like `PreparedStatement` with a given query and specific `ResutSet` type. The concurrency setting can be either read-only or updateable.

Enhancement This is an Orbix enhancement.

IT_PSS::Connector Interface

This is an Orbix-enhancement interface that lets you create a session manager.

```
// PSDL Code in module IT_PSS
/* local */ interface Connector : CosPersistentState::Connector {
    SessionManager it_create_session_manager(
        in CosPersistentState::ParameterList parameters
    );
};
```

Enhancement This is an Orbix enhancement.

See Also CosPersistentState::Connector

Connector::it_create_session_manager()

```
// PSDL Code
SessionManager it_create_session_manager(
    in CosPersistentState::ParameterList parameters
);
```

Creates and returns a session manager.

Parameters

`parameters`	See Table 30 for details about possible parameters. Other parameters are passed in each session creation call. You cannot, however, pass a parameter named `concurrent` when creating a session manager. The session manager's read-only read-committed session is created with concurrent set to true, whereas the session manager's read-write serializable sessions are created with concurrent set to false.

Table 30: *Additional PSS SessionManager Creation Parameters*

Parameter Name	Type	Description
`to`	`string`	This parameter is required. Some string that identifies what you connect to. For example with PSS/DB, it will be an environment name; with PSS/ODBC a datasource name; with PSS/Oracle, an Oracle database name.
`rw pool size`	`long`	Initial size of the pool of read-write transactional sessions managed by the session manager. Must be between 1 and 1000. This parameter is not required. The default value is 1.
`grow pool`	`boolean`	Create a new session to process a new request when all the read-write transactional sessions are busy? If false, wait until a read-write transactional session becomes available. This parameter is not required. The default value is false.
`single writer`	`boolean`	Can be true only when rw pool size is 1, in which case the read-write transactional session will be created with the single writer parameter set to true. This parameter is not required. The default value is false.

Enhancement This is an Orbix enhancement.

See Also `IT_PSS::SessionManager`

IT_PSS::PreparedStatement Interface

The `PreparedStatement` interface is a JDBC-like prepared statement which is an object that represents a pre-compiled SQL statement. An SQL statement is pre-compiled and stored in the `PreparedStatement` object so your application can then efficiently execute the statement multiple times.

```
// PSDL Code in module IT_PSS
local interface PreparedStatement : Statement {

    void execute_prepared();

    ResultSet execute_prepared_query();

    unsigned long execute_prepared_update();

    void define_parameter(
        in unsigned short parameter_index,
        in any parameter_value
    );

    void clear_parameters();
};
```

Enhancement This is an Orbix enhancement.

See Also IT_PSS::CatalogBase
IT_PSS::Statement

PreparedStatement::clear_parameters()

```
// PSDL Code
void clear_parameters();
```

Clears the current parameter values immediately.

Enhancement This is an Orbix enhancement.

PreparedStatement::define_parameter()

```
// PSDL Code
void define_parameter(
    in unsigned short parameter_index,
    in any parameter_value
);
```

Defines an SQL parameter value for the designated parameter index.

Enhancement This is an Orbix enhancement.

PreparedStatement::execute_prepared()

```
// PSDL Code
void execute_prepared();
```

Executes the prepared SQL statement.

Enhancement This is an Orbix enhancement.

See Also IT_PSS::PreparedStatement::execute_prepared_query()
IT_PSS::PreparedStatement::execute_prepared_update()

PreparedStatement::execute_prepared_query()

```
// PSDL Code
ResultSet execute_prepared_query();
```

Executes the SQL query in this `PreparedStatement` object and returns the result set generated by the query.

Enhancement This is an Orbix enhancement.

See Also IT_PSS::PreparedStatement::execute_prepared()
IT_PSS::PreparedStatement::execute_prepared_update()

PreparedStatement::execute_prepared_update()

```
// PSDL Code
unsigned long execute_prepared_update();
```

Executes the SQL INSERT, UPDATE or DELETE statement in this `PreparedStatement` object.

Enhancement This is an Orbix enhancement.

See Also `IT_PSS::PreparedStatement::execute_prepared()`
`IT_PSS::PreparedStatement::execute_prepared_query()`

IT_PSS:Master Interface

The Master interface provides functionality for master instances of replicated persistent objects using the persistent state service.

```
interface Master
{};
```

IT_PSS:Replica Interface

The Replica interface provides functionality for replicated databases. The persistent state service supports two styles of replicas: a push-style replica and a pull-style replcia. A push-style replica is updated by the master instance of the object. A pull-style replica requests updates periodically from the master instance of the object.

```
interface Replica
{
  boolean set_master(in Master new_master);

  readonly attribute unsigned long long last_successful_refresh;

  // Pull refresh now
  void refresh();
};
```

IT_PSS::Replica::set_master

```
boolean set_master(in Master new_master)
```

Registers the replica with a master instance of the object. It returns TRUE if the registration is successful.

Parameters This function takes an object of Master containing an object reference to the master instance of the object.

IT_PSS::Replica::last_successful_refresh

```
readonly attribute unsigned long long last_successful_refresh
```

Returns the amount of time that has passed since the last time the replica was successfully refreshed by the master instance of the object.

IT_PSS:Replica:refresh

```
void refresh()
```

Requests an update from the master instance of the object. The master will completely sync the replica as a result of this call.

IT_PSS::ResultSet Interface

The ResultSet interface provides access to a table of data similar to a JDBC result set. A ResultSet object is usually generated by executing a Statement or a PreparedStatement. A ResultSet maintains a cursor pointing to its current row of data. Initially the cursor is positioned before the first row.

Data types include:

Concurrency Type
FetchDirection Type
Type

Operations include:

absolute()	get_fetch_size()	next()
after_last()	get_row()	previous()
before_first()	get_statement()	refresh_row()
cancel_row_updates()	get_type()	relative()
close()	insert_row()	row_deleted()
delete_row()	is_after_last()	row_inserted()
find_state_member()	is_before_first()	row_updated()
first()	is_first()	set()
get()	is_last()	set_by_name()
get_by_name()	last()	set_fetch_direction()
get_concurrency()	move_to_current_row()	set_fetch_size()
get_fetch_direction()	move_to_insert_row()	update_row()

Enhancement This interface is an Orbix enhancement.

See Also IT_PSS::CatalogBase

```
// PSDL Code in module IT_PSS
local interface ResultSet {

    typedef unsigned short Type;
    const Type TYPE_FORWARD_ONLY =        1;
    const Type TYPE_SCROLL_INSENSITIVE =  2;
    const Type TYPE_SCROLL_SENSITIVE =    3;
```

```
typedef unsigned short Concurrency;
const Concurrency CONCUR_READ_ONLY = 1;
const Concurrency CONCUR_UPDATABLE = 2;

typedef unsigned short FetchDirection;
const FetchDirection FETCH_FORWARD = 1;
const FetchDirection FETCH_REVERSE = 2;
const FetchDirection FETCH_UNKNOWN = 3;

Statement get_statement();

// Basic operations
//
boolean next();
void close();

any get(
    in unsigned short index
);

any get_by_name(
    in string state_member_name
);

// Find state_member
//
unsigned short find_state_member(
   in string state_member_name
);

// Getting/setting the current row
//
boolean is_after_last();
boolean is_before_first();
boolean is_first();
boolean is_last();
void after_last();
void before_first();
boolean first();
boolean last();
unsigned short get_row();

boolean absolute(
```

```
    in short row
);

boolean relative(
    in short rows
);

boolean previous();
void move_to_insert_row();
void move_to_current_row();

// Fetch direction and size
//
void set_fetch_direction(
    in FetchDirection direction
);

FetchDirection get_fetch_direction();

void set_fetch_size(
    in unsigned short fetch_size
);

unsigned short get_fetch_size();

// Type and Concurrency
//
Type get_type();
Concurrency get_concurrency();

// Was row modified?
//
boolean row_updated();
boolean row_inserted();
boolean row_deleted();

// Write operations
//
void set(
    in unsigned short index,
    in any value
);
```

```
    void set_by_name(
        in string state_member_name,
        in any value
    );

    void insert_row();
    void update_row();
    void delete_row();
    void refresh_row();
    void cancel_row_updates();
};
```

ResultSet::absolute()

```
// PSDL Code
boolean absolute(
    in short row
);
```

Moves the cursor to the given row number in the result set.

Parameters

`row`

If the row number is positive, the cursor moves to the given row number with respect to the beginning of the result set. The first row is row 1, the second is row 2, and so on.

If the given row number is negative, the cursor moves to an absolute row position with respect to the end of the result set. For example, calling absolute(-1) positions the cursor on the last row, absolute(-2) indicates the next-to-last row, and so on.

An attempt to position the cursor beyond the first/last row in the result set leaves the cursor before/after the first/last row, respectively.

Enhancement This is an Orbix enhancement.

ResultSet::after_last()

```
// PSDL Code
void after_last();
```

Moves the cursor to the end of the result set, just after the last row. Has no effect if the result set contains no rows.

Enhancement This is an Orbix enhancement.

ResultSet::before_first()

```
// PSDL Code
void before_first();
```

Moves the cursor to the front of the result set, just before the first row. Has no effect if the result set contains no rows.

Enhancement This is an Orbix enhancement.

ResultSet::cancel_row_updates()

```
// PSDL Code
void cancel_row_updates();
```

Cancels the updates made to a row in the table.

Enhancement This is an Orbix enhancement.

ResultSet::close()

```
// PSDL Code
void close();
```

Releases this `ResultSet` object's database and JDBC resources immediately instead of waiting for this to happen when it is automatically closed.

Enhancement This is an Orbix enhancement.

ResultSet::Concurrency Type

```
// PSDL Code
typedef unsigned short Concurrency;
const Concurrency CONCUR_READ_ONLY = 1;
const Concurrency CONCUR_UPDATABLE = 2;
```

The concurrency mode of the table. It can be read-only or updated.

Enhancement This is an Orbix enhancement.

ResultSet::delete_row()

```
// PSDL Code
void delete_row();
```

Deletes the current row from the table.

Enhancement This is an Orbix enhancement.

ResultSet::FetchDirection Type

```
// PSDL Code
typedef unsigned short FetchDirection;
const FetchDirection FETCH_FORWARD = 1;
const FetchDirection FETCH_REVERSE = 2;
const FetchDirection FETCH_UNKNOWN = 3;
```

Defines the direction of table row processing.

FETCH_FORWARD	The rows in a result set will be processed in a forward direction; first-to-last.
FETCH_REVERSE	The rows in a result set will be processed in a reverse direction; last-to-first.
FETCH_UNKNOWN	The order in which rows in a result set will be processed is unknown.

Enhancement This is an Orbix enhancement.

ResultSet::find_state_member()

```
// PSDL Code
unsigned short find_state_member(
   in string state_member_name
);
```

Returns the index for the given result set's state member name.

Enhancement This is an Orbix enhancement.

ResultSet::first()

```
// PSDL Code
boolean first();
```

Moves the cursor to the first row in the result set. Returns true if the cursor is on a valid row; false if there are no rows in the result set

Enhancement This is an Orbix enhancement.

ResultSet::get()

```
// PSDL Code
any get(
    in unsigned short index
);
```

Returns the value for the given parameter index.

Enhancement This is an Orbix enhancement.

See Also IT_PSS::ResultSet::set()

ResultSet::get_by_name()

```
// PSDL Code
any get_by_name(
    in string state_member_name
);
```

Returns the value for a state member given the member name.

Enhancement This is an Orbix enhancement.

See Also IT_PSS::ResultSet::set_by_name()

ResultSet::get_concurrency()

```
// PSDL Code
Concurrency get_concurrency();
```

Returns the concurrency value.

Enhancement This is an Orbix enhancement.

ResultSet::get_fetch_direction()

```
// PSDL Code
FetchDirection get_fetch_direction();
```

Returns the direction of table row processing.

Enhancement This is an Orbix enhancement.

See Also IT_PSS::ResultSet::set_fetch_direction()

ResultSet::get_fetch_size()

```
// PSDL Code
unsigned short get_fetch_size();
```

Returns the number of rows that are fetched from the database when more rows are needed for this result set.

Enhancement This is an Orbix enhancement.

See Also IT_PSS::ResultSet::set_fetch_size()

ResultSet::get_row()

```
// PSDL Code
unsigned short get_row();
```

Returns the current row number. The first row is number 1, the second number is 2, and so on.

Enhancement This is an Orbix enhancement.

ResultSet::get_statement()

```
// PSDL Code
Statement get_statement();
```

Returns the `Statement` that produced this `ResultSet` object.

Enhancement This is an Orbix enhancement.

ResultSet::get_type()

```
// PSDL Code
Type get_type();
```

Returns the type of this result set. The type is determined by the `Statement` that created the result set.

Enhancement This is an Orbix enhancement.

ResultSet::insert_row()

```
// PSDL Code
void insert_row();
```

Inserts a row.

Enhancement This is an Orbix enhancement.

ResultSet::is_after_last()

```
// PSDL Code
boolean is_after_last();
```

Returns true if the cursor is after the last row in the result set, false if it is not.

Enhancement This is an Orbix enhancement.

ResultSet::is_before_first()

```
// PSDL Code
boolean is_before_first();
```

Returns true if the cursor is before the first row in the result set, false if it is not.

Enhancement This is an Orbix enhancement.

ResultSet::is_first()

```
// PSDL Code
boolean is_first();
```

Returns true if the cursor is on the first row of the result set, false if it is not.

Enhancement This is an Orbix enhancement.

ResultSet::is_last()

```
// PSDL Code
boolean is_last();
```

Returns true if the cursor is on the last row of the result set, false if it is not.

Enhancement This is an Orbix enhancement.

ResultSet::last()

```
// PSDL Code
boolean last();
```

Moves the cursor to the last row in the result set and returns true if the cursor is on a valid row; false if there are no rows in the result set.

Enhancement This is an Orbix enhancement.

ResultSet::move_to_current_row()

```
// PSDL Code
void move_to_current_row();
```

Moves the cursor to the remembered cursor position, usually the current row. This operation has no effect if the cursor is not on the insert row.

Enhancement This is an Orbix enhancement.

ResultSet::move_to_insert_row()

```
// PSDL Code
void move_to_insert_row();
```

Moves the cursor to the insert row.

Enhancement This is an Orbix enhancement.

ResultSet::next()

```
// PSDL Code
boolean next();
```

Moves the cursor down one row from its current position. A ResultSet cursor is initially positioned before the first row; the first call to next makes the first row the current row; the second call makes the second row the current row, and so on.

Enhancement This is an Orbix enhancement.

ResultSet::previous()

```
// PSDL Code
boolean previous();
```

Moves the cursor to the previous row in the result set.

Enhancement This is an Orbix enhancement.

ResultSet::refresh_row()

```
// PSDL Code
void refresh_row();
```

Refreshes the current row with its most recent value in the database. This cannot be called when the cursor is on the insert row.

Enhancement This is an Orbix enhancement.

ResultSet::relative()

```
// PSDL Code
boolean relative(
    in short rows
);
```

Moves the cursor a relative number of rows, either positive or negative. Attempting to move beyond the first/last row in the result set positions the cursor before/after the first/last row. Calling `relative(0) is` valid, but does not change the cursor position.

Enhancement This is an Orbix enhancement.

ResultSet::row_deleted()

```
// PSDL Code
boolean row_deleted();
```

Indicates whether a row has been deleted. A deleted row may leave a visible "hole" in a result set. This operation can be used to detect holes in a result set. The value returned depends on whether or not the result set can detect deletions.

Enhancement This is an Orbix enhancement.

ResultSet::row_inserted()

```
// PSDL Code
boolean row_inserted();
```

Indicates whether the current row has had an insertion. The value returned depends on whether or not the result set can detect visible inserts. The operation returns true if a row has had an insertion and insertions are detected.

Enhancement This is an Orbix enhancement.

ResultSet::row_updated()

```
// PSDL Code
boolean row_updated();
```

Indicates whether the current row has been updated. The value returned depends on whether or not the result set can detect updates. If the set can detect updates, the operation returns true if the row has been visibly updated by the owner or another.

Enhancement This is an Orbix enhancement.

ResultSet::set()

```
// PSDL Code
void set(
    in unsigned short index,
    in any value
);
```

Sets the value and parameter index.

Enhancement This is an Orbix enhancement.

See Also IT_PSS::ResultSet::get()

ResultSet::set_by_name()

```
// PSDL Code
void set_by_name(
    in string state_member_name,
    in any value
);
```

Sets the value for an object's member given the name of the member.

Enhancement This is an Orbix enhancement.

See Also IT_PSS::ResultSet::get_by_name()

ResultSet::set_fetch_direction()

```
// PSDL Code
void set_fetch_direction(
    in FetchDirection direction
);
```

Sets a hint as to the direction in which the rows in this result set will be processed. The initial value is determined by the statement that produced the result set. The fetch direction may be changed at any time.

Enhancement This is an Orbix enhancement.

See Also IT_PSS::ResultSet::get_fetch_direction()

ResultSet::set_fetch_size()

```
// PSDL Code
void set_fetch_size(
    in unsigned short fetch_size
);
```

The fetch size is a hint as to the number of rows that should be fetched from the database when more rows are needed for this result set. The default value is set by the Statement that created the result set. The fetch size may be changed at any time.

Parameters

`fetch_size`	If the fetch size is zero, a best guess is used.

Enhancement This is an Orbix enhancement.

See Also IT_PSS::ResultSet::get_fetch_size()

ResultSet::Type

```
// PSDL Code
typedef unsigned short Type;
const Type TYPE_FORWARD_ONLY =        1;
const Type TYPE_SCROLL_INSENSITIVE =  2;
const Type TYPE_SCROLL_SENSITIVE =    3;
```

The type of this result set. The type is determined by the Statement that created the result set.

Enhancement This is an Orbix enhancement.

ResultSet::update_row()

```
// PSDL Code
void update_row();
```

Updates the underlying database with the new contents of the current row. Cannot be called when the cursor is on the insert row.

Enhancement This is an Orbix enhancement.

IT_PSS::Session Interface

When you create a session with an IONA PSS implementation, you get an `IT_PSS::Session`.

```
// PSDL Code in module IT_PSS
local interface Session : CatalogBase, CosPersistentState::Session
    {};
```

Enhancement This interface is an Orbix enhancement.

See Also `IT_PSS::CatalogBase`
`CosPersistentState::Session`

IT_PSS::SessionManager Interface

PSS fully support transactions, and works with any compliant transaction service implementation. Unless you are developing a trivial demonstration program, you should use transactions when developing applications with PSS.

You can use a `SessionManager` object to manage transactional sessions. A common pattern when developing a transactional server using PSS is to use a shared read-only read-committed transactional session for simple read-only non-transactional requests. Of course, you can also create and manage your transactional sessions directly with the standard lower level PSS APIs from the `CosPersistentState` module.

```
//PSDL in module IT_PSS
local interface SessionManager {
    TransactionalSession get_shared_read_only_session_nc();
    void block_readers_until_idle();
};
```

Enhancement This interface is an Orbix enhancement.

See Also `IT_PSS::Connector::it_create_session_manager()`

SessionManager::get_shared_read_only_session_nc()

```
//PSDL code
TransactionalSession get_shared_read_only_session_nc();
```

Returns a shared, read-only transactional session. In this context, shared means the transactional session is usable by multiple threads.

Enhancement This is an Orbix enhancement.

SessionManager::block_readers_until_idle()

```
//PSDL code
void block_readers_until_idle();
```

Blocks new threads from using the shared, read-only transactional session until no thread is using the session.

Enhancement This is an Orbix enhancement.

IT_PSS::Statement Interface

The Statement interface provides operations for a JDBC-like statement, an object used for executing a static SQL statement and obtaining the results produced by it.

```
// PSDL Code in module IT_PSS
local interface Statement {
    void execute(
        in string pssql
    );

    ResultSet execute_query(
        in string pssql
    );

    unsigned long execute_update(
        in string pssql
    );

    ResultSet get_result_set();

    void close();

    // Default fetch direction and size
    //
    void set_fetch_direction(
        in ResultSet::FetchDirection direction
    );

    ResultSet::FetchDirection get_fetch_direction();

    void set_fetch_size(
        in unsigned short fetch_size
    );

    unsigned short get_fetch_size();

    // Type and Concurrency
```

```
    //
    ResultSet::Type get_result_set_type();
    ResultSet::Concurrency get_result_set_concurrency();
    CatalogBase get_catalog();
};
```

Enhancement This interface is an Orbix enhancement.

See Also IT_PSS::CatalogBase
IT_PSS::PreparedStatement

Statement::close()

```
// PSDL Code
void close();
```

Releases this Statement object's database and resources immediately instead of waiting for this to happen when it is automatically closed.

Enhancement This is an Orbix enhancement.

Statement::execute()

```
// PSDL Code
void execute(
    in string pssql
);
```

Executes an SQL statement that may obtain multiple results.

Enhancement This is an Orbix enhancement.

Statement::execute_query()

```
// PSDL Code
ResultSet execute_query(
    in string pssql
);
```

Executes an SQL statement that returns a single ResultSet.

Enhancement This is an Orbix enhancement.

Statement::execute_update()

```
// PSDL Code
unsigned long execute_update(
    in string pssql
);
```

Executes an SQL `INSERT`, `UPDATE` or `DELETE` statement.

Enhancement This is an Orbix enhancement.

Statement::get_catalog()

```
// PSDL Code
CatalogBase get_catalog();
```

Returns the catalog for this Statement.

Enhancement This is an Orbix enhancement.

Statement::get_fetch_direction()

```
// PSDL Code
ResultSet::FetchDirection get_fetch_direction();
```

Returns the direction for fetching rows from database tables that is the default for result sets generated from this `Statement` object.

Enhancement This is an Orbix enhancement.

Statement::get_fetch_size()

```
// PSDL Code
unsigned short get_fetch_size();
```

Returns the number of result set rows that is the default fetch size for result sets generated from this `Statement` object.

Enhancement This is an Orbix enhancement.

Statement::get_result_set()

```
// PSDL Code
ResultSet get_result_set();
```

Returns the current result as a ResultSet object.

Enhancement This is an Orbix enhancement.

Statement::get_result_set_concurrency()

```
// PSDL Code
ResultSet::Concurrency get_result_set_concurrency();
```

Returns the result set concurrency.

Enhancement This is an Orbix enhancement.

Statement::get_result_set_type()

```
// PSDL Code
ResultSet::Type get_result_set_type();
```

Returns the type of the ResultSet.

Enhancement This is an Orbix enhancement.

Statement::set_fetch_direction()

```
// PSDL Code
void set_fetch_direction(
    in ResultSet::FetchDirection direction
);
```

Sets a hint as to the direction in which the rows in a result set should be processed.

Enhancement This is an Orbix enhancement.

Statement::set_fetch_size()

```
// PSDL Code
void set_fetch_size(
    in unsigned short fetch_size
);
```

Gives a hint as to the number of rows that should be fetched from the database when more rows are needed.

Enhancement This is an Orbix enhancement.

IT_PSS_StorageHomeFactory Template

Use this template class to help implement your StorageHomeFactory.

```
template<class T>
class IT_PSS_StorageHomeFactory :
public CosPersistentState::StorageHomeFactory {
  public:

    IT_PSS_StorageHomeFactory();

    virtual void _add_ref();

    virtual void _remove_ref();

    virtual CosPersistentState::StorageHomeBase_ptr create()
        throw(CORBA::SystemException);

  private:
    ...
};
```

Enhancement This is an Orbix enhancement.

IT_PSS_StorageHomeFactory::_add_ref()

```
virtual void _add_ref();
```

Increases the reference count by one.

Enhancement This is an Orbix enhancement.

IT_PSS_StorageHomeFactory::create()

```
virtual CosPersistentState::StorageHomeBase_ptr create()
        throw(CORBA::SystemException);
```

Creates and returns a new StorageHomeBase object.

IT_PSS_StorageHomeFactory::IT_PSS_StorageHomeFactory()

```
IT_PSS_StorageHomeFactory();
```

The constructor.

Enhancement This is an Orbix enhancement.

IT_PSS_StorageHomeFactory::_remove_ref()

```
virtual void _remove_ref();
```

Decreases the reference count by one.

Enhancement This is an Orbix enhancement.

IT_PSS::StorageObject Interface

PSS presents persistent information as storage objects. Each storage object has a type that defines its members and operations. When you create a storage object with an IONA PSS implementation, you get an `IT_PSS::StorageObject`.

```
// PSDL Code in module IT_PSS
abstract storagetype StorageObject {
    void it_lock();

    };
```

Enhancement This interface is an Orbix enhancement.

See Also `CosPersistentState::StorageObject`

StorageObject::it_lock()

```
// PSDL Code
void it_lock();
```

This operation acquires an exclusive lock on behalf of a basic session or transactional session.

Enhancement This is an Orbix enhancement.

IT_PSS_StorageObjectFactory Template

Use this template class to help implement your StorageObjectFactory.

```
// c++
template<class T>
class IT_PSS_StorageObjectFactory :
public CosPersistentState::StorageObjectFactory {
  public:

    IT_PSS_StorageObjectFactory();

    virtual void  _add_ref();

    virtual void  _remove_ref();

    virtual CosPersistentState::StorageObject* create()
        throw(CORBA::SystemException);

  private:
    ...
};
```

Enhancement This is an Orbix enhancement.

IT_PSS_StorageObjectFactory::_add_ref()

```
virtual void _add_ref();
```

Increases the reference count by one.

IT_PSS_StorageObjectFactory::create()

```
virtual CosPersistentState::StorageObject* create()
        throw(CORBA::SystemException);
```

Creates and returns a new `StorageObject` object.

Enhancement This is an Orbix enhancement.

IT_PSS_StorageObjectFactory:: IT_PSS_StorageObjectFactory()

```
IT_PSS_StorageObjectFactory();
```

The constructor.

Enhancement This is an Orbix enhancement.

IT_PSS_StorageObjectFactory::_remove_ref()

```
virtual void _remove_ref();
```

Decreases the reference count by one.

Enhancement This is an Orbix enhancement.

IT_PSS::TransactionalSession Interface

When you create a transactional session with an IONA PSS implementation, you get an `IT_PSS::TransactionalSession` object.

```
// PSDL Code in module IT_PSS
local interface TransactionalSession :
Session, CosPersistentState::TransactionalSession
{
Master get_master();
boolean is_replica();
Replica get_replica();
};
```

This interface provides propriatary enhancements to the OGM `TransactionalSession` interface. It consists of functions to manage replicated persistent objects.

IT_PSS::TransactionalSession::get_master

```
Master get_master();
```

Returns an object reference to a replica's master instance. If the session is associated with a master, then it will return an object reference to itself. If the master instance was not set or is unreachable, the function will return `NIL`.

IT_PSS::TransactionalSession:is_replica

```
boolean is_replica();
```

Returns `TRUE` if the object is a replica of a datastore and `FALSE` if it is not.

IT_PSS::TransactionalSession:get_replica

```
Replica get_replica();
```

If the session is associated with a replica of a datastore, it will return an object reference to its `Replica` object. If the session is associated with a master instance, it will return `NIL`.

IT_PSS::TxSessionAssociation Class

You can use stack-allocated `TxSessionAssociation` objects to create associations between OTS transactions and PSS transactional sessions managed by a `SessionManager`.

```
class  TxSessionAssociation {
  public:
    TxSessionAssociation(
        IT_PSS::SessionManager_ptr              session_mgr,
        CosPersistentState::AccessMode          access_mode
    ) throw(CORBA::SystemException);

    TxSessionAssociation(
        IT_PSS::SessionManager_ptr              session_mgr,
        CosPersistentState::AccessMode          access_mode,
        CosTransactions::Coordinator_ptr        tx_coordinator
    ) throw(CORBA::SystemException);

    ~TxSessionAssociation()
        throw(CORBA::SystemException);

    IT_PSS::TransactionalSession_ptr  get_session_nc()
        const throw();

    CosTransactions::Coordinator_ptr  get_tx_coordinator_nc()
        const throw();

    void suspend()
        throw(CORBA::SystemException);

    void end(
        CORBA::Boolean success = IT_TRUE
    ) throw(CORBA::SystemException);

  private:
    ...
};
```

Enhancement This class is an Orbix enhancement.

TxSessionAssociation::end()

```
void end(
    CORBA::Boolean success = IT_TRUE
) throw(CORBA::SystemException);
```

Ends the association only if this object started or resumed the association.This method has no effect if the association already ended.

Parameters

success	Determines if the method was successful.

Enhancement This is an Orbix enhancement.

See Also `IT_PSS::TxSessionAssociation::suspend()`

TxSessionAssociation::get_session_nc()

```
IT_PSS::TransactionalSession_ptr  get_session_nc()
    const IT_THROW_DECL(());
```

Returns a non-copied reference to the session. This mean that the caller must not release the returned reference.

Enhancement This is an Orbix enhancement.

TxSessionAssociation::get_tx_coordinator_nc()

```
CosTransactions::Coordinator_ptr  get_tx_coordinator_nc()
    const IT_THROW_DECL(());
```

Returns a non-copied reference to the association's transaction coordinator. This mean that the caller must not release the returned reference. After a transaction-session association object is constructed, `get_tx_coordinator_nc()` returns nil when and only when the object represents an association between the session manager's read-only transaction and the session manager's shared read-only session.

Enhancement This is an Orbix enhancement.

TxSessionAssociation::TxSessionAssociation() Constructors

```
TxSessionAssociation(
    IT_PSS::SessionManager_ptr          session_mgr,
    CosPersistentState::AccessMode      access_mode
) throw(CORBA::SystemException);
```

A constructor without a supplied transaction.

```
TxSessionAssociation(
    IT_PSS::SessionManager_ptr          session_mgr,
    CosPersistentState::AccessMode      access_mode,
    CosTransactions::Coordinator_ptr    tx_coordinator
) throw(CORBA::SystemException);
```

A constructor with a transaction.

Parameters

`session_mgr`	The session manager.
`access_mode`	Access mode for the association. If `tx_coordinator` is not provided, the constructor's behavior is as follows: • If access mode is `READ_ONLY`, then start or use an association between the session manager's read-only transaction and the session manager's shared read-only session. • If access mode is `READ_WRITE`, then raise the `CORBA::TRANSACTION_REQUIRED`.
`tx_coordinator`	A transaction coordinator.

If a transaction is provided, the behavior depends on the number of associations between this transaction and sessions created by the session's manager connector:

Table 31: *Associations Between a Transaction and Sessions*

Number of Associations	Behavior
Greater than 1	Raises the `CORBA::IMPL_LIMIT` exception.
1	Does nothing if it is `ACTIVE`, otherwise it starts it.
none	Creates a new association between this transaction and a read-write transactional session managed by the session manager.

Enhancement This is an Orbix enhancement.

TxSessionAssociation::~TxSessionAssociation() Destructor

```
~TxSessionAssociation()
    throw(CORBA::SystemException);
```

If there is still an association when the destructor is called, and this object started the association, the association is suspended. If the suspend fails, the association ends with the success flag set to FALSE.

Enhancement This is an Orbix enhancement.

TxSessionAssociation::suspend()

```
void suspend()
    throw(CORBA::SystemException);
```

Suspends the association only when this object started or resumed the association. This method has no effect if the association has already suspended or ended.

Enhancement This is an Orbix enhancement.

See Also `IT_PSS::TxSessionAssociation::end()`

The IT_PSS_DB Module Overview

This module contains the single interface Env.

IT_PSS_DB::Env Interface

```
// IDL
module IT_PSS_DB {
    interface Env {
        readonly attribute string name;

        void pre_backup();
        void post_backup();
        void checkpoint();
    };
};
```

Enhancement This interface is an Orbix enhancement.

Env::checkpoint()

```
// IDL
void checkpoint();
```

Enhancement This is an Orbix enhancement.

Env::name Attribute

```
// IDL
readonly attribute string name;
```

Enhancement This is an Orbix enhancement.

Env::post_backup()

```
// IDL
void post_backup();
```

Enhancement This is an Orbix enhancement.

Env::pre_backup()

```
// IDL
void pre_backup();
```

Enhancement This is an Orbix enhancement.

Security

IT_Certificate Overview

The IT_Certificate module provides data types and interfaces that are used to manage and describe X.509 certificates. The following interfaces are provided in this module:

- AVA
- AVAList
- Extension
- ExtensionList
- Certificate
- X509Cert
- X509CertificateFactory

IT_Certificate::ASN_OID Structure

```
// IDL
struct ASN_OID
{
    OIDTag  tag;
    ASN1oid asn1_oid;
    string  tag_name;
};
```

Holds an ASN.1 object ID (OID).

The ASN.1 OID can be specified by setting either the tag or asn1_oid structure members.

An ASN_OID structure returned by Orbix SSL/TLS normally sets both the tag and asn1_oid members in the structure. The returned tag value will be IT_OIDT_UNKNOWN, however, if Orbix SSL/TLS does not recognize the OID from its internal table of known OIDs.

The structure has the following members:

`tag`	An Orbix-specific tag to identify an AVA. For example, the `IT_Certificate::IT_OIDT_COMMON_NAME` tag identifies the Common Name AVA. If you set `tag` equal to the special value `IT_Certificate::IT_OIDT_UNKNOWN`, it will be ignored and the `asn1_oid` member will be used instead.
`asn1_oid`	An ASN.1 OID to identify an AVA, specified in the standard way as a sequence of integers. For example, the sequence `2.5.4.3` identifies the Common Name AVA.
`tag_name`	Reserved for future use by Orbix SSL/TLS.

IT_Certificate::ASN1oid Sequence

```
typedef sequence<UShort> ASN1oid;
```

Holds an ASN.1 OID in the standard format, which is a sequence of integers. For example, the sequence `2.5.4.3` identifies the Common Name AVA.

IT_Certificate::Bytes Sequence

```
typedef sequence<octet> Bytes;
```

Holds raw binary data.

IT_Certificate::CertError Exception

```
// IDL
exception CertError
{
    Error e;
};
```

A certificate-related error.

IT_Certificate::DERData Sequence

```
typedef sequence<octet> DERData;
```

Holds data in distinguished encoding rules (DER) format.

IT_Certificate::Error Structure

```
struct Error
{
    Error_code err_code;
    string error_message;
};
```

Holds certificate-related error information.

IT_Certificate::Error_code Type

```
typedef short Error_code;
```

Holds the certificate-related error codes.

Values This type can have one of the following integer constant values:

```
IT_TLS_FAILURE
IT_TLS_UNSUPPORTED_FORMAT
IT_TLS_BAD_CERTIFICATE_DATA
IT_TLS_ERROR_READING_DATA
```

IT_Certificate::Format Structure

```
//IDL
typedef short Format;
```

Specifies a specific format for X.509 certificate data.

Values This type can have one of the following integer constant values:

`IT_FMT_DER`	This format corresponds to the DER encoding of the AVA. This option is usually only used by applications that require special processing of the DER data.

IT_FMT_PEM	Privacy enhanced mail (PEM) format certificate format. In this format, the certificate consists of standard ASCII characters that can be safely transmitted as text.
IT_FMT_STRING	This format corresponds to a null-terminated sequence of characters containing the actual data of the AVA. The data is not modified in any way, and can include non-printable characters if present in the actual AVA data. This is a string for normal printable string fields.
IT_FMT_HEX_STRING	This format corresponds to a formatted hexadecimal dump of the DER data of the AVA.

IT_Certificate::OIDTag Type

```
typedef UShort OIDTag;
```

An Orbix-specific tag type that represents an ASN.1 OID. Tags are defined for most of the commonly used AVAs in an X.509 certificate. These tags are provided as a convenient alternative to the standard OID format, `IT_Certificate::ASN1oid`.

Values This type can have one of the following integer constant values:

```
IT_OIDT_UNKNOWN
IT_OIDT_RSADSI
IT_OIDT_PKCS
IT_OIDT_MD2
IT_OIDT_MD5
IT_OIDT_RC4
IT_OIDT_RSA_ENCRYPTION
IT_OIDT_MD2_WITH_RSA_ENCRYPTION
IT_OIDT_MD5_WITH_RSA_ENCRYPTION
IT_OIDT_PBE_WITH_MD2_AND_DES_CBC
IT_OIDT_PBE_WITH_MD5_AND_DES_CBC
IT_OIDT_X500
IT_OIDT_X509
IT_OIDT_COMMON_NAME
IT_OIDT_COUNTRY_NAME
IT_OIDT_LOCALITY_NAME
IT_OIDT_STATE_OR_PROVINCE_NAME
IT_OIDT_ORGANIZATION_NAME
```

IT_OIDT_ORGANIZATIONAL_UNIT_NAME
IT_OIDT_RSA
IT_OIDT_PKCS7
IT_OIDT_PKCS7_DATA
IT_OIDT_PKCS7_SIGNED
IT_OIDT_PKCS7_ENVELOPED
IT_OIDT_PKCS7_SIGNED_AND_ENVELOPED
IT_OIDT_PKCS7_DIGEST
IT_OIDT_PKCS7_ENCRYPTED
IT_OIDT_PKCS3
IT_OIDT_DHKEY_AGREEMENT
IT_OIDT_DES_ECB
IT_OIDT_DES_CFB64
IT_OIDT_DES_CBC
IT_OIDT_DES_EDE
IT_OIDT_DES_EDE3
IT_OIDT_IDEA_CBC
IT_OIDT_IDEA_CFB64
IT_OIDT_IDEA_ECB
IT_OIDT_RC2_CBC
IT_OIDT_RC2_ECB
IT_OIDT_RC2_CFB64
IT_OIDT_RC2_OFB64
IT_OIDT_SHA
IT_OIDT_SHA_WITH_RSA_ENCRYPTION
IT_OIDT_DES_EDE_CBC
IT_OIDT_DES_EDE3_CBC
IT_OIDT_DES_OFB64
IT_OIDT_IDEA_OFB64
IT_OIDT_PKCS9
IT_OIDT_PKCS9_EMAIL_ADDRESS
IT_OIDT_PKCS9_UNSTRUCTURED_NAME
IT_OIDT_PKCS9_CONTENTTYPE
IT_OIDT_PKCS9_MESSAGE_DIGEST
IT_OIDT_PKCS9_SIGNING_TIME
IT_OIDT_PKCS9_COUNTER_SIGNATURE
IT_OIDT_PKCS9_CHALLENGE_PASSWORD
IT_OIDT_PKCS9_UNSTRUCTURED_ADDRESS
IT_OIDT_PKCS9_EXTCERT_ATTRIBUTES
IT_OIDT_NETSCAPE
IT_OIDT_NETSCAPE_CERT_EXTENSION
IT_OIDT_NETSCAPE_DATA_TYPE
IT_OIDT_DES_EDE_CFB64

```
IT_OIDT_DES_EDE3_CFB64
IT_OIDT_DES_EDE_OFB64
IT_OIDT_DES_EDE3_OFB64
IT_OIDT_SHA1
IT_OIDT_SHA1_WITH_RSA_ENCRYPTION
IT_OIDT_DSA_WITH_SHA
IT_OIDT_DSA_2
IT_OIDT_PBE_WITH_SHA1_AND_RC2_CBC
IT_OIDT_ID_PBKDF2
IT_OIDT_DSA_WITH_SHA1_2
IT_OIDT_NETSCAPE_CERT_TYPE
IT_OIDT_NETSCAPE_BASE_URL
IT_OIDT_NETSCAPE_REVOCATION_URL
IT_OIDT_NETSCAPE_CA_REVOCATION_URL
IT_OIDT_NETSCAPE_RENEWAL_URL
IT_OIDT_NETSCAPE_CA_POLICY_URL
IT_OIDT_NETSCAPE_SSL_SERVER_NAME
IT_OIDT_NETSCAPE_COMMENT
IT_OIDT_NETSCAPE_CERT_SEQUENCE
IT_OIDT_DESX_CBC
IT_OIDT_LD_CE
IT_OIDT_SUBJECT_KEY_IDENTIFIER
IT_OIDT_KEY_USAGE
IT_OIDT_PRIVATE_KEY_USAGE_PERIOD
IT_OIDT_SUBJECT_ALT_NAME
IT_OIDT_ISSUER_ALT_NAME
IT_OIDT_BASIC_CONSTRAINTS
IT_OIDT_CRL_NUMBER
IT_OIDT_CERTIFICATE_POLICIES
IT_OIDT_AUTHORITY_KEY_IDENTIFIER
IT_OIDT_BF_CBC
IT_OIDT_BF_ECB
IT_OIDT_BF_CFB64
IT_OIDT_BF_OFB64
IT_OIDT_MDC2
IT_OIDT_MDC2_WITH_RSA
IT_OIDT_RC4_40
IT_OIDT_RC2_40_CBC
IT_OIDT_GIVEN_NAME
IT_OIDT_SURNAME
IT_OIDT_INITIALS
IT_OIDT_UNIQUEIDENTIFIER
IT_OIDT_CRL_DISTRIBUTION_POINTS
```

IT_OIDT_MD5_WITH_RSA
IT_OIDT_SERIALNUMBER
IT_OIDT_TITLE
IT_OIDT_DESCRIPTION
IT_OIDT_CAST5_CBC
IT_OIDT_CAST5_ECB
IT_OIDT_CAST5_CFB64
IT_OIDT_CAST5_OFB64
IT_OIDT_PBE_WITH_MD5_AND_CAST5_CBC
IT_OIDT_DSA_WITH_SHA1
IT_OIDT_MD5_SHA1
IT_OIDT_SHA1_WITH_RSA
IT_OIDT_DSA
IT_OIDT_RIPEMD160
IT_OIDT_UNDEF
IT_OIDT_RIPEMD160_WITH_RSA
IT_OIDT_RC5_CBC
IT_OIDT_RC5_ECB
IT_OIDT_RC5_CFB64
IT_OIDT_RC5_OFB64
IT_OIDT_RLE_COMPRESSION
IT_OIDT_ZLIB_COMPRESSION
IT_OIDT_EXT_KEY_USAGE
IT_OIDT_ID_PKIX
IT_OIDT_ID_KP
IT_OIDT_SERVER_AUTH
IT_OIDT_CLIENT_AUTH
IT_OIDT_CODE_SIGN
IT_OIDT_EMAIL_PROTECT
IT_OIDT_TIME_STAMP
IT_OIDT_MS_CODE_IND
IT_OIDT_MS_CODE_COM
IT_OIDT_MS_CTL_SIGN
IT_OIDT_MS_SGC
IT_OIDT_MS_EFS
IT_OIDT_NS_SGC
IT_OIDT_DELTA_CRL
IT_OIDT_CRL_REASON
IT_OIDT_INVALIDITY_DATE
IT_OIDT_SXNET
IT_OIDT_PBE_WITH_SHA1_AND_128BITRC4
IT_OIDT_PBE_WITH_SHA1_AND_40BITRC4
IT_OIDT_PBE_WITH_SHA1_AND_3_KEY_TRIPLEDES_CBC

```
IT_OIDT_PBE_WITH_SHA1_AND_2_KEY_TRIPLEDES_CBC
IT_OIDT_PBE_WITH_SHA1_AND_128BITRC2_CBC
IT_OIDT_PBE_WITH_SHA1_AND_40BITRC2_CBC
IT_OIDT_KEY_BAG
IT_OIDT_PKCS8SHROUDEDKEY_BAG
IT_OIDT_CERT_BAG
IT_OIDT_CRL_BAG
IT_OIDT_SECRET_BAG
IT_OIDT_SAFECONTENTS_BAG
IT_OIDT_FRIENDLY_NAME
IT_OIDT_LOCALKEYID
IT_OIDT_X509CERTIFICATE
IT_OIDT_SDSICERTIFICATE
IT_OIDT_X509CRL
IT_OIDT_PBES2
IT_OIDT_PBMAC1
IT_OIDT_HMAC_WITH_SHA1
IT_OIDT_ID_QT_CPS
IT_OIDT_ID_QT_UNOTICE
IT_OIDT_RC2_64_CBC
IT_OIDT_SMIMECAPABILITIES
IT_OIDT_PBE_WITH_MD2_AND_RC2_CBC
IT_OIDT_PBE_WITH_MD5_AND_RC2_CBC
IT_OIDT_PBE_WITH_SHA1_AND_DES_CBC
```

IT_Certificate::ReplyStatus Type

```
typedef short ReplyStatus;
```

Gives the reply status of certain operations in the `IT_Certificate` module.

Values This type can have the following integer constant values:

```
SUCCESSFUL
AVA_NOT_PRESENT
EXTENSION_NOT_PRESENT
NO_EXTENSIONS_PRESENT
```

See Also

`IT_Certificate::AVAList`
`IT_Certificate::ExtensionList`
`IT_Certificate::X509Cert`

IT_Certificate::ULong Type

```
typedef unsigned long ULong;
```

An unsigned long integer.

IT_Certificate::UShort Type

```
typedef unsigned short UShort;
```

An unsigned short integer.

IT_Certificate::UTCTime Type

```
typedef sequence<string> UTCTime;
```

A type used to hold time (and date) information in a certificate.

IT_Certificate::X509CertChain Sequence

```
typedef sequence<X509Cert> X509CertChain;
```

A list of `X509Cert` object references.

IT_Certificate::X509CertList Sequence

```
typedef sequence<X509Cert> X509CertList;
```

A list of `X509Cert` object references.

IT_Certificate::AVA Interface

IDL

```
// IDL in module IT_Certificate
interface AVA
{
    readonly attribute UShort set;
    readonly attribute ASN_OID oid;

    // raises minor code IT_TLS_UNSUPPORTED_FORMAT
    Bytes convert(in Format f) raises(CertError);
};
```

Individual AVA objects represent an element of the distinguished name such as the common name field (CN) or organization unit (OU). You can retrieve a desired AVA object can using the AVAList class.

AVA objects can be converted to a number of different forms such as string format or DER format.

AVA::convert()

```
// IDL
Bytes convert(in Format f) raises(CertError);
```

Description

This operation returns the contents of the AVA object in the requested data format.

Parameters This operation takes the following parameter

`f` The format of the required conversion. The following `Format` values are supported:

`IT_FMT_DER`. This format corresponds to the DER encoding of the AVA. This option is usually only used by applications that require special processing of the DER data.

`IT_FMT_STRING`. This format corresponds to a null-terminated sequence of characters containing the actual data of the AVA. The data is not modified in any way, and can include non-printable characters if present in the actual AVA data. This is a string for normal printable string fields.

`IT_FMT_HEX_STRING`. This format corresponds to a formatted hexadecimal dump of the DER data of the AVA.

Exceptions

`CertError` with error code `IT_TLS_UNSUPPORTED_FORMAT`	An unknown format is specified.

AVA::oid

```
// IDL
readonly attribute ASN_OID oid;
```

Description Return the ASN.1 OID tag for this AVA object, in the form of an `ASN_OID` structure.

AVA::set

```
// IDL
readonly attribute UShort set;
```

Description A number that identifies the set to which the AVA belongs. Because a set normally contains just a single AVA, the number returned by the `set` attribute is usually distinct for each AVA.

Theoretically, more than one AVA could belong to the same set, in which case two or more AVAs could share the same `set` number. In practice, this rarely ever happens.

IT_Certificate::AVAList Interface

IDL

```
// IDL in module IT_Certificate
interface AVAList
{
    typedef sequence<AVA> ListOfAVAs;
    readonly    attribute ListOfAVAs ava_list;

    UShort get_num_avas();

    // Returns SUCCESSFUL or AVA_NOT_PRESENT
    IT_Certificate::ReplyStatus
    get_ava_by_oid_tag(
        in OIDTag t,
        out AVA a
    ) raises(CertError);

    // Returns SUCCESSFUL or AVA_NOT_PRESENT
    IT_Certificate::ReplyStatus
    get_ava_by_oid(
        in ASN_OID seq,
        in UShort n,
        out AVA a
    ) raises(CertError);

    // raises minor code IT_TLS_UNSUPPORTED_FORMAT
    Bytes convert(
        in Format f
    ) raises(CertError);
};
```

Description

An `AVAList` is an abstraction of a distinguished name from a certificate. An `AVAList` consists of a number of `AVA` objects.

Individual `AVA` objects represent an element of the distinguished name such as the common name field (CN) or organization unit (OU). You can retrieve a desired `AVA` object using the `AVAList`.

`AVA` objects can be converted to a number of different forms such as string format or DER format.

AVAList::ava_list

IDL `readonly attribute ListOfAVAs ava_list;`

Description Returns the AVA list as a sequence of AVA object references.

AVAList::convert()

IDL `Bytes convert(in Format f) raises (CertError);`

Description This operation converts the AVAList to a specified format.

Parameters This operation takes the following parameter:

`f` The format of the required conversion. The following `Format` values are supported:

`IT_FMT_DER`. This format corresponds to the DER encoding of the AVA. This option is usually only used by applications that require special processing of the DER data.

`IT_FMT_STRING`. This format corresponds to a null-terminated sequence of characters containing the actual data of the AVA. The data is not modified in any way, and can include non-printable characters if present in the actual AVA data. This is a string for normal printable string fields.

`IT_FMT_HEX_STRING`. This format corresponds to a formatted hexadecimal dump of the DER data of the AVA.

Exceptions

`CertError`, error code `IT_TLS_UNSUPPORTED_FORMAT`	An unknown format is specified.

AVAList::get_ava_by_oid_tag()

IDL

```
// Returns SUCCESSFUL or AVA_NOT_PRESENT
IT_Certificate::ReplyStatus
get_ava_by_oid_tag(
    in OIDTag t,
    out AVA a
) raises(CertError);
```

Description This operation retrieves an AVA object from an AVAList according to its OID tag.

Parameters

t	An OID tag
a	The returned AVA object reference.

AVAList::get_ava_by_oid()

IDL

```
// Returns SUCCESSFUL or AVA_NOT_PRESENT
IT_Certificate::ReplyStatus
get_ava_by_oid(
    in ASN_OID seq,
    in UShort n,
    out AVA a
) raises(CertError);
```

Description This operation retrieves an AVA object from an AVAList, selected by the specified ASN_OID structure.

Parameters

seq	An ASN OID.
n	
a	The returned AVA object reference.

AVAList::get_num_avas()

IDL

```
UShort get_num_avas()
```

Description This operation retrieves the number of AVA objects in a `AVAList`.

IT_Certificate::Certificate Interface

IDL

```
// IDL in module IT_Certificate
interface Certificate
{
    readonly attribute DERData encoded_form;
};
```

Description This is the base interface for security certificate objects.

Certificate::encoded_form

IDL

```
readonly attribute DERData encoded_form;
```

Description This attribute returns the certificate data encoded in DER format.

IT_Certificate::Extension Interface

IDL

```
// IDL in module IT_Certificate
interface Extension
{
    readonly attribute UShort  critical;
    readonly attribute ASN_OID oid;

    // raises minor code IT_TLS_UNSUPPORTED_FORMAT
    Bytes convert(in Format f) raises(CertError);
};
```

Description

The `Extension` interface provides the developer with an interface to any X.509 version 3.0 extensions that an X.509 certificate can contain.

The `Extension` interface enables you to access the data for one particular extension. Using the `Extension::convert()` operations, the data can be converted into a number of representations.

Extension::convert()

IDL

```
Bytes convert(in Format f) raises(CertError);
```

Description

This operation returns data that corresponds to the contents of the `Extension` object converted to the requested format. The data is converted to the requested format and returned as an array of bytes.

Parameters This operation takes the following parameter:

`f` The format of the required conversion. The following `Format` values are supported:

`IT_FMT_DER`. This format corresponds to the DER encoding of the extension. This option is usually only used by applications that require special processing of the DER data.

`IT_FMT_STRING`. This format corresponds to a null terminated sequence of characters containing the actual data contained in the extension. This data has not been modified in any way, and may include non printable characters if present in the actual extension data. This is a regular 'C' string for printable string fields.

`IT_FMT_HEX_STRING`. This format contains a formatted hexadecimal dump of the DER data of the extension.

Extension::critical

IDL

```
readonly attribute UShort critical;
```

Description This attribute returns a non-zero value if the extension is critical; zero if the extension is not critical. A critical extension is an extension that should not be ignored by the authentication code.

Extension::oid

IDL

```
readonly attribute ASN_OID oid;
```

Description This attribute returns the ASN.1 OID for the extension. Extensions are identified by an ASN.1 OID, just like regular AVAs.

IT_Certificate::ExtensionList Interface

IDL

```
// IDL in module IT_Certificate
interface ExtensionList
{
    typedef sequence<Extension> ListOfExtensions;
    readonly    attribute ListOfExtensions    ext_list;

    UShort get_num_extensions();

    // Returns SUCCESSFUL or EXTENSION_NOT_PRESENT
    IT_Certificate::ReplyStatus
    get_extension_by_oid_tag(
        in OIDTag t,
        out Extension e
    ) raises(CertError);

    // Returns SUCCESSFUL or EXTENSION_NOT_PRESENT
    IT_Certificate::ReplyStatus
    get_extension_by_oid(
        in ASN_OID seq,
        in UShort n,
        out Extension e
    ) raises(CertError);

    // raises minor code IT_TLS_UNSUPPORTED_FORMAT
    Bytes convert(in Format f) raises(CertError);
};
```

Description

The `Extension` and `ExtensionList` interfaces provide you with access to any X.509 version three extensions.

The `Extension` interface provides an interface to accessing the data for one particular extension.

ExtensionList::convert()

IDL

```
Bytes convert(in Format f) raises(CertError);
```

Description convert() returns data in the requested format corresponding to the contents of the ExtensionList object. The operation returns this data as an array of bytes, or NULL if the the required conversion is not supported.

Note: Generally convert() is called on the individual extensions. This operation is not commonly used.

Parameters This operation takes the following parameter:

f — The format of the required conversion. The following Format value is supported:

IT_FMT_DER. This format corresponds to the DER encoding of the AVA. This option is usually only used by applications that require special processing of the DER data.

IT_FMT_STRING. This format corresponds to a null-terminated sequence of characters containing the actual data of the AVA. The data is not modified in any way, and can include non-printable characters if present in the actual AVA data. This is a string for normal printable string fields.

IT_FMT_HEX_STRING. This format corresponds to a formatted hexadecimal dump of the DER data of the AVA.

Exceptions

CertError, error code IT_TLS_UNSUPPORTED_FORMAT	An unknown format is specified.

ExtensionList::ext_list

IDL

```
readonly attribute ListOfExtensions ext_list;
```

Description This attribute returns the complete list of extensions as a sequence of Extension objects.

ExtensionList::get_extension_by_oid()

IDL

```
IT_Certificate::ReplyStatus
get_extension_by_oid(
    in ASN_OID seq,
    in UShort n,
    out Extension e
) raises(CertError);
```

Description Obtains the Extension element of the ExtensionList that has the requested object identifier, seq.

f the extension is found, a SUCCESSFUL reply status is returned; otherwise an EXTENSION_NOT_PRESENT reply status is returned.

Parameters This operation takes the following parameters

seq	An array of integers representing the ASN.1 object identifier.
n	The number of elements in the array.
e	The returned Extension object.

ExtensionList::get_extension_by_oid_tag()

IDL

```
IT_Certificate::ReplyStatus
get_extension_by_oid_tag(
    in OIDTag t,
    out Extension e
) raises(CertError);
```

Description Obtains the Extension element of the ExtensionList that corresponds to the supplied OIDTag value, t.

If the extension is found, a SUCCESSFUL reply status is returned; otherwise an EXTENSION_NOT_PRESENT reply status is returned.

Parameters

t	The OIDTag variable that identifies the extension to retrieve.
e	The returned Extension object.

ExtensionList::get_num_extensions();

IDL `UShort get_num_extensions();`

Description This operation returns the number of extensions in the list.

IT_Certificate::X509Cert Interface

IDL

```
// IDL in module IT_Certificate
interface X509Cert : IT_Certificate::Certificate
{
    exception IntegerTooLarge { };

    long    get_version();
    UTCTime get_not_before();
    UTCTime get_not_after();
    ASN_OID get_signature_algorithm_id();

    ULong get_serial_number()
    raises(
       CertError,
       IntegerTooLarge
    );

    DERData get_der_serial_number() raises (CertError);
    string  get_subject_dn_string() raises (CertError);
    string  get_issuer_dn_string()  raises (CertError);
    string
    get_subject_ava_string(in OIDTag t) raises (CertError);
    string
    get_issuer_ava_string(in OIDTag t)  raises (CertError);
    AVAList get_issuer_avalist()  raises (CertError);
    AVAList get_subject_avalist() raises(CertError);

    // Returns SUCCESSFUL or NO_EXTENSIONS_PRESENT
    IT_Certificate::ReplyStatus
    get_extensions(out ExtensionList el) raises (CertError);

    // raises minor code IT_TLS_UNSUPPORTED_FORMAT
    Bytes convert(in Format f) raises(CertError);
};
```

Description

The `X509Cert` interface provides a high-level interface to an X.509 certificate. A number of operations are provided to obtain information contained in the certificate. This interface, along with other certificate interfaces, shields the

developer from having to know about low-level details such as the encoding of X.509 certificates. Access to low-level DER information is, however, also provided.

For example, the `get_issuer_dn_string()`, `get_issuer_ava_string()`, `get_subject_dn_string()`, and `get_subject_ava_string()` provide easy access to the *issuer* and *subject* entries in a certificate. Typical issuer and subject entries have the following form:

```
issuer :/C=IE/ST=Co. Dublin/L=Dublin/O=IONA Technologies/OU=IDD/
   CN=IssuerName/Email=IssuerName@iona.com
subject:/C=IE/ST=Co. Dublin/O=IONA Technologies/OU=IDD/
   CN=SubjectName/Email=SubjectName@iona.com
```

X509Cert::convert()

IDL

```
Bytes
convert(in Format f) raises (CertError);
```

Description Converts the certificate to the specified format.

Parameters This operation takes the following parameter:

`f` The format of the required conversion. The following `Format` values are supported:

`IT_FMT_DER`. This format corresponds to the DER encoding of the extension. This option is usually only used by applications that require special processing of the DER data.

`IT_FMT_STRING`. This format corresponds to a null terminated sequence of characters containing the actual data contained in the extension. This data has not been modified in any way, and may include non printable characters if present in the actual extension data. This is a regular 'C' string for printable string fields.

`IT_FMT_HEX_STRING`. This format contains a formatted hexadecimal dump of the DER data of the extension.

X509Cert::get_der_serial_number()

IDL

```
DERData
get_der_serial_number() raises (CertError);
```

Description Obtains the serial number of the certificate and returns it as `DERData` object.

X509Cert::get_extensions()

IDL

```
IT_Certificate::ReplyStatus
get_extensions(out ExtensionList el) raises (CertError);
```

Description Obtains the complete extension list, `el`, for this certificate.

If the extensions are found, a `SUCCESSFUL` reply status is returned; otherwise an `NO_EXTENSIONS_PRESENT` reply status is returned.

Parameters

`el`	An out parameter containing the extension list as a sequence of `Extension` objects.

X509Cert::get_issuer_avalist()

IDL

```
AVAList
get_issuer_avalist() raises (CertError);
```

Description Retrieves the distinguished name of the certificate issuer as an `AVAList` instance. Individual components of the distinguished name (for example, the common name or the organization name) can be retrieved from the `AVAList` instance.

X509Cert::get_issuer_ava_string()

IDL

```
string
get_issuer_ava_string(in OIDTag t) raises (CertError);
```

Description Returns a string representing the AVA field selected by the `OIDTag`, `t`, from the certificate issuer AVA list.

X509Cert::get_issuer_dn_string()

IDL

```
string
get_issuer_dn_string() raises (CertError);
```

Description Returns a string representing the certificate issuer's distinguished name (DN).

X509Cert::get_not_after()

IDL

```
UTCTime get_not_after();
```

Description Extracts the `notAfter` field from an X.509 certificate. This field is used to determine the date validity of a certificate in conjunction with the `notBefore` field. A certificate can be specified as not valid until after some point in the future.

X509Cert::get_not_before()

IDL

```
UTCTime get_not_before();
```

Description Extracts the `notBefore` field from an X.509 certificate. This field is used in determining the date validity of a certificate in conjunction with the `notAfter` field. A certificate can be specified as not valid until some point in the future.

X509Cert::get_serial_number()

IDL

```
DERData
get_der_serial_number() raises (CertError);
```

Description Returns the serial number of the certificate.

X509Cert::get_signature_algorithm_id()

IDL

```
ASN_OID
get_signature_algorithm_id();
```

Description This operation returns the ASN.1 OID of the signature algorithm that was used to sign the certificate. For example, MD2, MD5, or SHA.

See Also

```
IT_Certificate::OIDTag
IT_Certificate::IT_OIDT_MD2
IT_Certificate::IT_OIDT_MD5
IT_Certificate::IT_OIDT_SHA
```

X509Cert::get_subject_avalist()

IDL

```
AVAList
get_subject_avalist() raises(CertError);
```

Description Returns the subject of the certificate in the form of an `AVAList`.

X509Cert::get_subject_ava_string()

IDL

```
string
get_subject_ava_string(in OIDTag t) raises (CertError);
```

Description Returns a string representing the AVA field selected by the OIDTag, `t`, from the certificate subject AVA list.

X509Cert::get_subject_dn_string()

IDL

```
string
get_subject_dn_string() raises (CertError);
```

Description Returns a string representing the certificate subject's distinguished name (DN).

X509Cert::get_version()

IDL

```
long get_version();
```

Description Returns the version minus one for the X.509 standard to which the certificate conforms. Hence, this operation returns 0 for v1, 1 for v2, and 2 for v3 in accordance with ASN.1 conventions. Most certificates conform to v3, which has support for AVA extensions.

X509Cert::IntegerTooLarge Exception

IDL `exception IntegerTooLarge`

Description Exception thrown in the unlikely case that an attempt to convert a DER encoded integer to the `CORBA::ULong` type fails because the specified DER encoded integer corresponds to a value that is too large to be represented by the DER encoded integer. In this unlikely case, the DER data form of the integer would have to be examined directly by the application.

IT_Certificate::X509CertificateFactory Interface

IDL

```
// IDL in module IT_Certificate
interface X509CertificateFactory
{
    // Following function creates x509Cert from DER data.
    // where DERData is a sequence of octets
    //
    // raises minor code IT_TLS_BAD_CERTIFICATE_DATA
    //
    X509Cert
    create_x509_certificate_from_der(
        in DERData der
    ) raises(CertError);

    //
    // Read CertList from a file.
    // raises minor code IT_TLS_BAD_CERTIFICATE_DATA.
    // raises minor code IT_TLS_ERROR_READING_DATA.
    //
    X509CertList
    load_x509_cert_list(
        in string location
    ) raises(CertError);
};
```

Description

This interface is a factory that generates X.509 certificates of IT_Certificate::X509Cert type.

This interface contains one operation, `create_x509_cert()`, that generates an X.509 certificate on receiving data in the form of DER.

X509CertificateFactory::create_x509_certificate_from_der()

IDL

```
X509Cert
```

```
create_x509_certificate_from_der(
    in DERData der
) raises(CertError);
```

Description Generates an X.509 certificate based on a parameter supplied in DER format, `der`.

Parameters This operation takes the following parameter:

`der`	The certificate data in DER format (of `DERData` type).

Exceptions

`CertError`, error code `IT_TLS_BAD_CERTIFICATE_DATA`	The `der` parameter is inconsistent or incorrectly formatted

X509CertificateFactory::load_x509_cert_list()

IDL

```
X509CertList
load_x509_cert_list(in string location) raises(CertError);
```

Description Generates a list of X.509 certificates based on data read from the file specified by `location`. The file must contain a chain of certificates in PEM format.

Parameters This operation takes the following parameter:

`location`	The absolute path name of the file containing the PEM certificate chain.

Exceptions

CertError, error code IT_TLS_ERROR_READING_DATA	Orbix cannot read the specified certificate file
CertError, error code IT_TLS_BAD_CERTIFICATE_DATA	The content of the certificate file is inconsistent or incorrectly formatted.

Security Overview

The standard `Security` module defines data types and constants that are used throughout the CORBA security specification. This section documents only the definitions relevant to Orbix SSL/TLS.

There is also a reference in Javadoc format.

Security::AssociationOptions Type

```
// IDL
typedef unsigned short AssociationOptions;
```

A data type that holds a set of association options in its bit fields.

See Also

`Security::NoProtection`
`Security::Integrity`
`Security::Confidentiality`
`Security::DetectReplay`
`Security::DetectMisordering`
`Security::EstablishTrustInTarget`
`Security::EstablishTrustInClient`
`Security::NoDelegation`
`Security::SimpleDelegation`
`Security::CompositeDelegation`

Security::AttributeList Sequence

```
// IDL
typedef sequence <SecAttribute> AttributeList;
```

Security::AuthenticationMethod Type

```
// IDL
typedef unsigned long AuthenticationMethod;
```

Constants of this type are used by the SecurityLevel2::PrincipalAuthenticator::authenticate() operation to identify an authentication method. Orbix SSL/TLS defines a range of AuthenticationMethod constants in the IT_TLS_API module—for example, IT_TLS_API::IT_TLS_AUTH_METH_PKSC12_FILE.

Security::AuthenticationMethodList Sequence

```
// IDL
typedef sequence<AuthenticationMethod> AuthenticationMethodList;
```

A list of authentication methods.

Security::AuthenticationStatus Enumeration

```
// IDL
enum AuthenticationStatus {
    SecAuthSuccess,
    SecAuthFailure,
    SecAuthContinue,
    SecAuthExpired
};
```

Used by the SecurityLevel2::PrincipalAuthenticator::authenticate() operation to give the status of the returned credentials.

Values

The status of a newly-generated Credentials object, creds, is indicated as follows:

SecAuthSuccess	A valid Credentials object is available in the creds parameter.
SecAuthFailure	Authentication was in some way inconsistent or erroneous. Credentials have therefore not been created.

SecAuthContinue	The authentication procedure uses a challenge and response mechanism. The creds parameter references a partially initialized Credentials object and the continuation_data indicates details of the challenge. Not supported by Orbix SSL/TLS.
SecAuthExpired	The authentication data, auth_data, has expired. Credentials have therefore not been created.

Security::CommunicationDirection Enumeration

```
// IDL
enum CommunicationDirection {
    SecDirectionBoth,
    SecDirectionRequest,
    SecDirectionReply
};
```

Indicates a particular communication direction along a secure association.

See Also SecurityLevel2::Credentials::get_security_feature()

Security::CompositeDelegation Constant

```
// IDL
const AssociationOptions CompositeDelegation = 512;
```

Not supported in Orbix SSL/TLS.

Security::Confidentiality Constant

```
// IDL
const AssociationOptions Confidentiality = 4;
```

Specifies that an object supports or requires confidentiality-protected invocations.

Security::DetectMisordering Constant

```
// IDL
const AssociationOptions DetectMisordering = 16;
```

Specifies that an object supports or requires error detection on fragments of invocation messages. In Orbix SSL/TLS this option can be set only through configuration.

Security::DetectReplay Constant

```
// IDL
const AssociationOptions DetectReplay = 8;
```

Specifies that an object supports or requires replay detection on invocation messages. In Orbix SSL/TLS this option can be set only through configuration.

Security::EstablishTrust Structure

```
// IDL
struct EstablishTrust {
   boolean trust_in_client;
   boolean trust_in_target;
};
```

Parameters

This structure is used to hold the data associated with the SecurityLevel2::EstablishTrustPolicy.

The elements of the structure are, as follows:

`trust_in_client`	Specifies whether or not an invocation must select credentials and a mechanism that allow the client to be authenticated to the target. (Some mechanisms might not support client authentication).
`trust_in_target`	Specifies whether or not an invocation must establish trust in the target.

Security::EstablishTrustInClient Constant

```
// IDL
const AssociationOptions EstablishTrustInClient = 64;
```

Specifies that a client supports or requires that the target authenticate its identity to the client.

See Also SecurityLevel2::EstablishTrustPolicy

Security::EstablishTrustInTarget Constant

```
// IDL
const AssociationOptions EstablishTrustInTarget = 32;
```

Specifies that a target object requires the client to authenticate its privileges to the target.

See Also SecurityLevel2::EstablishTrustPolicy

Security::Integrity Constant

```
// IDL
const AssociationOptions Integrity = 2;
```

Specifies that an object supports integrity-protected invocations.

Security::InvocationCredentialsType Enumeration

```
// IDL
enum InvocationCredentialsType {
    SecOwnCredentials,
    SecReceivedCredentials,
    SecTargetCredentials
};
```

Identifies the underlying type of a SecurityLevel2::Credentials object, as follows:

SecOwnCredentials	The underlying type is SecurityLevel2::Credentials.

SecReceivedCredentials	The underlying type is SecurityLevel2::ReceivedCredentials.
SecTargetCredentials	The underlying type is SecurityLevel2::TargetCredentials.

Security::MechanismType Type

```
// IDL
typedef string MechanismType;
```

Identifies a security mechanism.

See Also SecurityLevel2::MechanismPolicy

Security::MechanismTypeList Sequence

```
// IDL
typedef sequence<MechanismType> MechanismTypeList;
```

A list of security mechanisms.

See Also SecurityLevel2::MechanismPolicy

Security::NoDelegation Constant

```
// IDL
const AssociationOptions NoDelegation = 128;
```

Not supported in Orbix SSL/TLS.

Security::NoProtection Constant

```
// IDL
const AssociationOptions NoProtection = 1;
```

When used with the target secure invocation policy, indicates that the target can accept insecure connections.

When used with the client secure invocation policy, indicates that the client can open insecure connections.

Security::Opaque Type

```
// IDL
typedef sequence <octet> Opaque;
```

A general purpose type that is used to hold binary data.

Security::QOP Enumeration

```
// IDL
enum QOP {
    SecQOPNoProtection,
    SecQOPIntegrity,
    SecQOPConfidentiality,
    SecQOPIntegrityAndConfidentiality
};
```

Identifies the range of security features that can be associated with an individual object reference (quality of protection).

Values

`SecQOPNoProtection`	The `Security::NoProtection` association option.
`SecQOPIntegrity`	The `Security::Integrity` association option.
`SecQOPConfidentiality`	The `Security::Confidentiality` association option.
`SecQOPIntegrityAndConfidentiality`	Both the `Security::Integrity` and `Security::Confidentiality` association options.

Security::SecApplicationAccess Constant

```
// IDL
const CORBA::PolicyType SecApplicationAccess = 3;
```

Not supported in Orbix SSL/TLS.

Security::SecAttribute Structure

```
// IDL
struct SecAttribute {
    AttributeType attribute_type;
    OID defining_authority;
    Opaque value;
};
```

Security::SecClientInvocationAccess Constant

```
// IDL
const CORBA::PolicyType SecClientInvocationAccess = 1;
```

Not supported in Orbix SSL/TLS.

Security::SecClientSecureInvocation Constant

```
// IDL
const CORBA::PolicyType SecClientSecureInvocation = 8;
```

Defines one of the policy types for the `SecurityAdmin::SecureInvocationPolicy` interface. This policy can only be set through configuration.

Security::SecEstablishTrustPolicy Constant

```
// IDL
const CORBA::PolicyType SecEstablishTrustPolicy = 39;
```

Defines the policy type for the SecurityLevel2::EstablishTrustPolicy interface.

Security::SecInvocationCredentialsPolicy Constant

```
// IDL
const CORBA::PolicyType SecInvocationCredentialsPolicy = 13;
```

Defines the policy type for the SecurityLevel2::InvocationCredentialsPolicy interface.

Security::SecMechanismsPolicy Constant

```
// IDL
const CORBA::PolicyType SecMechanismsPolicy = 12;
```

Defines the policy type for the SecurityLevel2::MechanismsPolicy interface.

See Also IT_TLS_API::TLS::create_mechanism_policy()

Security::SecQOPPolicy Constant

```
// IDL
const CORBA::PolicyType SecQOPPolicy = 15;
```

Defines the policy type for the SecurityLevel2::QOPPolicy interface.

Security::SecTargetInvocationAccess Constant

```
// IDL
const CORBA::PolicyType SecTargetInvocationAccess = 2;
```

Not supported in Orbix SSL/TLS.

Security::SecTargetSecureInvocation Constant

```
// IDL
const CORBA::PolicyType SecTargetSecureInvocation = 9;
```

Defines one of the policy types for the `SecurityAdmin::SecureInvocationPolicy` interface. This policy can only be set through configuration.

Security::SecurityFeature Enumeration

```
// IDL
enum SecurityFeature {
    SecNoDelegation,
    SecSimpleDelegation,
    SecCompositeDelegation,
    SecNoProtection,
    SecIntegrity,
    SecConfidentiality,
    SecIntegrityAndConfidentiality,
    SecDetectReplay,
    SecDetectMisordering,
    SecEstablishTrustInTarget,
    SecEstablishTrustInClient
};
```

Identifies the range of security features that can be associated with a `Credentials` object, including association options.

Values This enumeration can have the following values:

`SecNoDelegation`	The `Security::NoDelegation` association option.
`SecSimpleDelegation`	The `Security::SimpleDelegation` association option. Not supported in Orbix SSL/TLS.
`SecCompositeDelegation`	The `Security::CompositeDelegation` association option. Not supported in Orbix SSL/TLS.
`SecNoProtection`	The `Security::NoProtection` association option.
`SecIntegrity`	The `Security::Integrity` association option.

SecConfidentiality	The Security::Confidentiality association option.
SecIntegrityAndConfidentiality	Both the Security::Integrity and Security::Confidentiality association options.
SecDetectReplay	The Security::DetectReplay association option.
SecDetectMisordering	The Security::DetectMisordering association option.
SecEstablishTrustInTarget	The Security::EstablishTrustInTarget association option.
SecEstablishTrustInClient	The Security::EstablishTrustInClient association option.

See Also SecurityLevel2::Credentials::get_security_feature()
Security::AssociationOptions

Security::SecurityName Type

```
// IDL
typedef string SecurityName;
```

A string that identifies a principal (for example, a login name).

Not used by Orbix SSL/TLS.

Security::SimpleDelegation Constant

```
// IDL
const AssociationOptions SimpleDelegation = 256;
```

Not supported in Orbix SSL/TLS.

SecurityLevel1 Overview

Because security level 1 is aimed at security-unaware applications, there is little IDL defined at this level—most of the security features are controlled by an administrator. Currently, there is one IDL interface defined at level 1:

- `SecurityLevel1::Current`

SecurityLevel1::Current Interface

IDL

```
// IDL in module SecurityLevel1
local interface Current : CORBA::Current { // Locality Constrained
    // thread specific operations
    Security::AttributeList get_attributes (
        in Security::AttributeTypeList attributes
    );
};
```

Description The `Current` object enables you to access information about the execution context. In Orbix SSL/TLS, it enables a server object to access a client's credentials.

Current::get_attributes()

IDL

```
Security::AttributeList get_attributes (
    in Security::AttributeTypeList attributes
);
```

Description Not implemented in Orbix SSL/TLS.

You can use the `Credentials::get_attributes()` operation instead.

See Also `SecurityLevel2::Current::received_credentials`
`SecurityLevel2::Credentials::get_attributes()`

SecurityLevel2 Overview

At security level 2, IDL interfaces are defined to enable security-aware application to access security information and specify security policies. Orbix SSL/TLS implements the following IDL interfaces from the `SecurityLevel2` IDL module:

- `PrincipalAuthenticator` interface.
- `Credentials` inteface.
- `ReceivedCredentials` interface.
- `TargetCredentials` interface.
- `QOPPolicy` interface.
- `MechanismPolicy` interface.
- `InvocationCredentialsPolicy` interface.
- `EstablishTrustPolicy` interface.
- `SecurityManager` interface.
- `Current` interface.

SecurityLevel2::CredentialsList Sequence

```
// IDL
typedef sequence <Credentials> CredentialsList;
```

A sequence to hold a list of `Credentials` objects.

SecurityLevel2::Credentials Interface

IDL

```
// IDL in module SecurityLevel2
interface Credentials {     // Locality Constrained
# pragma version Credentials 1.7
    Credentials copy();

    void destroy();

    readonly attribute Security::InvocationCredentialsType
        credentials_type;
    readonly attribute Security::AuthenticationStatus
        authentication_state;
    readonly attribute Security::MechanismType
        mechanism;

    attribute Security::AssociationOptions
        accepting_options_supported;
    attribute Security::AssociationOptions
        accepting_options_required;
    attribute Security::AssociationOptions
        invocation_options_supported;
    attribute Security::AssociationOptions
        invocation_options_required;

    boolean get_security_feature(
        in Security::CommunicationDirection direction,
        in Security::SecurityFeature feature
    );

    boolean set_attributes (
        in Security::AttributeList requested_attributes,
        out Security::AttributeList actual_attributes
    );

    Security::AttributeList get_attributes (
       in Security::AttributeTypeList attributes
    );
```

```
    boolean is_valid (out Security::UtcT expiry_time);

    boolean refresh(in any refresh_data);
};
```

Description The `Credentials` interface is used either as a base interface or as a concrete interface (most derived type is `Credentials`). An object of `Credentials` type can represent one of the following kinds of credential:

- *Own credentials*—when the most derived type of the `Credentials` object is `Credentials`.
- *Received credentials*—when the most derived type of the `Credentials` object is `ReceivedCredentials`.
- *Target credentials*—when the most derived type of the `Credentials` object is `TargetCredentials`.

A `Credentials` object holds the security attributes of a principal.

See Also
`IT_TLS_API::TLSCredentials`
`IT_TLS_API::TLSReceivedCredentials`
`IT_TLS_API::TLSTargetCredentials`

Credentials::accepting_options_required Attribute

IDL
```
attribute Security::AssociationOptions accepting_options_required;
```

Description Not implemented in Orbix SSL/TLS.

Credentials::accepting_options_supported Attribute

IDL
```
attribute Security::AssociationOptions
   accepting_options_supported;
```

Description Not implemented in Orbix SSL/TLS.

Credentials::authentication_state Attribute

IDL
```
readonly attribute Security::AuthenticationStatus
   authentication_state;
```

Description Specifies how a `Credentials` object is initialized (authentication state) at the time it is created by the `PrincipalAuthenticator` object.

Values The authentication state can have one of the following values:

`SecAuthSuccess`	The `Credentials` object is fully initialized and valid.
`SecAuthExpired`	The credentials initialization has expired and the credentials are invalid.

Credentials::copy()

IDL

```
Credentials copy();
```

Description Returns a reference to a deep copy of the target `Credentials` object.

Not implemented in Orbix SSL/TLS.

Credentials::credentials_type Attribute

IDL

```
readonly attribute Security::InvocationCredentialsType
    credentials_type;
```

Description Indicates whether the `Credentials` object represents an application's own credentials (of `Credentials` type), or received credentials (of `ReceivedCredentials` type), or target credentials (of `TargetCredentials` type).

Values This attribute can have one of the following values:

`Security::SecOwnCredentials`	Indicates own credentials
`Security::SecReceivedCredentials`	Indicates received credentials.
`Security::SecTargetCredentials`	indicates target credentials

Credentials::destroy()

IDL

```
void destroy();
```

Description Destroys the `Credentials` object.

Not implemented in Orbix SSL/TLS.

Credentials::get_attributes()

IDL

```
Security::AttributeList get_attributes(
    in AttributeTypeList attributes
);
```

Description Returns the security attributes from a `Credentials` object.

Parameters This operation takes the following parameter:

`attributes`	The set of security attributes (attributes and identities) whose values are desired. If this list is empty, all attributes are returned.

Credentials::get_security_feature()

IDL

```
boolean get_security_feature(
    in Security::CommunicationDirection direction,
    in Security::SecurityFeature feature
);
```

Description Not implemented in Orbix SSL/TLS.

Credentials:invocation_options_required Attribute

IDL

```
attribute Security::AssociationOptions
    invocation_options_required;
```

Description Not implemented in Orbix SSL/TLS.

Use `SecurityLevel2::QOPPolicy` programmatically or secure invocation policies in the configuration file instead.

Credentials::invocation_options_supported Attribute

IDL

```
attribute Security::AssociationOptions
    invocation_options_supported;
```

Description Not implemented in Orbix SSL/TLS.

Use `SecurityLevel2::QOPPolicy` programmatically or secure invocation policies in the configuration file instead.

Credentials::is_valid()

IDL

```
boolean is_valid(out Security::UtcT expiry_time);
```

Description Returns TRUE if the `Credentials` object is valid and FALSE otherwise.

Not implemented in Orbix SSL/TLS.

Credentials::mechanism Attribute

IDL

```
readonly attribute Security::MechanismType mechanism;
```

Description A string, of `Security::MechanismType` type, that identifies the underlying security mechanism.

Values Orbix SSL/TLS returns the string `20` which represents SSL/TLS.

See Also `IT_TLS_API::TLS::create_mechanism_policy()`

Credentials::refresh()

IDL

```
boolean refresh(in any refresh_data);
```

Description Not implemented in Orbix SSL/TLS.

Some security mechanisms allow you to extend the expiry time of a `Credentials` object by refreshing the credentials.

Credentials::set_attributes()

IDL

```
boolean set_attributes (
    in Security::AttributeList requested_attributes,
    out Security::AttributeList actual_attributes
);
```

Description Not implemented in Orbix SSL/TLS.

SecurityLevel2::Current Interface

IDL

```
// IDL in module SecurityLevel2
local interface Current : SecurityLevel1::Current {
# pragma version Current 1.7
    // Thread specific
    readonly attribute ReceivedCredentials received_credentials;
};
```

Description

The `Current` object accesses information about the execution context. In Orbix SSL/TLS, the level 2 `Current` interface provides received credentials (originating from a client) to a target object's execution context.

Current::received_credentials Attribute

IDL

```
readonly attribute ReceivedCredentials received_credentials;
```

At a target object, this thread-specific attribute is the credentials received from a client. They are the credentials of the authenticated principal that made the invocation.

If you have enabled Common Secure Interoperability (CSIv2), the `SecurityLevel2::Current::received_credentials()` operation returns the following credentials type:

- Propagated identity credentials, if present
- Authenticated credentials over the transport, if present and propagated identity credentials are not.
- Transport TLS credentials, if present and the above two are not.

See `IT_CSI::CSIReceivedCredentials` for more details.

Exceptions

In the case of a pure client, that is, an application that is not servicing an invocation on one of its objects, accessing the `received_credentials` attribute causes a `CORBA::BAD_OPERATION` exception to be raised.

SecurityLevel2::EstablishTrustPolicy Interface

IDL

```
// IDL in module SecurityLevel2
interface EstablishTrustPolicy : CORBA::Policy {
    readonly attribute EstablishTrust trust;
};
```

Description A policy of this type can be passed to the `set_policy_overrides()` operation to obtain an object reference that obeys the given trust policy.

The `EstablishTrustPolicy` object has a policy type of `Security::SecEstablishTrustPolicy` and is locality constrained.

EstablishTrustPolicy::trust Attribute

IDL

```
readonly attribute EstablishTrust trust;
```

Description The `trust` attribute is a structure that contains two members, each stipulating whether trust in the client and trust in the target is enabled.

SecurityLevel2::InvocationCredentials Policy Interface

IDL

```
// IDL in module SecurityLevel2
interface InvocationCredentialsPolicy : CORBA::Policy {
   readonly attribute CredentialsList creds;
};
```

Description A policy of this type can be passed to the `set_policy_overrides()` operation to obtain an object reference that uses the given credentials list, `creds`, for operation and attribute invocations.

The `InvocationCredentialsPolicy` object has a policy type of `Security::SecInvocationCredentialsPolicy` and is locality constrained.

InvocationCredentialsPolicy::creds

IDL

```
readonly attribute CredentialsList creds;
```

Description The list of `Credentials` objects associated with the `InvocationCredentialsPolicy` object.

SecurityLevel2::MechanismPolicy Interface

IDL

```
// IDL in module SecurityLevel2
interface MechanismPolicy : CORBA::Policy {
   readonly attribute Security::MechanismTypeList mechanisms;
};
```

Description A policy of this type can be passed to the `set_policy_overrides()` operation to obtain an object reference that uses the specified security mechanisms.

The `MechanismPolicy` object has a policy type of `Security::SecMechanismsPolicy` and is locality constrained.

See Also `IT_TLS_API::TLS::create_mechanism_policy()`

MechanismPolicy::mechanisms

IDL

```
readonly attribute Security::MechanismTypeList mechanisms;
```

Description The mechanisms, in the form of a `Security::MechanismTypeList`, associated with the `MechanismPolicy` object.

SecurityLevel2::PrincipalAuthenticator Interface

IDL

```
// IDL in module SecurityLevel2
interface PrincipalAuthenticator {  // Locality Constrained
#   pragma version PrincipalAuthenticator 1.5

    Security::AuthenticationMethodList
    get_supported_authen_methods(
        in Security::MechanismType mechanism
    );

    Security::AuthenticationStatus authenticate (
        in Security::AuthenticationMethod method,
        in Security::MechanismType mechanism,
        in Security::SecurityName security_name,
        in any auth_data,
        in Security::AttributeList privileges,
        out Credentials creds,
        out any continuation_data,
        out any auth_specific_data
    );

    Security::AuthenticationStatus continue_authentication (
        in any response_data,
        in Credentials creds,
        out any continuation_data,
        out any auth_specific_data
    );
};
```

Description

This interface provides operations to authenticate a principal and provide it with credentials. For example, the `authenticate()` operation is typically called when a user logs on to an application.

PrincipalAuthenticator::authenticate()

IDL

```
Security::AuthenticationStatus authenticate (
    in Security::AuthenticationMethod method,
    in Security::MechanismType mechanism,
    in Security::SecurityName security_name,
    in any auth_data,
    in Security::AttributeList privileges,
    out Credentials creds,
    out any continuation_data,
    out any auth_specific_data
);
```

Description

This operation is called to authenticate the principal. It can also request privilege attributes that the principal requires during its capsule-specific session with the system.

It creates a capsule-specific `Credentials` object including the required attributes and is placed on the `SecurityManager` object's `own_credentials` list according to the credential's mechanism type.

In Orbix SSL/TLS, a capsule is effectively identified with an ORB object. The main consequence of this is that credentials are not shared between ORB objects. If you create more than one ORB object in your application, you must call `authenticate()` for each ORB object to make credentials available to both ORBs.

Return Value

The return value indicates the status of the `creds` parameter:

`SecAuthSuccess`	A valid `Credentials` object is available in the `creds` parameter.
`SecAuthFailure`	Authentication was in some way inconsistent or erroneous. Credentials have therefore not been created.
`SecAuthContinue`	The authentication procedure uses a challenge and response mechanism. The `creds` parameter references a partially initialized `Credentials` object and the `continuation_data` indicates details of the challenge. Not supported by Orbix SSL/TLS.
`SecAuthExpired`	The authentication data, `auth_data`, has expired. Credentials have therefore not been created.

Parameters

`method`	The authentication method to use. For example, `IT_TLS_API::IT_TLS_AUTH_METH_PKCS12_FILE`. See the `IT_TLS_API` module for the complete list of authentication methods supported by Orbix SSL/TLS.
`mechanism`	The security mechanism for creating the returned `Credentials` object. Leave this parameter blank. It defaults to SSL/TLS.
`security_name`	The principal's identification information (such as login name). Not used by Orbix SSL/TLS.
`auth_data`	The principal's authentication information, typically consisting of a certificate, private key and pass phrase. The data inserted into the `auth_data` parameter depends on the specified authentication method, `method`.
`privileges`	The requested privilege attributes. Not supported by Orbix SSL/TLS.
`creds`	This parameter contains the locality constrained object reference of the newly created `Credentials` object. It is usable and placed on the `Current` object's `own_credentials` list only if the return value is `SecAuthSuccess`.
`continuation_data`	Not supported by Orbix SSL/TLS.
`auth_specific_data`	Not supported by Orbix SSL/TLS.

PrincipalAuthenticator::continue_authentication()

IDL

```
Security::AuthenticationStatus continue_authentication (
    in any response_data,
    in Credentials creds,
    out any continuation_data,
    out any auth_specific_data
);
```

Description Not supported by Orbix SSL/TLS.

PrincipalAuthenticator::get_supported_authen_methods()

IDL

```
Security::AuthenticationMethodList
get_supported_authen_methods(
    in Security::MechanismType mechanism
);
```

Description Not implemented in Orbix SSL/TLS.

SecurityLevel2::QOPPolicy Interface

IDL

```
// IDL in module SecurityLevel2
interface QOPPolicy : CORBA::Policy {
   readonly attribute Security::QOP qop;
};
```

Description A QOP policy object can be passed to the `set_policy_overrides()` operation to obtain an object reference that uses the specified quality of protection policy.

See Also `Security::SecQOPPolicy`

QOPPolicy::qop Attribute

IDL

```
readonly attribute Security::QOP qop;
```

Description The quality of protection, of `Security::QOP` enumeration type, associated with the `QOPPolicy` object.

SecurityLevel2::ReceivedCredentials Interface

IDL

```
// IDL in module SecurityLevel2
interface ReceivedCredentials : Credentials {
# pragma version ReceivedCredentials 1.5
    readonly attribute Credentials accepting_credentials;

    readonly attribute Security::AssociationOptions
        association_options_used;

    readonly attribute Security::DelegationState delegation_state;

    readonly attribute Security::DelegationMode delegation_mode;
};
```

Description A `ReceivedCredentials` object stores the security attributes of a remote client. It is made available in an execution context on the server side and can be obtained from a `SecurityLevel2::Current` object.

See Also `SecurityLevel2::Current`
`IT_TLS_API::TLSReceivedCredentials`

ReceivedCredentials::accepting_credentials Attribute

IDL

```
readonly attribute Credentials accepting_credentials;
```

Description Not implemented in Orbix SSL/TLS.

ReceivedCredentials::association_options_used Attribute

IDL

```
readonly attribute Security::AssociationOptions
    association_options_used;
```

Description Not implemented in Orbix SSL/TLS.

ReceivedCredentials::delegation_mode Attribute

IDL `readonly attribute Security::DelegationMode delegation_mode;`

Description Not implemented in Orbix SSL/TLS.

ReceivedCredentials::delegation_state Attribute

IDL `readonly attribute Security::DelegationState delegation_state;`

Description Not implemented in Orbix SSL/TLS.

SecurityLevel2::SecurityManager Interface

IDL

```
// IDL in module SecurityLevel2
interface SecurityManager {
    readonly attribute Security::MechandOptionsList
        supported_mechanisms;
    readonly attribute CredentialsList own_credentials;
    readonly attribute RequiredRights required_rights_object;
    readonly attribute PrincipalAuthenticator
        principal_authenticator;

    readonly attribute AccessDecision access_decision;
    readonly attribute AuditDecision audit_decision;

    TargetCredentials get_target_credentials (in Object obj_ref);

    void remove_own_credentials(in Credentials creds);

    CORBA::Policy get_security_policy (
        in CORBA::PolicyType policy_type
    );
};
```

Description In Orbix SSL/TLS, this class is used to access ORB-specific information.

SecurityManager::access_decision Attribute

IDL

```
readonly attribute AccessDecision access_decision;
```

Description Not implemented in Orbix SSL/TLS.

SecurityManager::audit_decision Attribute

IDL

```
readonly attribute AuditDecision audit_decision;
```

Description Not implemented in Orbix SSL/TLS.

SecurityManager::get_security_policy()

IDL

```
CORBA::Policy get_security_policy (
    in CORBA::PolicyType policy_type
);
```

Description Not implemented in Orbix SSL/TLS.

SecurityManager::get_target_credentials()

IDL

```
TargetCredentials get_target_credentials(
   in Object target;
};
```

Description Returns the target credentials for an object referenced by the specified object reference, `target`. For example, this operation is typically used on the client side to obtain the target credentials for a remote object.

Parameters

`target`	An object reference.

SecurityManager::own_credentials Attribute

IDL

```
readonly attribute CredentialsList own_credentials;
```

Description Holds an application's own credentials, which are established by calling `authenticate()` on the application's own `PrincipalAuthenticator` object.

SecurityManager::principal_authenticator Attribute

IDL

```
readonly attribute PrincipalAuthenticator principal_authenticator;
```

Description Holds a reference to the `PrincipalAuthenticator` object that can be used by the application to authenticate principals and obtain credentials.

SecurityManager::remove_own_credentials()

IDL

```
void remove_own_credentials(
   in Credentials creds
);
```

Description Removes credentials that were put on the own_credentials list using the PrincipalAuthenticator. This operation does not manipulate or destroy the objects in any way.

Parameters

creds	The Credentials object to be removed from the list.

SecurityManager::required_rights_object Attribute

IDL

```
readonly attribute RequiredRights required_rights_object;
```

Description Not implemented in Orbix SSL/TLS.

SecurityManager::supported_mechanisms Attribute

IDL

```
readonly attribute Security::MechandOptionsList
    supported_mechanisms;
```

Description Not implemented in Orbix SSL/TLS.

SecurityLevel2::TargetCredentials Interface

IDL

```
// IDL in module SecurityLevel2
interface TargetCredentials : Credentials {
    readonly attribute Credentials
        initiating_credentials;

    readonly attribute Security::AssociationOptions
        association_options_used;
};
```

Description A `TargetCredentials` object holds the security attributes of an authenticated target object. To obtain the target credentials for a remote object, call the `SecurityManager::get_target_credentials()` operation.

See Also `IT_TLS_API::TLSTargetCredentials`

TargetCredentials::association_options_used Attribute

IDL

```
readonly attribute Security::AssociationOptions
    association_options_used;
```

Description Not implemented in Orbix SSL/TLS.

TargetCredentials::initiating_credentials Attribute

IDL

```
readonly attribute Credentials initiating_credentials;
```

Description Not implemented in Orbix SSL/TLS.

IT_TLS Overview

The IT_TLS module defines a single IDL interface, as follows:

- IT_TLS::CertValidator

The following data types are defined in the scope of IT_TLS to describe certificate validation errors:

- IT_TLS::CertChainErrorCode enumeration
- IT_TLS::CertChainErrorInfo structure.

IT_TLS::CACHE_NONE Constant

```
const SessionCachingMode CACHE_NONE = 0;
```

A flag that specifies no caching.

See Also IT_TLS_API::SessionCachingPolicy

IT_TLS::CACHE_SERVER Constant

```
const SessionCachingMode CACHE_SERVER = 0x01;
```

A flag that specifies server-side caching only.

See Also IT_TLS_API::SessionCachingPolicy

IT_TLS::CACHE_CLIENT Constant

```
const SessionCachingMode CACHE_CLIENT = 0x02;
```

A flag that specifies client-side caching only.

See Also IT_TLS_API::SessionCachingPolicy

IT_TLS::CACHE_SERVER_AND_CLIENT Constant

```
const SessionCachingMode CACHE_SERVER_AND_CLIENT = 0x04;
```

A flag that specifies both server-side and client-side caching.

See Also IT_TLS_API::SessionCachingPolicy

IT_TLS::CertChainErrorCode Enumeration

```
//IDL
enum CertChainErrorCode
{
    CERTIFICATE_UNKNOWN,
    CERTIFICATE_DECODE_ERROR,
    CERTIFICATE_SIGNED_BY_UNKNOWN_CA,
    UNSUPPORTED_CERTIFICATE,
    CERTIFICATE_EXPIRED,
    CERTIFICATE_NOT_YET_VALID,
    CERTIFICATE_REVOKED,
    BAD_CERTIFICATE,
    CERTIFICATE_SIGNED_BY_NON_CA_CERTIFICATE,
    CERTIFICATE_CHAIN_TOO_LONG,
    CERTIFICATE_FAILED_CONSTRAINTS_VALIDATION,
    CERTIFICATE_FAILED_APPLICATION_VALIDATION,
    CERTIFICATE_SUBJECT_ISSUER_MISMATCH
};
```

An Orbix-specific error code that gives the reason why a certificate failed to validate.

IT_TLS::CertChainErrorInfo Structure

```
//IDL
struct CertChainErrorInfo
{
    short         error_depth;
    string        error_message;

    CertChainErrorCode error_reason;

    // If this field is true then the two subsequent field may be
```

```
    // examined to get more detail from the underlying toolkit if
    // required. These are non portable values and are only ever
    // likely to be used for diagnostic  purposes.
    boolean  external_error_set;
    short    external_error_depth;
    long     external_error;
    string   external_error_string;
};
```

This structure is initialized with error information if a certificate chain fails the validation checks made by Orbix SSL/TLS. Two different levels of error information are generated by the Orbix SSL/TLS runtime:

- Error information generated by Orbix SSL/TLS—provided by the `error_depth`, `error_message`, and `error_reason` members.
- Error information generated by an underlying third-party toolkit—provided by the `external_error_depth`, `external_error`, and `external_error_string` members.

The structure contains the following elements:

`error_depth`	A positive integer that indexes the chain depth of the certificate causing the error. Zero indicates the peer certificate.
`error_message`	A descriptive error string (possibly from the lower level toolkit).
`error_reason`	An Orbix-specific error code.
`external_error_set`	If TRUE, external error details are provided by the underlying toolkit in the member variables following this one.
`external_error_depth`	The index of the certificate that caused the error, as counted by the underlying toolkit.
`external_error`	The error code from the underlying toolkit.
`external_error_string`	A descriptive error string from the underlying toolkit.

IT_TLS::CipherSuite Type

```
typedef unsigned long CipherSuite;
```

A type that identifies a cipher suite.

Values The following constants of `IT_TLS::CipherSuite` type are defined in `IT_TLS`:

```
TLS_RSA_WITH_NULL_MD5
TLS_RSA_WITH_NULL_SHA
TLS_RSA_EXPORT_WITH_RC4_40_MD5
TLS_RSA_WITH_RC4_128_MD5
TLS_RSA_WITH_RC4_128_SHA
TLS_RSA_EXPORT_WITH_RC2_CBC_40_MD5
TLS_RSA_WITH_IDEA_CBC_SHA
TLS_RSA_EXPORT_WITH_DES40_CBC_SHA
TLS_RSA_WITH_DES_CBC_SHA
TLS_RSA_WITH_3DES_EDE_CBC_SHA
TLS_DH_DSS_EXPORT_WITH_DES40_CBC_SHA
TLS_DH_DSS_WITH_DES_CBC_SHA
TLS_DH_DSS_WITH_3DES_EDE_CBC_SHA
TLS_DH_RSA_EXPORT_WITH_DES40_CBC_SHA
TLS_DH_RSA_WITH_DES_CBC_SHA
TLS_DH_RSA_WITH_3DES_EDE_CBC_SHA
TLS_DHE_DSS_EXPORT_WITH_DES40_CBC_SHA
TLS_DHE_DSS_WITH_DES_CBC_SHA
TLS_DHE_DSS_WITH_3DES_EDE_CBC_SHA
TLS_DHE_RSA_EXPORT_WITH_DES40_CBC_SHA
TLS_DHE_RSA_WITH_DES_CBC_SHA
TLS_DHE_RSA_WITH_3DES_EDE_CBC_SHA
TLS_DH_ANON_EXPORT_WITH_RC4_40_MD5
TLS_DH_ANON_WITH_RC4_128_MD5
TLS_DH_ANON_EXPORT_WITH_DES40_CBC_SHA
TLS_DH_ANON_WITH_DES_CBC_SHA
TLS_DH_ANON_WITH_3DES_EDE_CBC_SHA
TLS_FORTEZZA_DMS_WITH_NULL_SHA
TLS_FORTEZZA_DMS_WITH_FORTEZZA_CBC_SHA
```

IT_TLS::CipherSuiteList Sequence

```
typedef sequence<CipherSuite> CipherSuiteList;
```

A list of cipher suites.

IT_TLS::SessionCachingMode Type

```
typedef unsigned short SessionCachingMode;
```

A type that holds a session caching mode flag.

See Also IT_TLS_API::SessionCachingPolicy

IT_TLS::CertValidator Interface

IDL

```
// IDL in module IT_TLS
interface CertValidator
{
    boolean
    validate_cert_chain(
        in boolean                      chain_is_valid,
        in IT_Certificate::X509CertChain cert_chain,
        in CertChainErrorInfo           error_info
    );
};
```

Description

The `CertValidator` interface is a callback interface that can be used to check the validity of a certificate chain. A developer can provide custom validation for secure associations by implementing the `CertValidator` interface, defining the `validate_cert_chain()` operation to do the checking. The developer then creates an instance of the custom `CertValidator` and registers the callback by setting an `IT_TLS_API::TLS_CERT_VALIDATOR_POLICY` policy.

CertValidator::validate_cert_chain()

IDL

```
boolean
validate_cert_chain(
    in boolean                      chain_is_valid,
    in IT_Certificate::X509CertChain cert_chain,
    in CertChainErrorInfo           error_info
);
```

Description

Returns TRUE if the implementation of `validate_cert_chain()` considers the certificate chain to be valid; otherwise returns FALSE.

Parameters

`chain_is_valid`	TRUE if the certificate chain has passed the validity checks made automatically by the Orbix SSL/TLS toolkit; otherwise FALSE.
`cert_chain`	The X.509 certificate chain to be checked.
`error_info`	If the certificate chain has failed the validity checks made by Orbix SSL/TLS, this parameter provides details of the error in the certificate chain.

IT_TLS_API Overview

The IT_TLS_API module defines Orbix-specific security policies and an interface, TLS, that acts as a factory for certain kinds of security policy. This module contains the following IDL interfaces:

- CertConstraintsPolicy Interface
- CertValidatorPolicy Interface
- MaxChainLengthPolicy Interface
- SessionCachingPolicy Interface
- TrustedCAListPolicy Interface
- TLS Interface
- TLSCredentials Interface
- TLSReceivedCredentials Interface
- TLSTargetCredentials Interface

Associated with each of the security policies, the IT_TLS_API module defines the following policy type constants (of CORBA::PolicyType type):

```
IT_TLS_API::TLS_CERT_CONSTRAINTS_POLICY
IT_TLS_API::TLS_CERT_VALIDATOR_POLICY
IT_TLS_API::TLS_MAX_CHAIN_LENGTH_POLICY
IT_TLS_API::TLS_SESSION_CACHING_POLICY
IT_TLS_API::TLS_TRUSTED_CA_LIST_POLICY
```

The IT_TLS_API module also defines IDL structures that are used to supply authentication information to the PrincipalAuthenticator::authenticate() operation, depending on the authentication method used. The following structures are defined:

- PasswordAuthData
- PEMCertChainFileAuthData
- PKCS12DERAuthData
- PKCS12FileAuthData
- X509CertChainAuthData
- PKCS11AuthData

Associated with each of the authentication structures, the IT_TLS_API module defines the following authentication method constants (of Security::AuthenticationMethod type):

Table 32: *Authentication Method Constants and Authentication Structures*

Authentication Method Constant	Authentication Structure
IT_TLS_AUTH_METH_PASSWORD	PasswordAuthData
IT_TLS_AUTH_METH_CERT_CHAIN_FILE	PEMCertChainFileAuthData
IT_TLS_AUTH_METH_PKCS12_DER	PKCS12DERAuthData
IT_TLS_AUTH_METH_PKCS12_FILE	PKCS12FileAuthData
IT_TLS_AUTH_METH_CERT_CHAIN	X509CertChainAuthData
IT_TLS_AUTH_METH_PKCS11	PKCS11AuthData

IT_TLS_API::CertConstraints Sequence

IDL

```
typedef sequence<string> CertConstraints;
```

Holds a list of certificate constraints for a certificate constraints policy.

See Also IT_TLS_API::CertConstraintsPolicy

IT_TLS_API::PasswordAuthData

IDL

```
struct PasswordAuthData {
    string password;
};
```

Supplies only a password as authentication data.

Notes Reserved for future use.

IT_TLS_API::PEMCertChainFileAuthData

IDL

```
struct PEMCertChainFileAuthData {
    string    password;
    string    filename;
};
```

Supplies a password and the file name of a privacy-enhanced mail (PEM) encrypted X.509 certificate chain.

Notes Reserved for future use.

IT_TLS_API::PKCS12DERAuthData

IDL

```
struct PKCS12DERAuthData {
    string    password;
    IT_Certificate::DERData  cert_chain;
};
```

Supplies a password and a certificate chain in DER format.

Notes Reserved for future use.

IT_TLS_API::PKCS12FileAuthData

IDL

```
struct PKCS12FileAuthData {
    string password;
    string filename;
};
```

Supplies a password and the file name of a PKCS#12 encrypted X.509 certificate chain. The file name should be an absolute path name.

IT_TLS_API::X509CertChainAuthData

IDL

```
struct X509CertChainAuthData {
     IT_Certificate::DERData       private_key;
     IT_Certificate::X509CertChain cert_chain;
};
```

Supplies an asymmetric private key and an X.509 certificate chain.

IT_TLS_API::PKCS11AuthData

IDL

```
struct PKCS11AuthData {
    string                          provider;
    string                          slot;
    string                          pin;
};
```

Supplies the provider name, slot number, and PIN for a smart card that is accessed through a PKCS #11 interface. In this case, the user's private key and certificate chain are stored on the smart card. The PIN is used to gain access to the smart card.

IT_TLS_API::CertConstraintsPolicy Interface

IDL

```
// IDL in module IT_TLS_API
local interface CertConstraintsPolicy : CORBA::Policy
{
   readonly attribute CertConstraints cert_constraints;
};
```

Description This policy defines a list of constraints to be applied to certificates. This policy type is identified by the IT_TLS_API::TLS_CERT_CONSTRAINTS_POLICY policy type constant.

CertConstraintsPolicy::cert_constraints Attribute

IDL

```
readonly attribute CertConstraints cert_constraints;
```

Description Holds the list of certificate constraints as a sequence of strings, of IT_TLS_API::CertConstraints type.

IT_TLS_API::CertValidatorPolicy Interface

IDL

```
// IDL in module IT_TLS_API
local interface CertValidatorPolicy : CORBA::Policy
{
    readonly attribute IT_TLS::CertValidator    cert_validator;
};
```

Description This policy can be used to register a customized certificate callback object, of IT_TLS::CertValidator type. This policy type is identified by the IT_TLS_API::TLS_CERT_VALIDATOR_POLICY policy type constant.

CertValidatorPolicy::cert_validator Attribute

IDL

```
readonly attribute IT_TLS::CertValidator    cert_validator;
```

Description Holds the customized certificate callback object, of IT_TLS::CertValidator type

IT_TLS_API::MaxChainLengthPolicy Interface

IDL

```
// IDL in module IT_TLS_API
local interface MaxChainLengthPolicy : CORBA::Policy
{
    readonly attribute unsigned short max_chain_length;
};
```

Description This is a simple integer-based policy that controls the maximum certificate chain length permitted. The policy is applicable to servers and clients. This policy type is identified by the `IT_TLS_API::TLS_MAX_CHAIN_LENGTH_POLICY` policy type constant.

Notes Default is 2.

MaxChainLengthPolicy::max_chain_length Attribute

IDL

```
readonly attribute unsigned short max_chain_length;
```

Description Holds the maximum chain length value.

IT_TLS_API::SessionCachingPolicy Interface

IDL

```
// IDL in module IT_TLS_API
local interface SessionCachingPolicy : CORBA::Policy{
    readonly attribute unsigned short cache_mode;
};
```

Description

An Orbix-specific policy to specify the caching mode. This policy applies to clients and servers. This policy type is identified by the IT_TLS_API::TLS_SESSION_CACHING_POLICY policy type constant.

Session caching is an Orbix-specific feature that enables secure associations (for example, over TCP/IP connections) to be re-established more quickly after being closed.

To enable session caching for a client-server connection, the client must support client-side caching (CACHE_CLIENT or CACHE_SERVER_AND_CLIENT policy) and the server must support server-side caching (CACHE_SERVER or CACHE_SERVER_AND_CLIENT policy). The first time a secure association is established between the client and the server, session information is cached at both ends of the association. If the association is subsequently closed and re-established (as can happen when Automatic Connection Management is enabled), the reconnection occurs more rapidly because some of the steps in the security handshake can be skipped.

The caching optimization is effective only if both client and server are running continuously between the closing and the re-establishment of the connection. Session caching data is not stored persistently and is, therefore, not available to restarted applications.

Each TLS listener uses a separate session cache. For example, if you have two POAs with different InvocationCredentialsPolicy values, Orbix SSL/TLS creates a TLS listener and session cache for each POA.

A client will not offer a cached session for reuse to a server if the session was initially created with different effective security policies.

SessionCachingPolicy::cache_mode Attribute

IDL

```
readonly attribute unsigned short cache_mode;
```

Description Holds the client caching mode. The default value is IT_TLS::CACHE_NONE.

Values The values for this policy are as follows:

IT_TLS::CACHE_NONE	No caching.
IT_TLS::CACHE_SERVER	Perform server-side caching only.
IT_TLS::CACHE_CLIENT	Perform client-side caching only.
IT_TLS::CACHE_SERVER_AND_CLIENT	Perform both server-side and client-side caching.

IT_TLS_API::TLS Interface

IDL

```
// IDL in module IT_TLS_API
local interface TLS  {
    SecurityLevel2::MechanismPolicy
    create_mechanism_policy(
        in IT_TLS::CipherSuiteList ciphersuite_list
    );
};
```

Description This interface provides helper operations for the TSL module.

TLS::create_mechanism_policy()

IDL

```
SecurityLevel2::MechanismPolicy
create_mechanism_policy(
    in IT_TLS::CipherSuiteList ciphersuite_list
);
```

Description Creates a `SecurityLevel2::MechanismPolicy` object from a list of ciphersuites, `ciphersuite_list`.

See Also `IT_TLS::CipherSuite`

IT_TLS_API::TLSCredentials Interface

```
// IDL
local interface TLSCredentials : SecurityLevel2::Credentials
{
    IT_Certificate::X509Cert get_x509_cert();

    IT_Certificate::X509CertChain get_x509_cert_chain_nc();

    IT_Certificate::X509CertChain get_x509_cert_chain();
};
```

Description This interface is the base interface for the `IT_TLS_API::TLSReceivedCredentials` and the `IT_TLS_API::TLSTargetCredentials` interfaces. The interface defines operations to retrieve an X.509 certificate chain from the credentials.

TLSCredentials::get_x509_cert()

```
// IDL
IT_Certificate::X509Cert get_x509_cert();
```

Description Returns a reference to the X.509 peer certificate (first certificate in the chain) contained in the credentials.

TLSCredentials::get_x509_cert_chain()

```
// IDL
IT_Certificate::X509CertChain get_x509_cert_chain();
```

Description Returns a copy of the X.509 certificate chain contained in the credentials. In C++ applications it is preferable to use the non-copying operation `get_x509_cert_chain_nc()` for greater efficiency.

TLSCredentials::get_x509_cert_chain_nc()

```
// IDL
IT_Certificate::X509CertChain get_x509_cert_chain_nc();
```

Description Returns a reference to the X.509 certificate chain contained in the credentials. In C++, the mapped function, `get_x509_cert_chain_nc()`, does not make a deep copy of the certificate sequence. The returned X.509 certificate chain can only be used while the credential from which it was obtained remains in memory.

IT_TLS_API::TLSReceivedCredentials Interface

IDL

```
local interface TLSReceivedCredentials :
    TLSCredentials,
    SecurityLevel2::ReceivedCredentials
{
};
```

Description

The interface of an Orbix-specific received credentials object, which inherits from the standard `SecurityLevel2::ReceivedCredentials` interface. `TLSReceivedCredentials` provides extra operations (inherited from `IT_TLS_API::TLSCredentials`) to extract the X.509 certificate chain from the credentials.

An instance of a `TLSReceivedCredentials` object can be obtained by narrowing the `SecurityLevel2::ReceivedCredentials` object reference obtained from the `SecurityLevel2::Current::received_credentials` attribute.

IT_TLS_API::TLSTargetCredentials Interface

IDL

```
local interface TLSTargetCredentials :
    TLSCredentials,
    SecurityLevel2::TargetCredentials
{
};
```

Description

The interface of an Orbix-specific target credentials object, which inherits from the standard `SecurityLevel2::TargetCredentials` interface. `TLSTargetCredentials` provides extra operations (inherited from `IT_TLS_API::TLSCredentials`) to extract the X.509 certificate chain from the credentials.

An instance of a `TLSTargetCredentials` object can be obtained by narrowing the `SecurityLevel2::TargetCredentials` object reference returned from the `SecurityLevel2::SecurityManager::get_target_credentials()` operation.

IT_TLS_API::TrustedCAListPolicy Interface

IDL

```
local interface TrustedCAListPolicy : CORBA::Policy
{
    readonly attribute IT_Certificate::X509CertList
        trusted_ca_list;
};
```

Description This policy specifies a list of trusted CA certificates. The policy is applicable to both servers and clients. This policy type is identified by the IT_TLS_API::TLS_TRUSTED_CA_LIST_POLICY policy type constant.

TrustedCAListPolicy::trusted_ca_list Attribute

IDL

```
readonly attribute IT_Certificate::X509CertList trusted_ca_list;
```

Description Holds the list of trusted CA certificates.

Telecom Logging Service

Telecom Logging Service Interfaces

The Telecom Logging Service is defined in terms of the following IDL modules:

Table 33: *Notification IDL Modules*

IDL Module	Contents
DsLogAdmin	Defines the exceptions, datatypes, and the base interfaces that implement the telecom logging service.
DsEventLogAdmin	Defines the EventLog and EventLogFactory interfaces.
DsNotifyLogAdmin	Defines the NotifyLog and NotifyLogFactory interfaces.
DsLogNotification	Defines the data sturctures to support log generated evetns.
IT_LogAdmin	Provides additional configuration defaults and management interfaces specific to IONA's telecom logging service implementation.
IT_NotifyLogAdmin	Extends the OMG specified NotifyLog and NotifyLogFactory interfaces to support event subsrciption and publication. Also provides access to a default filter factory.

The following chapters describe these modules in detail.

DsLogAdmin Module

DsLogAdmin specifies the Log interfaces which forms the basis for the BasicLog interface, EventLog interface, and the NotifyLog interface. DsLogAdmin also specifies the BasicLog and BasicLogFactory to support the basic logging service. In addtion, this module specifys the Iterator interface to support the iterators returned when retrieving records from a log.

This module also specifies all of the exceptions and major datatypes used by the telecom logging service.

DsLogAdmin Exceptions

DsLogAdmin::InvalidParam Exception

```
exception InvalidParam {string details;};
```

Raised when an illegal value is used to set a log's properties. It contains the name of the property being set and the illegal value.

DsLogAdmin::InvalidThreshold Exception

```
exception InvalidThreshold {};
```

Raised when an attempt is made to set a threshold alarm at a value outside the range of 0%-99%.

DsLogAdmin::InvalidTime Exception

```
exception InvalidTime{};
```

Raised by set_week_mask() when one of the values specified for a start or stop time is not within the valid range.

DsLogAdmin::InvalidTimeInterval Exception

```
exception InvalidTimeInterval{};
```

Raised by `set_week_mask()` when one of the time intervals used to set a log's schedule is improperly formed. For example, the stop time is before the start. Also raised if the intervals overlap.

DsLogAdmin::InvalidMask Exception

```
exception InvalidMask{};
```

Raised by `set_week_mask()` when the days parameter used in setting a log's schedule is malformed.

DsLogAdmin::LogIdAlreadyExists Exception

```
exception LogIdAlreadyExists{};
```

Raised by `create_with_id()` if an attempt is made to create a log with an id that is already in use.

DsLogAdmin::InvalidGrammar Exception

```
exception InvalidGrammar{};
```

Raised by `query()` and `delete_records()` if an unsupported constraint grammar is specified. The grammar implemented in Iona's telecom logging service is `EXTENDED_TCL`.

DsLogAdmin::InvalidConstraint Exception

```
exception InvalidConstraint{};
```

Raised by `query()` and `delete_records()` if a constraint expression is not syntactically correct according to the specified grammar.

DsLogAdmin::LogFull Exception

```
exception LogFull{short n_records_written;};
```

Raised when an attempt is made to log records in a log that is full and has its `full_action` set to `halt`. It returns the number of records that were successfully written to the log.

DsLogAdmin::LogOffDuty Exception

```
exception LogOffDuty{};
```

Raised when an attempt is made to log records in a log whose availability status is off duty.

DsLogAdmin::LogLocked Exception

```
exception LogLocked{};
```

Raised when an attempt is made to log records in a log whose administrative state is `locked`.

DsLogAdmin::LogDisabled Exception

```
exception LogDisabled{};
```

Raised when an attempt is made to log records in a log whose operational state is `disabled`.

DsLogAdmin::InvalidRecordId Exception

```
exception InvalidRecordId{};
```

Raised when the record id specified does not exist in the log.

DsLogAdmin::InvalidAttribute Exception

```
exception InvalidAttribute{string attr_name; any value;};
```

Raised when one of the attributes set on a record is invalid. It returns the name of the invalid attribute and the value specified for it.

DsLogAdmin::InvalidLogFullAction Exception

```
exception InvalidLogFullAction{};
```

Raised if an attempt is made to set a log's `full_action` to a value other than `wrap` or `halt`.

DsLogAdmin::UnsupportedQoS Exception

```
exception UnsupportedQoS{QoSList denied};
```

DsLogAdmin Constants

`DsLogAdmin` defines the majority of the constant values used when developing a telecom logging service application.

Querying Constants

`DsLogAdmin` defines one constant to support queries:

```
const string default_grammar = "EXTENDED_TCL";
```

Full Action Constants

Two constants are defined to support a log's `full_action`:

```
const LogFullActionType wrap = 0;
const LogFullActionType halt = 1;
```

Scheduling Constants

DsLogAdmin defines the following constants to support log scheduling:

```
const unsigned short Sunday    =  1;
const unsigned short Monday    =  2;
const unsigned short Tuesday   =  4;
const unsigned short Wednesday =  8;
const unsigned short Thursday  = 16;
const unsigned short Friday    = 32;
const unsigned short Saturday  = 64;
```

QoS Constants

DsLogAdmin defines the following constants to support log QoS properties:

```
const QoSType QoSNone = 0;
const QoSType QoSFlush = 1;
const QoSType QoSReliable = 2;
```

DsLogAdmin Datatypes

DsLogAdmin::LogId Type

```
typedef unsigned long LogId;
```

Specifies a log's unique id. The id is used by several methods for specifying which log to use or to locate a specific log.

DsLogAdmin::RecordId Type

```
typedef unsigned long long RecordId;
```

Specifies a record's id. A record's id is unique within the log storing it.

DsLogAdmin::RecordIdList Sequence

```
typedef sequence<RecordId> RecordIdList;
```

Specifies a list of record ids. The list does not need to be in any particular order.

DsLogAdmin::Constraint Type

```
typedef string Constraint;
```

Specifies the constraints used for querying a log's records.

DsLogAdmin::TimeT Type

```
typedef TimeBase::TimeT TimeT;
```

Used to record logging times and for setting a log's duration.

DsLogAdmin::NVPair Structure

```
struct NVPair
  {
  string name;
  any    value;
  };
```

Specifies a name/value pair used to construct attributes for records.

Members

`name`	The name of the attribute. The value can be any string.
`value`	An any containing the setting for the attribute.

DsLogAdmin::NVList Sequence

```
typedef sequence<NVPair> NVList;
```

A list of name/value record attributes.

DsLogAdmin::TimeInterval Structure

```
struct TimeInterval
{
    TimeT start;
    TimeT stop;
};
```

Specifies the start and stop times for a logging session.

Members

`start`	The start time for the current logging session.
`stop`	The end time for the current logging session.

DsLogAdmin::LogRecord Structure

```
struct LogRecord
{
    RecordId id;
    TimeT    time;
    NVList   attr_list;
    any      info;
};
```

The data stored when a new record is logged.

Members

`id`	The unique identifier for the record
`time`	The time at which the record was logged.
`attr_list`	An optional list of attributes specified by the client
`info`	The data contained in the record.

DsLogAdmin::RecordList Sequence

```
typedef sequence<LogRecord> RecordList;
```

A list of records.

DsLogAdmin::Anys Sequence

```
typedef sequence<any> Anys;
```

A sequence of data stored in individual any packages.

DsLogAdmin::AvailabilityStatus Structure

```
struct AvailabilityStatus
{
    boolean off_duty;
    boolean log_full;
};
```

Represents the availability of a log.

Members

`off_duty`	`true` means the log is not scheduled to accept new events. `false` means it is schedualed to recieve new events.
`log_full`	If the log is full this member will be `true`.

DsLogAdmin::LogFullActionType Type

```
typedef unsigned short LogFullActionType;
```

Specifies a log's `full_action`. It can either be `halt` or `wrap`.

DsLogAdmin::Time24 Structure

```
struct Time24
{
    unsigned short hour;   // 0-23
    unsigned short minute; // 0-59
};
```

Specifies the fine grained times for a log's schedule

Members

hour	An hour specified in 24 hour format
minute	The minute within an hour. Can be a value from 0-59.

DsLogAdmin::Time24Interval Structure

```
struct Time24Interval
{
    Time24 start;
    Time24 stop;
};
```

A fine grained interval during which a log is scheduled to log new records.

Members

start	The time at which a log will begin logging new records.
stop	The time at which a log will stop logging new records.

DsLogAdmin::IntervalsOfDay Sequence

```
typedef sequence<Time24Interval> IntervalsOfDay;
```

A list of fine grained logging intervals.

DsLogAdmin::DaysOfWeek Type

```
typedef unsigned short DaysOfWeek;
```

A bit mask specifying the days of the week a fine grained logging interval is valid. It is constructed using the scheduling constants listed in “Scheduling Constants” on page 1321.

DsLogAdmin::WeekMaskItem Structure

```
struct WeekMaskItem
{
```

```
    DaysOfWeek     days;
    IntervalsOfDay intervals;
};
```

Specifies a fined grain log schedule.

Members

`days`	A bitmask specifying the days of the week for which the specified intervals are valid.
`intervals`	The fine grained logging intervals.

DsLogAdmin::WeekMask Sequence

```
typedef sequence<WeekMaskItem> WeekMask;
```

Specifies a log's fine grained logging schedule.

DsLogAdmin::Threshold Type

```
typedef unsigned short Threshold;
```

Specifies a threshold point, in terms of a percentage of how full a log is, at which to generate an alarm. Valid values are from `0-100`.

DsLogAdmin::CapacityAlarmThresholdList Sequence

```
typedef sequence<Threshold> CapacityAlarmThresholdList;
```

A list of thresholds at which alarms are generated.

DsLogAdmin::OperationalState Enum

```
enum OperationalState {disabled, enabled};
```

Specifies if a log is ready to log new records.

Table 34: *Log operational states*

Operational State	Reason
`enabled`	The log is healthy and its full functionality is available for use.
`disabled`	The log has encountered a runtime error and is unavailable. The log will not accept any new records and it may not be able to retrieve valid records. The log will still attempt to forward events if its `ForwardingState` is set to on.

DsLogAdmin::AdministrativeState Enum

```
enum AdministrativeState {locked, unlocked};
```

Specifies if a log can accept new records.

DsLogAdmin::ForwardingState Enum

```
enum ForwardingState {on, off}
```

Specifies if a log will forward events or not.

DsLogAdmin::LogList Sequence

```
typedef sequence<Log> LogList;
```

A sequence of log object references.

DsLogAdmin::LogIdList Sequence

```
typedef sequence<LogId> LogIdList;
```

A sequence of log ids.

DsLogAdmin::QoSType Type

```
typedef unsigned short QoSType;
```

Specifies the log's QoS level. Valid values are `QoSNone`, `QoSFlush`, and `QoSReliable`.

DsLogAdmin::QoSList Sequence

```
typedef sequence<QoSType> QoSList;
```

A list of `QoSType`.

DsLogAdmin::Iterator Interface

The `Iterator` interface provides the methods for accessing records returned by the iterator when querying a log. It also provides the method used to release the resources consumed by the returned iterator.

```
interface Iterator
{
  RecordList get(in unsigned long position,
                 in unsigned long how_many)
  raises(InvalidParam);

  void destroy();
};
```

Iterator::get()

```
  RecordList get(in unsigned long position,
                 in unsigned long how_many)
  raises(InvalidParam);
```

Retrieves the specified number of records from the iterator object and returns them as a `RecordList`.

Parameters

`position`	The number of the record from which to start retrieving records.
`how_many`	The number of records to return.

Exceptions

`InvalidParam`	Raised if the position is negative or past the end of the list.

Iterator::destroy()

```
void destroy();
```

Releases the resources used by the iterator object. If an iterator object is returned, you must explicitly destroy it.

DsLogAdmin::Log Interface

The Log interface provides all of the basic functionality for log objects. All other log interfaces inherit from this interface. The Log interface provides the methods for managing a log's functional properties including its full_action and maximum size. It also defines the methods for querying the log for records, retrieving records from the log, and deleting records from the log. In addition, it defines the flush() method and two methods for copying logs.

```
interface Log
{
  LogMgr my_factory();
  LogId id();

  unsigned long get_max_record_life();
  void set_max_record_life(in unsigned long life);

  unsigned long long get_max_size();
  void set_max_size(in unsigned long long size)
    raises (InvalidParam);
  unsigned long long get_current_size();
  unsigned long long get_n_records();

  LogFullActionType get_log_full_action();
  void set_log_full_action(in LogFullActionType action)
    raises(InvalidLogFullAction);

  AdministrativeState get_administrative_state();
  void set_administrative_state(in AdministrativeState state);

  ForwardingState get_forwarding_state();
  void set_forwarding_state(in ForwardingState state);

  OperationalState get_operational_state();
  AvailabilityStatus get_availability_status();

  TimeInterval get_interval();
  void set_interval(in TimeInterval interval)
    raises (InvalidTime, InvalidTimeInterval);

  CapacityAlarmThresholdList get_capacity_alarm_thresholds();
```

```
void set_capacity_alarm_thresholds(in CapacityAlarmThresholdList
 threshs)
 raises (InvalidThreshold);

WeekMask get_week_mask();
void set_week_mask(in WeekMask masks)
  raises (InvalidTime, InvalidTimeInterval, InvalidMask);

QoSList get_log_qos();
void set_log_qos(in QoSList qos) raises (UnsupportedQoS)

RecordList query(in string grammar, in Constraint c,
                 out Iterator i)
  raises(InvalidGrammar, InvalidConstraint);

RecordList retrieve(in TimeT from_time, in long how_many,
                    out Iterator i);

unsigned long match(in  string grammar, in  Constraint c)
  raises(InvalidGrammar, InvalidConstraint);

unsigned long delete_records(in string grammar, in Constraint c)
  raises(InvalidGrammar, InvalidConstraint);
unsigned long delete_records_by_id(in RecordIdList ids);

void write_records(in Anys records)
  raises(LogFull, LogOffDuty, LogLocked, LogDisabled);
void write_recordlist(in RecordList list)
  raises(LogFull, LogOffDuty, LogLocked, LogDisabled);

void set_record_attribute(in RecordId id, in NVList attr_list)
  raises(InvalidRecordId, InvalidAttribute);
unsigned long set_records_attribute(in string grammar,
                                    in Constraint c,
                                    in NVList attr_list)
  raises(InvalidGrammar, InvalidConstraint, InvalidAttribute);

NVList get_record_attribute(in RecordId id)
  raises(InvalidRecordId);

Log copy(out LogId id);
Log copy_with_id(in LogId id) raises(LogIdAlreadyExists);
```

```
  void flush() raises(UnsupportedQoS);
};
```

Log::my_factory()

```
LogMgr my_factory();
```

Returns an object reference to the log object's log factory.

Log::id()

```
LogId id();
```

Returns the id of the log.

Log::get_max_record_life()

```
unsigned long get_max_record_life();
```

Returns the maximum amount of time, in seconds, that a record stays valid in the log.

Log::set_max_record_life()

```
void set_max_record_life(in unsigned long life);
```

Sets the maximum amount of time, in seconds, that a record stays valid in the log. After a record has become stale, it will automatically be removed from the log.

Parameters

`life`	The number of seconds for which records will remain valid. Zero specifies an infinite life span.

Log::get_max_size()

```
unsigned long long get_max_size();
```

Returns the maximum size, in bytes, of the log.

Log::set_max_size()

```
void set_max_size(in unsigned long long size)
  raises(InvalidParam);
```

Set the maximum size, in bytes, of the log.

Parameters

`size`	The maximum size of the log object in bytes.

Exceptions

`InvalidParam`	The size specified is smaller than the current size of the log.

Log::get_current_size()

```
unsigned long long get_current_size();
```

Returns the current size of the log in octets.

Log::get_n_records()

```
unsigned long long get_n_records();
```

Returns the current number of records in the log.

Log::get_log_full_action()

`LogFullActionType` `get_log_full_action();`

Returns the log's `full_action` setting.

Log::set_log_full_action()

```
void set_log_full_action(in LogFullActionType action)
raises(InvalidLogFullAction);
```

Sets the log's `full_action`.

Parameters

`action`	The log's `full_action`. Valid values are `wrap` and `halt`.

Exceptions

`InvalidLogFullAction`	The `full_action` specified is not a supported.

Log::get_administrative_state()

```
AdministrativeState get_administrative_state();
```

Returns the log's administrative state.

Log::set_administrative_state()

```
void set_administrative_state(in AdministrativeState state);
```

Sets the log's administrative state.

Parameters

`state`	The new administrative state for the log. Valid states are `locked` and `unlocked`.

Log::get_forwarding_state()

```
ForwardingState get_forwarding_state();
```

Returns the log's forwarding state. If the log's forwarding state is on, the log will forward events.

Log::set_forwarding_state()

```
void set_forwarding_state(in ForwardingState state);
```

Changes the log's forwarding state.

Parameters

`state`	The new forwarding state. The valid values are: **on** specifies that the log will forward events. **off** specifies that the log will not forward events.

Log::get_operational_state()

```
OperationalState get_operational_state();
```

Returns the log's operational state. The log can either be `enabled` or `disabled`.

Log::get_interval()

```
TimeInterval get_interval();
```

Returns the log's coarse grained logging interval.

Log::set_interval()

```
void set_interval(in TimeInterval interval)
raises (InvalidTime, InvalidTimeInterval);
```

Changes the log's coarse grained logging interval.

Parameters

`interval`	The log's new coarse grained logging interval. Zero sets the log to an infinite duration.

Exceptions

InvalidTime	One of the times specified is not a legal time.
InvalidTimeInterval	The start time of the interval is after the stop time. Also, the stop time is prior to the current time.

Log::get_availability_status()

```
AvailabilityStatus get_availability_status();
```

Returns the log's availability. The log can be on duty, off duty, full, or both off duty and full.

Log::get_capacity_alarm_thresholds()

```
CapacityAlarmThresholdList get_capacity_alarm_thresholds();
```

Returns a list of the log's alarm thresholds.

Log::set_capacity_alarm_thresholds()

```
void set_capacity_alarm_thresholds(in CapacityAlarmThresholdList
threshs)
raises (InvalidThreshold);
```

Sets threshold alarms in the log.

Parameters

threshs	A sequence of Threshold specifying at what points threshold alarm events are to be generated.

Exceptions

InvalidThreshold	Raised if one of the thresholds is not in the valid range.

Log::get_week_mask()

```
WeekMask get_week_mask();
```

Returns the log's weekly schedule.

Log::set_week_mask()

```
void set_week_mask(in WeekMask masks)
raises (InvalidTime, InvalidTimeInterval, InvalidMask);
```

Changes the log's weekly schedule.

Parameters

masks	The new schedule to set on the log.

Exceptions

InvalidTime	One of the times set on the log is not a valid time.
InvalidTimeInterval	One of the stop times specified is before its associated start time. Also, one of the time intervals overlaps another time interval.
InvalidMask	The WeekMask is malformed.

Log::get_log_qos()

```
QoSList get_log_qos();
```

Returns the log's QoS settings.

Log::set_log_qos()

```
void set_log_qos(in QoSList qos) raises (UnsupportedQoS);
```

Sets the log's QoS type. Valid settings are `QoSNone`, `QoSFlush`, and `QosReliable`.

Parameters

qos	The QoS properties to set on the log.

Exceptions

UnsupportedQoS	One of the QoS properties specified for the log is invalid. The invalid setting is returned.

Log::query()

```
RecordList query(in string grammar, in Constraint c, out Iterator i)
raises(InvalidGrammar, InvalidConstraint);
```

Retreives records from the log based on a constraint.

Parameters

grammar	The grammar used to consruct the contraint. The telecom logging service support the EXTENDED_TCL grammar
c	The contraint string against which records are matched.
i	Used when a large number of records are retreived. If it not used it will be nil.

Exceptions

InvalidGrammar	The telecom logging service does not support the specified grammar.
InvalidConstraint	The constraint does not conform to the specified grammar.

Log::retrieve()

```
RecordList retrieve(in TimeT from_time, in long how_many,
                    out Iterator i);
```

Returns the specified number of records starting at the specified time. If the number of records is larger than can be stored in the return parameter, the remaining records are accessible through the Iterator.

Parameters

`from_time`	The time at which the first record to retrieve was logged.
`how_many`	The number of records to retrieve. A negative value causes the method to retireve records prior to the specified time.
`i`	The `Iterator` object reference.

Log::match()

```
unsigned long match(in string grammar, in Constraint c)
raises(InvalidGrammar, InvalidConstraint);
```

Returns the number of records that match the specified constraint.

Parameters

`grammar`	The grammar used to specify the constraint. The telecom logging service supports the `EXTENDED_TCL` grammar.
`c`	The constraint string.

Exceptions

`InvalidGrammar`	The telecom logging service does not support the specified grammar.
`InvalidConstraint`	The constraint does not conform to the specified grammar.

Log::delete_records()

```
unsigned long delete_records(in string grammar, in Constraint c)
raises(InvalidGrammar, InvalidConstraint);
```

Deletes all of the records that match the specified constraint and returns the number of records deleted.

Parameters

`grammar`	The grammar used to specify the constraint. The telecom logging service supports the `EXTENDED_TCL` grammar.
`c`	The constraint string.

Exceptions

`InvalidGrammar`	The telecom logging service does not support the specified grammar.
`InvalidConstraint`	The constraint does not conform to the specified grammar.

Log::delete_records_by_id()

```
unsigned long delete_records_by_id(in RecordIdList ids);
```

Deletes the specified records and returns the number of deleted records.

Parameters

`ids`	A sequence of record ids specifying the records to delete.

Log::write_records()

```
void write_records(in Anys records)
raises(LogFull, LogOffDuty, LogLocked, LogDisabled);
```

Writes a series of records to a log. The you cannot specifiy any optional attributes and cannot discover the records id.

Parameters

`records`	A sequence of any that contains the data for a group of records.

Exceptions

`LogFull`	The log is full and its `full_action` is set to `halt`.
`LogOffDuty`	The log is not currently scheduled to accept new records.

LogLocked	The log's administrative state is set to not accept new records.
LogDisabled	The log has encountered a processing error and is unable to accept new records.

Log::write_recordlist()

```
void write_recordlist(in RecordList list)
raises(LogFull, LogOffDuty, LogLocked, LogDisabled);
```

Writes a series of records to the log. You can construct records that include an optional attribute list and each record in the list will be updated to include the time it was logged and its record id.

Parameters

list	A sequence of LogRecord that contains the data for a group of records.

Exceptions

LogFull	The log is full and its full_action is set to halt.
LogOffDuty	The log is not currently scheduled to accept new records.
LogLocked	The log's administrative state is set to not accept new records.
LogDisabled	The log has encountered a processing error and is unable to accept new records.

Log::set_record_attribute()

```
void set_record_attribute(in RecordId id, in NVList attr_list)
raises(InvalidRecordId, InvalidAttribute);
```

Sets attributes for a single record which is specified by its record id.

Parameters

id	The id of the record on which you wish to set attributes.
attr_list	The list of attributes that you want to set on the record.

Exceptions

InvalidRecordId	The record specified dose not exist.
InvalidAttribute	One of the attributes is illegal.

Log::set_records_attribute()

```
unsigned long set_records_attribute(in string     grammar,
                                    in Constraint c,
                                    in NVList     attr_list)
raises(InvalidGrammar, InvalidConstraint, InvalidAttribute);
```

Sets attributes for all records that match the constraint. It returns the numbers of records whose attributes were changed.

Parameters

grammar	The grammar used to specify the constraint. The telecom logging service supports the EXTENDED_TCL grammar.
c	The constraint string.
attr_list	The list of attributes that you want to set on the record.

Exceptions

InvalidGrammar	The telecom logging service does not support the specified grammar.
InvalidConstraint	The constraint does not conform to the specified grammar.
InvalidAttribute	One of the attributes is illegal.

Log::get_record_attribute()

```
NVList get_record_attribute(in RecordId id)
raises(InvalidRecordId);
```

Returns the list of attributes that are set on the specified record.

Parameters

`id`	The id of the record whose attributes you want to retrieve.

Exceptions

`InvalidRecordId`	The record specified does not exist.

Log::copy()

```
Log copy(out LogId id);
```

Copies the log object and returns a reference to the new log object.

Parameters

`id`	The id assigned to the newly created log.

Log::copy_with_id()

```
Log copy_with_id(in LogId id)
raises (LogIdAlreadyExists);
```

Copies the log and returns a reference to the newly created log. This method allows you to specifiy the logs id.

Parameters

`id`	The new log's id.

Exceptions

`LogIdAlreadyExists`	The user assigned id is already in use.

Log::flush()

```
void flush()
raises(UnsupportedQoS);
```

Cuases the log to flush its memory buffer to its associated permanent store.

Exceptions

`UnsupportedQoS`	The log does not support `QoSFlush`.

DsLogAdmin::BasicLog Interface

The `BasicLog` interface extend the `Log` interface to support the loggging by event-unaware CORBA objects. It defines only one method, `destroy()`, which is used to destroy a `BasicLog` object.

```
interface BasicLog : Log
{
  void destroy();
};
```

DsLogAdmin::LogMgr Interface

The LogMgr interface is inherited by all the log factory interfaces. It defines three methods of discovering deployed log objects.

```
interface LogMgr
{
  LogList list_logs();
  Log find_log(in LogId id);
  LogIdList list_logs_by_id();
};
```

LogMgr::list_logs()

```
LogList list_logs();
```

Returns a list of object references, one for each log object associated with the factroy.

LogMgr::find_log()

```
Log find_log(in LogId id);
```

Returns an object reference to the specified log. If the log does not exist, it returns a `nil` reference.

LogMgr::list_logs_by_id()

```
LogIdList list_logs_by_id();
```

Returns a list containing the ids of all logs associated with the factory.

DsLogAdmin::BasicLogFactory Interface

The BasicLogFactory interface provides the functionality to instantiate a BasicLog object.

```
interface BasicLogFactory : LogMgr
{
  BasicLog create(in LogFullActionType full_action,
                  in unsigned long long max_size,
                  out LogId id)
    raises (InvalidLogFullAction);

  BasicLog create_with_id(in LogId id,
                          in LogFullActionType full_action,
                          in unsigned long long max_size)
    raises (LogIdAlreadyExists, InvalidLogFullAction);
};
```

BasicLogFactory::create()

```
BasicLog create(in LogFullActionType full_action,
                in unsigned long long max_size,
                out LogId id);
    raises (InvalidLogFullAction);
```

Returns an instantiated BasicLog object. The LogId returned is assigned by the service and can be used to access the returned BasicLog object.

Parameters

full_action	Specifies what the log object will do when it fills up.
max_size	Specifies the maximum amount of data, in bytes, the log can hold.
id	The LogId assigned to the BasicLog object by the service.

Exceptions

InvalidLogFullAction	The specified full_action is not a valid LogFullActionType.

BasicLogFactory::create_with_id()

```
BasicLog create_with_id(in LogId id,
                        in LogFullActionType full_action,
                        in unsigned long long max_size)
    raises (LogIdAlreadyExists, InvalidLogFullAction);
```

Returns an instantiated BasicLog object with a user supplied id.

Parameters

id	Specifies the LogId to assign the BasicLog.
full_action	Specifies what the log object will do when it fills up.
max_size	Specifies the maximum amount of data, in bytes, the log can hold.

Exceptions

InvalidLogFullAction	The specified full_action is not a valid LogFullActionType.
LogIdAlreadyExists	A log with the specified id already exists.

DsEventLogAdmin Module

The `DsEventLogAdmin` module defines the `EventLog` interface which provides logging capabilities for event service clients. This module also defines the `EventLogFactory` interface which is used to instantiate `EventLog` objects.

DsEventLogAdmin::EventLog Interface

The EventLog interface extends the functionality of the Log interface by also inheriting from CosEventChannelAdmin::EventChannel. This inheritence provides EventLog objects the ability to log events as they are passed through an event channel. The EventLog interface does not define any operations.

```
interface EventLog : DsLogAdmin::Log,
                     CosEventChannelAdmin::EventChannel
{
};
```

DsEventLogAdmin::EventLogFactory Interface

The EventLogFactory interface defines two operations for instatiating EventLog objects.

```
interface EventLogFactory : DsLogAdmin::LogMgr,
                            CosEventChannelAdmin::ConsumerAdmin
{
  EventLog create(
              in DsLogAdmin::LogFullActionType full_action,
              in unsigned long long max_size,
             in DsLogAdmin::CapacityAlarmThresholdList thresholds,
              out DsLogAdmin::LogId id)
  raises(DsLogAdmin::InvalidLogFullAction,
         DsLogAdmin::InvalidThreshold);

  EventLog create_with_id(
              in DsLogAdmin::LogId id,
              in DsLogAdmin::LogFullActionType full_action,
              in unsigned long long max_size,
             in DsLogAdmin::CapacityAlarmThresholdList thresholds)
  raises(DsLogAdmin::LogIdAlreadyExists,
         DsLogAdmin::InvalidLogFullAction,
         DsLogAdmin::InvalidThreshold);
};
```

EventLogFactory::create()

```
EventLog create(in LogFullActionType full_action,
              in unsigned long long max_size,
             in DsLogAdmin::CapacityAlarmThresholdList thresholds,
              out LogId id);
    raises (InvalidLogFullAction
            InvalidThreshold);
```

Returns an instantiated EventLog object. The LogId returned is assigned by the service and can be used to access the returned EventLog object.

Parameters

`full_action`	Specifies what the log object will do when it fills up.
`max_size`	Specifies the maximum amount of data, in bytes, the log can hold.
`thresholds`	Specifies , as a percentage of max log size, the points at which an `ThresholdAlarm` event will be generated.
`id`	The `LogId` assigned to the `EventLog` object by the service.

Exceptions

`InvalidLogFullAction`	The specified `full_action` is not a valid `LogFullActionType`.
`InvalidThreshold`	One of the thresholds specified is invalid.

EventLogFactory::create_with_id()

```
EventLog create_with_id(in LogId id,
                        in LogFullActionType full_action,
                        in unsigned long long max_size)
                       in DsLogAdmin::CapacityAlarmThresholdList thresholds)
  raises(DsLogAdmin::LogIdAlreadyExists,
         DsLogAdmin::InvalidLogFullAction,
         DsLogAdmin::InvalidThreshold);
```

Returns an instantiated `EventLog` object with a user supplied id.

Parameters

`id`	Specifies the `LogId` to assign the `EventLog`.
`full_action`	Specifies what the log object will do when it fills up.
`max_size`	Specifies the maximum amount of data, in bytes, the log can hold.
`thresholds`	Specifies , as a percentage of max log size, the points at which an `ThresholdAlarm` event will be generated.

Exceptions

LogIdAlreadyExists	A log with the specified id already exists.
InvalidLogFullAction	The specified full_action is not a valid LogFullActionType.
InvalidThreshold	One of the thresholds specified is invalid.

DsLogNotification Module

The `DsLogNotification` module defines the data types used to transmit log generated events to logging clients.

```
module DsLogNotification
{
typedef DsLogAdmin::Log       Log;
typedef DsLogAdmin::LogId     LogId;
typedef DsLogAdmin::Threshold Threshold;
typedef TimeBase::TimeT       TimeT;

typedef unsigned short PerceivedSeverityType;
const PerceivedSeverityType critical = 0;
const PerceivedSeverityType minor = 1;
const PerceivedSeverityType cleared = 2;
struct ThresholdAlarm
{
    Log       logref;
    LogId     id;
    TimeT     time;
    Threshold crossed_value;
    Threshold observed_value;
    PerceivedSeverityType perceived_severity;
};

struct ObjectCreation
{
    LogId id;
    TimeT time;
};

struct ObjectDeletion
{
    LogId id;
    TimeT time;
};

typedef unsigned short AttributeType;
```

```
const AttributeType capacityAlarmThreshold = 0;
const AttributeType logFullAction          = 1;
const AttributeType maxLogSize             = 2;
const AttributeType startTime              = 3;
const AttributeType stopTime               = 4;
const AttributeType weekMask               = 5;
const AttributeType filter                 = 6;
const AttributeType maxRecordLife          = 7;
const AttributeType qualityOfService       = 8;
struct AttributeValueChange
{
    Log   logref;
    LogId id;
    TimeT time;
    AttributeType type;
    any old_value;
    any new_value;
};

typedef unsigned short StateType;
const StateType administrativeState = 0;
const StateType operationalState    = 1;
const StateType forwardingState     = 2;
struct StateChange
{
    Log   logref;
    LogId id;
    TimeT time;
    StateType type;
    any new_value;
};

struct ProcessingErrorAlarm
{
    long error_num;
    string error_string;
};

};
```

DsLogNotification::PerceivedSeverityType Type

```
typedef unsigned short PerceivedSeverityType;
const PerceivedSeverityType critical = 0;
const PerceivedSeverityType minor = 1;
const PerceivedSeverityType cleared = 2;
```

Defines the severity of a threshold alarm. A threshold alarm's severity is considered `minor` unless the log is full.

DsLogNotification::ThresholdAlarm Structure

```
struct ThresholdAlarm
{
    Log       logref;
    LogId     id;
    TimeT     time;
    Threshold crossed_value;
    Threshold observed_value;
    PerceivedSeverityType perceived_severity;
};
```

The data type passed in a threshold alarm event.

Members

`logref`	An object reference to the log object which caused the event.
`id`	The id of the log object which caused the event.
`time`	The time the event was generated.
`crossed_value`	The capacity threshold which was passed to trigger the event.
`observed_value`	The actual percentage of the log that is full.
`perceived_severity`	The severity of the alarm. If the severity is critical then the log object is full.

DsLogNotification::ObjectCreation Structure

```
struct ObjectCreation
{
    LogId id;
    TimeT time;
};
```

The data type passed in an object creation event.

Members

`id`	The id of the newly created log object.
`time`	The time the log object was generated.

DsLogNotification::ObjectDeletion Structure

```
struct ObjectDeletion
{
    LogId id;
    TimeT time;
};
```

The data type passed in an object deletion event.

Members

`id`	The id of the deleted log object.
`time`	The time the log object was deleted.

DsLogNotification::AttributeType Type

```
typedef unsigned short AttributeType;
const AttributeType capacityAlarmThreshold = 0;
const AttributeType logFullAction          = 1;
const AttributeType maxLogSize             = 2;
const AttributeType startTime              = 3;
const AttributeType stopTime               = 4;
const AttributeType weekMask               = 5;
const AttributeType filter                 = 6;
```

```
const AttributeType maxRecordLife          = 7;
const AttributeType qualityOfService       = 8;
```

The data type and constants used to represent the type of attribute changed in an attribute change event.

DsLogNotification::AttributeValueChange Structure

```
struct AttributeValueChange

{
    Log logref;
    LogId id;
    TimeT time;
    AttributeType type;
    any old_value;
    any new_value;
};
```

Members

`logref`	An object reference to the log object which caused the event.
`id`	The id of the log object which caused the event.
`time`	The time the event was generated.
`type`	The attribute that was changed.
`old_value`	The previous value of the attribute.
`new_value`	The attribute's new value.

DsLogNotification::StateType Type

```
typedef unsigned short StateType;
const StateType administrativeState = 0;
const StateType operationalState    = 1;
const StateType forwardingState     = 2;
```

The data type and constants used to represent which type of state was changed in a state change event.

DsLogNotification::StateChange Structure

```
struct StateChange
{
    Log   logref;
    LogId id;
    TimeT time;
    StateType type;
    any new_value;
};
```

The data type passed in a state change event.

Members

`logref`	An object reference to the log object which caused the event.
`id`	The id of the log object which caused the event.
`time`	The time the event was generated.
`type`	The type of state that was changed.
`new_value`	The new state.

DsLogNotification::ProcessingErrorAlarm Structure

```
struct ProcessingErrorAlarm
{
    long error_num;
    string error_string;
};
```

The data type passed when a processing error event occurs.

Members

`error_num`	The error number.
`error_string`	A string explaining the error.

DsNotifyLogAdmin Module

The `DsNotifyLogAdmin` module extends the functionality of the interfaces specified in the `DsLogAdmin` module to support notification style push and pull communication and forwarding of structured and sequenced events. The extended functionality also includes notification style event filtering and subscription/publication functionality.

DsNotifyLogAdmin::NotifyLog Interface

The NotifyLog interface extends the functionality of the Log interface to support notification style filters. It inherits from the EventChannel interface of module CosNotifyChannelAdmin.

```
interface NotifyLog : DsEventLogAdmin::EventLog,
                      CosNotifyChannelAdmin::EventChannel
{
  CosNotifyFilter::Filter get_filter();
  void set_filter(in CosNotifyFilter::Filter filter);
};
```

NotifyLog::get_filter()

```
CosNotifyFilter::Filter get_filter();
```

Returns a reference to the filter object associated with the log.

NotifyLog::set_filter()

```
void set_filter(in CosNotifyFilter::Filter filter);
```

Associates a filter with the log. The filter will determine which events will be logged.

Parameters

`filter`	The filter you want to set on the log.

DsNotifyLogAdmin::NotifyLogFactory Interface

The NotifyLogFactory extends the functionality of the LogMgr interface to support the creation of NotifyLog objects. It also inherits from the CosNotifyChannelAdmin::ConsumerAdmin interface. This inheritance allows it to forward events to the clients of its associated NotifyLog objects.

```
interface NotifyLogFactory : DsLogAdmin::LogMgr,
                             CosNotifyChannelAdmin::ConsumerAdmin
{
  NotifyLog create(in DsLogAdmin::LogFullActionType full_action,
              in unsigned long long max_size,
             in DsLogAdmin::CapacityAlarmThresholdList thresholds,
              in CosNotification::QoSProperties initial_qos,
              in CosNotification::AdminProperties initial_admin,
              out DsLogAdmin::LogId id)
    raises(DsLogAdmin::InvalidLogFullAction,
           DsLogAdmin::InvalidThreshold,
           CosNotification::UnsupportedQoS,
           CosNotification::UnsupportedAdmin);

  NotifyLog create_with_id(in DsLogAdmin::LogId id,
              in DsLogAdmin::LogFullActionType full_action,
              in unsigned long long max_size,
             in DsLogAdmin::CapacityAlarmThresholdList thresholds,
              in CosNotification::QoSProperties initial_qos,
              in CosNotification::AdminProperties initial_admin)
    raises(DsLogAdmin::LogIdAlreadyExists,
           DsLogAdmin::InvalidLogFullAction,
           DsLogAdmin::InvalidThreshold,
           CosNotification::UnsupportedQoS,
           CosNotification::UnsupportedAdmin);
};
```

NotifyLog::create()

```
NotifyLog create(in DsLogAdmin::LogFullActionType full_action,
                 in unsigned long long max_size,
                in DsLogAdmin::CapacityAlarmThresholdList thresholds,
                 in CosNotification::QoSProperties initial_qos,
                 in CosNotification::AdminProperties initial_admin,
                 out DsLogAdmin::LogId id)
  raises(DsLogAdmin::InvalidLogFullAction,
         DsLogAdmin::InvalidThreshold,
         CosNotification::UnsupportedQoS,
         CosNotification::UnsupportedAdmin);
```

Creates a new NotifyLog object, assigns the new log a unique id, and returns a reference to the newly instantiated log object.

Parameters

`full_action`	The log's behavior when it reaches its maximum size. Valid values are `wrap` and `halt`.
`max_size`	The maximum size of the log in bytes.
`thresholds`	The thresholds when alarm events will be generated. Specified as a percentage of the log's size.
`initial_qos`	The initial notification style QoS properties to set on the log object's associated notification channel.
`initial_admin`	The initial administrative properties to set on the log object's associated notification channel.
`id`	Returns the log object's factory assigned id.

Exceptions

InvalidLogFullAction	The value for the log's `full_action` was not a valid `full_action`.
InvalidThreshold	One of the threshold alarm values was not within the valid range
UnsupportedQoS	One of the QoS properties is invalid or does not support the value you are trying to set for it.
UnsupportedAdmin	One of the administrative properties is invalid or does not support the value you are trying to set for it.

NotifyLog::create_with_id()

```
NotifyLog create_with_id(in DsLogAdmin::LogId id,
                         in DsLogAdmin::LogFullActionType full_action,
                         in unsigned long long max_size,
                        in DsLogAdmin::CapacityAlarmThresholdList thresholds,
                         in CosNotification::QoSProperties initial_qos,
                         in CosNotification::AdminProperties initial_admin)
  raises(DsLogAdmin::LogIdAlreadyExists,
         DsLogAdmin::InvalidLogFullAction,
         DsLogAdmin::InvalidThreshold,
         CosNotification::UnsupportedQoS,
         CosNotification::UnsupportedAdmin);
```

Creates a new NotifyLog object using a user assigned id and returns a reference to the newly instantiated log object.

Parameters

`id`	The log object's id.
`full_action`	The log's behavior when it reaches its maximum size. Valid values are `wrap` and `halt`.
`max_size`	The maximum size of the log in bytes.
`thresholds`	The thresholds when alarm events will be generated. Specified as a percentage of the log's size.
`initial_qos`	The initial notification style QoS properties to set on the log object's associated notification channel.
`initial_admin`	The initial administrative properties to set on the log object's associated notification channel.

Exceptions

LogIdAlreadyExists	A log already exists with the specified id.
InvalidLogFullAction	The value for the log's `full_action` was not a valid `full_action`.
InvalidThreshold	One of the threshold alarm values was not within the valid range
UnsupportedQoS	One of the QoS properties is invalid or does not support the value you are trying to set for it.

UnsupportedAdmin	One of the administrative properties is invalid or does not support the value you are trying to set for it.

IT_NotifyLogAdmin Module

This module extends the OMG specified NotifyLog and NotifyLogFactory interfaces to support event subscription and publication. Also provides access to a default filter factory.

```
module IT_NotifyLogAdmin
{
  interface NotifyLog :DsNotifyLogAdmin::NotifyLog
    {
    CosNotification::EventTypeSeq obtain_offered_types();
    CosNotification::EventTypeSeq obtain_subscribed_types();
    };

  interface NotifyLogFactory :DsNotifyLogAdmin::NotifyLogFactory
    {
    readonly attribute CosNotifyFilter::FilterFactory
   default_filter_factory;
    readonly attribute IT_LogAdmin::Manager manager;
    };
};
```

IT_NotifyLogAdmin::NotifyLog Interface

This interface provides IONA specific extensions to DsNotifyLogAdmin::NotifyLog to support notification style event publication and subscription.

```
interface NotifyLog :DsNotifyLogAdmin::NotifyLog
  {
   CosNotification::EventTypeSeq obtain_offered_types();
   CosNotification::EventTypeSeq obtain_subscribed_types();
  };
```

NotifyLog::obtain_offered_types()

```
CosNotification::EventTypeSeq obtain_offered_types();
```

Allows event consumers to ascertain what events are being advertised by event suppliers.

NotifyLog::obtain_subscribed_types()

```
CosNotification::EventTypeSeq obtain_subscribed_types();
```

Allows event suppliers to ascertain which events the event consumers in the channel are interested in receiving.

IT_NotifyLogAdmin::NotifyLogFactory Interface

Extends DsNotifyLogAdmin::NotifyLogFactory to include a link to the notification channel's default filter factory and a link to the telecom logging service's manager.

```
interface NotifyLogFactory :DsNotifyLogAdmin::NotifyLogFactory
  {
  readonly attribute CosNotifyFilter::FilterFactory
   default_filter_factory;
  readonly attribute IT_LogAdmin::Manager manager;
  };
```

NotifyLogFactory::default_filter_factory Attribute

```
readonly attribute CosNotifyFilter::FilterFactory
default_filter_factory;
```

Provides a reference to the notification channel's default filter factory, which is used to create new filter objects for `NotifyLog` objects.

NotifyLogFactory::manager Attribute

```
readonly attribute IT_LogAdmin::Manager manager;
```

Provides a link to the telecom logging service's manager.

Trader Service

CosTrading Module

Contains the major functional interfaces of a trading service.

CosTrading Data Types

CosTrading::Constraint Data Type

```
typedef Istring Constraint;
```

A query constraint expression. The constraint is used to filter offers during a query, and must evaluate to a boolean expression.

The constraint language consists of the following elements:

- comparative functions: ==, !=, >, >=,
- boolean connectives: and, or, not
- property existence: exist
- property names
- numeric, boolean and string constants
- mathematical operators: +, -, *, /
- grouping operators: (,)

The following property value types can be manipulated using the constraint language:

- boolean, short, unsigned short, long, unsigned long, float, double, char, Ichar, string, Istring
- sequences of the above types

Only the `exist` operator can be used on properties of other types.

Notes

The constraint language keywords are case-sensitive

Literal strings should be enclosed in single quotes

The boolean literals are TRUE and FALSE

CosTrading::Istring Data Type

```
typedef string Istring;
```

When internationalized strings are widely supported, this definition will be changed.

CosTrading::LinkName Data Type

```
typedef Istring LinkName;
```

The name of a unidirectional link from one trader to another. The only restriction on the format of a link name is it cannot be an empty string.

CosTrading::LinkNameSeq Data Type

```
typedef sequence<LinkName> LinkNameSeq;
```

CosTrading::OfferId Data Type

```
typedef string OfferId;
```

An offer identifier is an opaque string whose format is determined entirely by the trading service from which the offer identifier was obtained, and can only be used with that trading service.

CosTrading::OfferIdSeq Data Type

```
typedef sequence<OfferId> OfferIdSeq;
```

CosTrading::OfferSeq Data Type

```
typedef sequence<Offer> OfferSeq;
```

CosTrading::PolicyName Data Type

```
typedef string PolicyName;
```

The name of a policy used to control the trader's behavior. The only restriction on the format of a policy name is it cannot be an empty string.

CosTrading::PolicyNameSeq Data Type

```
typedef sequence<PolicyName> PolicyNameSeq;
```

CosTrading::PolicySeq Data Type

```
typedef sequence<Policy> PolicySeq;
```

CosTrading::PolicyValue Data Type

```
typedef any PolicyValue;
```

CosTrading::PropertyName Data Type

```
typedef Istring PropertyName;
```

Although not explicitly defined in the specification, a property name should start with a letter, may contain digits and underscores, and should not contain spaces.

CosTrading::PropertyNameSeq DataType

```
typedef sequence<PropertyName> PropertyNameSeq;
```

CosTrading::PropertySeq Data Type

```
typedef sequence<Property> PropertySeq;
```

CosTrading::PropertyValue Data Type

```
typedef any PropertyValue;
```

A `CORBA::Any` containing the value of the property. Orbix Trader allows arbitrarily complex user-defined types to be used as property values.

CosTrading::ServiceTypeName Data Type

```
typedef Istring ServiceTypeName;
```

A service type name can have one of two formats, both representing formats that appear in the Interface Repository.

- **Scoped Name** - A scoped name has the form `::One::Two`. Other supported variations are `Three::Four` and `Five`.
- **Interface Repository Identifier** - An interface repository identifier has the form `IDL:[prefix/][module/]name:X.Y`. For example, `IDL:omg.org/CosTrading/Lookup:1.0` is a valid interface repository identifier, and you can use the same format for your service type names.

Note: Although a service type name can appear similar to names used in the interface repository, the trading service never uses servicetype names to look up information in the interface repository.

CosTrader::TraderName Data Type

```
typedef LinkNameSeq TraderName;
```

A `TraderName` represents a path from one trader to the desired trader by following a sequence of links. The `starting_trader` importer policy, if specified for a query operation, should contain a value of this type.

Cos:Trading::TypeRepository Data Type

```
typedef Object TypeRepository;
```

TypeRepository represents an object reference for a CosTradingRepos::ServiceTypeRepository object. You will need to narrow this reference before you can interact with the service type repository.

CosTrading::FollowOption Enum

```
enum FollowOption
{
    local_only,
    if_no_local,
    always
};
```

Determines the follow behavior for linked traders.

The member values are defined as follows:

local_only	The trader will not follow a link.
if_no_local	The trader will only follow a link if no offers were found locally.
always	The trader will always follow a link.

CosTrading::Offer Struct

```
struct Offer
{
    Object reference;
    PropertySeq properties;
};
```

The description of a service offer. The data members contains the following data:

reference	The object reference associated with this offer. Depending on the configuration of the server, this reference may be nil.
properties	A sequence of properties associated with this offer.

CosTrading::Policy Struct

```
struct Policy
{
    PolicyName name;
    PolicyValue value;
};
```

CosTrading::Property Struct

```
struct Property
{
    PropertyName name;
    PropertyValue value;
};
```

A name-value pair associated with a service offer or proxy offer. If the property name matches the name of a property in the offer's service type, then the TypeCode of the value must match the property definition in the service type.

Note: Orbix Trader allows properties to be associated with an offer even if the property name does not match any property in the service type. These properties can also be used in query constraint and preference expressions.

CosTrading Exceptions

CosTrading::DuplicatePolicyName

```
exception DuplicatePolicyName {PolicyName name};
```

More than one value was supplied for a policy. The policy name that caused the exception is returned.

CosTrading::DuplicatePropertyName

```
exception DuplicatePropertyName {PropertyName name};
```

The property name has already appeared once. The duplicated property name is returned.

CosTrading::IllegalConstraint

```
exception IllegalConstraint{Constraint constr};
```

An error occurred while parsing the constraint expression. The invalid constraint is passed back.

CosTrading::IllegalOfferId

```
exception IllegalOfferId {OfferId id};
```

The offer identifier is empty or malformed. The invalid id is returned.

CosTrading::IllegalPropertyName

```
exception IllegalPropertyName {PropertyName name};
```

The property name is empty or does not conform the format supported by the trader. The property name that caused the exception is returned.

CosTrading::IllegalServiceType

```
exception IllegalServiceType {ServiceTypeName type};
```

A service type name does not conform to the formats supported by the trader. The name that caused the exception is returned.

CosTrading::InvalidLookupRef

```
exception InvalidLookupRef {Lookup target};
```

The Lookup object reference cannot be `nil`.

CosTrading::MissingMandatoryProperty

```
exception MissingMandatoryProperty
{
    ServiceTypeName type;
    PropertyName name;
};
```

No value was supplied for a property defined as mandatory by the service type.

CosTrading::NotImplemented

```
exception NotImplemented {};
```

The requested operation is not supported by this trading service.

CosTrading::PropertyTypeMismatch

```
exception PropertyTypeMismatch
{
    ServiceTypeName type;
    Property prop;
};
```

The property value type conflicts with the property's definition in the service type.

CosTrading::ReadonlyDynamicProperty

```
exception ReadonlyDynamicProperty
{
    ServiceTypeName type;
    PropertyName name;
};
```

A property that is defined as read-only by the service type cannot have a dynamic value.

CosTrading::UnknownMaxLeft

```
exception UnknownMaxLeft {};
```

The iterator does not know how many items are left.

CosTrading::UnknownOfferId

```
exception UnknownOfferId {OfferId id};
```

The trader does not contain an offer with the given identifier. The unresolved ID is returned.

CosTrading::UnknownServiceType

```
exception UnknownServiceType {ServiceTypeName type};
```

The service type repository used by the trader does not have the requested service type. The unresolved name is returned.

CosTrading::Admin Interface

```
// IDL in CosTrading
interface Admin :
  TraderComponents, SupportAttributes,
  ImportAttributes, LinkAttributes
{
  typedef sequence OctetSeq;

  readonly attribute OctetSeq request_id_stem;

  unsigned long set_def_search_card (in unsigned long value);

  unsigned long set_max_search_card (in unsigned long value);

  unsigned long set_def_match_card (in unsigned long value);

  unsigned long set_max_match_card (in unsigned long value);

  unsigned long set_def_return_card (in unsigned long value);

  unsigned long set_max_return_card (in unsigned long value);

  unsigned long set_max_list (in unsigned long value);

  boolean set_supports_modifiable_properties (in boolean value);

  boolean set_supports_dynamic_properties (in boolean value);

  boolean set_supports_proxy_offers (in boolean value);

  unsigned long set_def_hop_count (in unsigned long value);

  unsigned long set_max_hop_count (in unsigned long value);

  FollowOption set_def_follow_policy (in FollowOption policy);

  FollowOption set_max_follow_policy (in FollowOption policy);
```

```
  FollowOption set_max_link_follow_policy (
               in FollowOption policy);

  TypeRepository set_type_repos (in TypeRepository repository);

  OctetSeq set_request_id_stem (in OctetSeq stem);

  void list_offers( in unsigned long how_many,
                    out OfferIdSeq ids,
                    out OfferIdIterator id_itr )
  raises ( NotImplemented );

  void list_proxies( in unsigned long how_many,
                     out OfferIdSeq ids,
                     out OfferIdIterator id_itr )
  raises ( NotImplemented );
};
```

Interface `Admin` provides attributes and operations for administrative control of the trading service.

Admin::request_id_stem Attribute

```
readonly attribute OctetSeq request_id_stem;
```

The request identifier “stem” is a sequence of octets that comprise the prefix for a request identifier. The trader will append additional octets to ensure the uniqueness of each request identifier it generates.

Admin::list_offers()

```
void list_offers(in unsigned long how_many,
                 out OfferIdSeq ids,
                 out OfferIdIterator id_itr)
raises(NotImplemented);
```

Obtains the identifiers for the service offers in this trader.

Parameters

how_namy	Indicates how many identifiers to return in ids.
ids	Contains at most how_many identifiers. If the number of identifiers exceeds how_many, the id_itr parameter will hold a reference to an iterator object through which the remaining identifiers can be obtained.
id_itr	Will hold nil if no identifiers were found or if all of the identifiers were returned in ids. Otherwise, holds a reference to an iterator object through which the remaining identifiers can be obtained.

Admin::list_proxies()

```
void list_proxies(in unsigned long how_many,
                  out OfferIdSeq ids,
                  out OfferIdIterator id_itr)
raises(NotImplemented);
```

Obtains the identifiers for the proxy offers in this trader.

Parameters

how_many	Indicates how many identifiers to return in ids.
ids	Contains at most how_many identifiers. If the number of identifiers exceeds how_many, the id_itr parameter will hold a reference to an iterator object through which the remaining identifiers can be obtained.
id_itr	Will hold nil if no identifiers were found or if all of the identifiers were returned in ids. Otherwise, holds a reference to an iterator object through which the remaining identifiers can be obtained.

Admin::set_def_follow_policy()

```
FollowOption set_def_follow_policy(in FollowOption policy);
```

Changes the value of the default link follow attribute and returns the previous value.

Parameters

`policy` The new value

Admin::set_def_hop_count()

```
unsigned long set_def_hop_count(in unsigned long value);
```

Changes the value of the default hop count attribute and returns the previous value.

Parameters

`value` The new value

Admin::set_def_match_card()

```
unsigned long set_def_match_card(in unsigned long value);
```

Changes the value of the default match cardinality attribute and returns the previous value.

Parameters

`value` The new value

Admin::set_def_return_card()

```
unsigned long set_def_return_card(in unsigned long value);
```

Changes the value of the default return cardinality attribute and returns the previous value.

Parameters

`value` The new value

Admin::set_def_search_card()

```
unsigned long set_def_search_card(in unsigned long value);
```

Changes the value of the default search cardinality attribute and returns the previous value.

Parameters

`value`	The new value

See Also `CosTrading::ImportAttributes`

Admin::set_max_follow_policy()

```
FollowOption set_max_follow_policy(in FollowOption policy);
```

Changes the value of the maximum link follow attribute and returns the previous value.

Parameters

`policy`	The new value

Admin::set_max_hop_count()

```
unsigned long set_max_hop_count(in unsigned long value);
```

Changes the value of the maximum hop count attribute and returns the previous value.

Parameters

`value`	The new value

Admin::set_max_link_follow_policy()

```
FollowOption set_max_link_follow_policy(in FollowOption policy);
```

Changes the value of the maximum link follow policy and returns the previous value.

Parameters

`policy`	The new value

Admin::set_max_list()

```
unsigned long set_max_list(in unsigned long value);
```

Changes the value of the maximum list attributes and returns the previous value.

Parameters

`value`	The new value

Admin::set_max_match_card()

```
unsigned long set_max_match_card(in unsigned long value);
```

Changes the value of the maximum match cardinality attribute and returns the previous value.

Parameters

`value`	The new value

Admin::set_max_return_card()

```
unsigned long set_max_return_card(in unsigned long value);
```

Changes the value of the maximum return cardinality attribute and returns the previous value.

Parameters

`value`	The new value

Admin::set_max_search_card()

```
unsigned long set_max_search_card(in unsigned long value);
```

Changes the value of the maximum search cardinality attribute and returns the previous value.

Parameters

`value`	The new value

Admin::set_request_id_stem()

```
OctetSeq set_request_id_stem(in OctetSeq stem);
```

Changes the value of the request identifier stem and returns the previous value.

Parameters

`stem`	The new value

Admin::set_supports_dynamic_properties()

```
boolean set_supports_dynamic_properties(in boolean value);
```

Establishes whether the trader considers offers with dynamic properties during a query and returns the previous setting.

Parameters

`value`	The new value

Admin::set_supports_modifiable_properties()

```
boolean set_supports_modifiable_properties(in boolean value);
```

Establishes whether the trader supports property modification and returns the previous setting.

Parameters

`value`
- TRUE activates property modification support.
- FALSE deactives property modification support.

Admin::set_supports_proxy_offers()

```
boolean set_supports_proxy_offers(in boolean value);
```

Establishes whether the trader supports proxy offers and returns the previous setting.

Parameters

`value`
- TRUE turns on proxy support.
- FALSE turns off proxy support.

Admin:set_type_repos()

```
TypeRepository set_type_repos(in TypeRepository repository);
```

Establishes the service type repository to be used by the trader and returns a reference to the previous type repository.

Parameters

`repository` A reference to a type repository.

CosTrading::SupportAttributes Interface

```
interface SupportAttributes
```

The read-only attributes in this interface determine what additional functionality a trader supports, and also provide access to the service type repository used by the trader.

SupportAttributes::supports_dynamic_properties Attribute

```
readonly attribute boolean supports_dynamic_properties;
```

If FALSE, offers with dynamic properties will not be considered during a query.

SupportAttributes::supports_modifiable_properties Attribute

```
readonly attribute boolean supports_modifiable_properties;
```

If FALSE, the `modify` operation of the `Register` interface will raise `NotImplemented`.

SupportAttributes::supports_proxy_offers Attribute

```
readonly attribute boolean supports_proxy_offers;
```

If FALSE, the `proxy_if` attribute of the `TraderComponents` interface will return `nil`, and proxy offers will not be considered during a query.

SupportAttributes::type_repos Attribute

```
readonly attribute TypeRepository type_repos;
```

Returns the object reference of the service type repository used by the trader.

CosTrading::Register Interface

```
interface Register
inherits from CosTrading::TraderComponents, CosTrading::
   SupportAttributes
```

Provides operations for managing service offers.

Register::OfferInfo Structure

```
struct OfferInfo
{
    Object reference;
    ServiceTypeName type;
    PropertySeq properties;
};
```

A complete description of a service offer.

`reference`	The object reference associated with this offer. Depending on the configuration of the server, this reference may be `nil`.
`type`	The service type for which this offer was exported
`properties`	A sequence of properties associated with this offer.

Register::IllegalTraderName Exception

```
exception IllegalTraderName
{
    TraderName name;
};
```

The trader name was empty, or a component of the name was not a valid link name.

Register::InterfaceTypeMismatch Exception

```
exception InterfaceTypeMismatch
{
    ServiceTypeName type;
    Object reference;
};
```

If the trader is configured to use the interface repository, then it will attempt to confirm that the interface of the object reference conforms to the interface of the service type. If the trader is able to determine that there is a mismatch, this exception is thrown.

Register::InvalidObjectRef Exception

```
exception InvalidObjectRef
{
    Object ref;
};
```

The object reference is `nil`, and the trader is is configured to reject offers with `nil` references.

Register::MandatoryProperty Exception

```
exception MandatoryProperty
{
    ServiceTypeName type;
    PropertyName name;
};
```

A mandatory property cannot be removed.

Register::NoMatchingOffers Exception

```
exception NoMatchingOffers
{
    Constraint constr;
};
```

No matching offers were found matching the constraint expression.

Register::ProxyOfferId Exception

```
exception ProxyOfferId
{
    OfferId id;
};
```

The offer identifier actually refers to a proxy offer.

Register::ReadonlyProperty Exception

```
exception ReadonlyProperty
{
    ServiceTypeName type;
    PropertyName name;
};
```

A read-only property cannot be modified.

Register::RegisterNotSupported Exception

```
exception RegisterNotSupported
{
    TraderName name;
};
```

The resolve operation is not supported by this trader.

Register::UnknownPropertyName Exception

```
exception UnknownPropertyName
{
    PropertyName name;
};
```

A property was identified for removal that does not exist in the offer.

Register::UnknownTraderName Exception

```
exception UnknownTraderName
{
    TraderName name;
};
```

The trader name could not be correctly resolved to a trader.

Register::describe()

```
OfferInfo describe(in OfferId id)
    raises(IllegalOfferId,
           UnknownOfferId,
           ProxyOfferId);
```

Obtains the description of a service offer and and returns it in an `OfferInfo` structure.

Parameters

`id`	Identifier of the offer of interest

Exceptions

`IllegalOfferId`	Offer identifier is empty or has an invalid format
`UnknownOfferId`	No offer was found with the given identifier
`ProxyOfferId`	Offer identifier refers to a proxy offer. Proxy offers must be described using the `Proxy` interface.

Register::export()

```
OfferId export(in Object reference,
               in ServiceTypeName type,
               in PropertySeq properties)
    raises(InvalidObjectRef,
           IllegalServiceType,
           UnknownServiceType,
           InterfaceTypeMismatch,
           IllegalPropertyName,
```

```
        PropertyTypeMismatch,
        ReadonlyDynamicProperty,
        MissingMandatoryProperty,
        DuplicatePropertyName);
```

Creates a new service offer and returns an identifer object for the new service. A client wishing to advertise a new offer is called an *exporter*.

Parameters

`reference`	Reference to an object that enables a client to interact with a remote server.
`type`	Identifies the service type for which this offer is advertised.
`properties`	List of named values that describe the service being offered.

Exceptions

`InvalidObjectRef`	Object reference is `nil` and the trader has been configured to reject `nil` references
`IllegalServiceType`	Service type name is empty or has an invalid format
`UnknownServiceType`	Service type was not found in service type repository
`InterfaceTypeMismatch`	Trader was able to determine that the interface of the object reference does not conform to the the interface of the service type
`IllegalPropertyName`	Property name is empty or has an invalid format
`PropertyTypeMismatch`	Property value type does not match the property definition of the service type
`ReadonlyDynamicProperty`	Read-only properties cannot have dynamic values
`MissingMandatoryProperty`	No value was supplied for a mandatory property
`DuplicatePropertyName`	Property name appeared more than once in list of properties

Register::modify()

```
void modify(in OfferId id,
            in PropertyNameSeq del_list,
            in PropertySeq modify_list)
    raises(NotImplemented,
           IllegalOfferId,
           UnknownOfferId,
           ProxyOfferId,
           IllegalPropertyName,
           UnknownPropertyName,
           PropertyTypeMismatch,
           ReadonlyDynamicProperty,
           MandatoryProperty,
           ReadonlyProperty,
           DuplicatePropertyName);
```

Modifies an existing service offer to add new properties, and change or delete existing properties.

Parameters

`id`	Identifier of the offer to be modified
`del_list`	Names of properties to be removed
`modify_list`	Properties to be added or modified

Exceptions

`NotImplemented`	Trader does not support modification of properties
`IllegalOfferId`	Offer identifier is empty or has an invalid format
`UnknownOfferId`	No offer was found with the given identifier
`ProxyOfferId`	Offer identifier refers to a proxy offer. Proxy offers must be described using the `Proxy` interface.
`IllegalPropertyName`	Property name is empty or has an invalid format
`UnknownPropertyName`	Property to be removed does not exist in offer
`PropertyTypeMismatch`	Property value type does not match the property definition of the service type
`ReadonlyDynamicProperty`	Read-only properties cannot have dynamic values

`MandatoryProperty`	Mandatory properties cannot be removed
`ReadonlyProperty`	Read-only properties cannot be modified
`DuplicatePropertyName`	Property name appeared more than once in list of properties

Register::resolve()

```
Register resolve(in TraderName name)
    raises(IllegalTraderName,
           UnknownTraderName,
           RegisterNotSupported);
```

Resolves a context-relative name for another trader and returns a `Register` object for the resolved trader.

Parameters

`name`	Identifies the trader to be resolved

Exceptions

`IllegalTraderName`	Trader name was empty, or a component of the name was not a valid link name
`UnknownTraderName`	Trader name could not be correctly resolved to a trader
`RegisterNotSupported`	Trader does not support this operation

Register::withdraw()

```
void withdraw(in OfferId id)
    raises(IllegalOfferId,
           UnknownOfferId,
           ProxyOfferId);
```

Removes a service offer.

Parameters

`id`	Identifier of the offer to be withdrawn

Exceptions

`IllegalOfferId`	Offer identifier is empty or has an invalid format
`UnknownOfferId`	No offer was found with the given identifier
`ProxyOfferId`	Offer identifier refers to a proxy offer. Proxy offers must be removed using the `Proxy` interface.

Register::withdraw_using_constraint()

```
void withdraw_using_constraint(in ServiceTypeName type,
                               in Constraint constr)
    raises(IllegalServiceType,
           UnknownServiceType,
           IllegalConstraint,
           NoMatchingOffers);
```

Withdraws all offers for a particular service type that match a constraint expression. Only offers that exactly match the given service type are considered. Proxy offers are not considered, and links are not followed.

Parameters

`type`	Identifies the service type for which offers are to be removed.
`constr`	Limits the search to only those offers for which this expression is true. The simplest constraint expression is `TRUE`, which matches any offer and is an efficient way to withdraw all offers for a service type.

Exceptions

`IllegalServiceType`	Service type name is empty or has an invalid format
`UnknownServiceType`	Service type was not found in service type repository
`IllegalConstraint`	An error occurred while parsing the constraint expression
`NoMatchingOffers`	No matching offers were found

CosTrading::Proxy Interface

```
interface Proxy :
          TraderComponents,
          SupportAttributes
{
  typedef Istring ConstraintRecipe;

  struct ProxyInfo
  {
    ServiceTypeName type;
    Lookup target;
    PropertySeq properties;
    boolean if_match_all;
    ConstraintRecipe recipe;
    PolicySeq policies_to_pass_on;
};

  exception IllegalRecipe {ConstraintRecipe recipe};
  exception NotProxyOfferId {OfferId id};

  OfferId export_proxy(in Lookup target, in ServiceTypeName type,
                       in PropertySeq properties,
                       in boolean if_match_all,
                       in ConstraintRecipe recipe,
                       in PolicySeq policies_to_pass_on)
  raises (IllegalServiceType, UnknownServiceType,
          InvalidLookupRef, IllegalPropertyName,
          PropertyTypeMismatch, ReadonlyDynamicProperty,
          MissingMandatoryProperty, IllegalRecipe,
          DuplicatePropertyName, DuplicatePolicyName);

  void withdraw_proxy( in OfferId id )
  raises (IllegalOfferId, UnknownOfferId, NotProxyOfferId);

  ProxyInfo describe_proxy( in OfferId id )
  raises (IllegalOfferId, UnknownOfferId, NotProxyOfferId);
};
```

Provides datatypes, exceptions and methods for managing proxy offers.

Proxy::ConstraintRecipe Data Type

```
typedef Istring ConstraintRecipe;
```

A constraint recipe specifies how the trader should rewrite a constraint before invoking the `query` operation of the proxy offer's Lookup interface. Using a constraint recipe, the exporter can have the trader rewrite a constraint into a completely different constraint language (one that is understood by the proxy offer's Lookup target).

The constraint recipe can include the value of properties using the expression "*$(property-name)*". The recipe can also include the entire text of the original constraint using the special syntax "$*".

For example, assume the property name has the value `"Joe"`, and the property age has the value 33. The constraint recipe `"Name == $(name) and Age"` would be rewritten as `"Name == 'Joe' and Age"`.

Proxy::ProxyInfo Data Structure

```
struct ProxyInfo
{
    ServiceTypeName type;
    Lookup target;
    PropertySeq properties;
    boolean if_match_all;
    ConstraintRecipe recipe;
    PolicySeq policies_to_pass_on;
};
```

A complete description of a proxy offer which contains the following members:

`type`	The service type for which tis offer was exported.
`target`	The target Lookup object.
`properties`	A sequence of properties associated with this offer.

`if_match_all`	If `TRUE`, type conformance is all that is necessary for this offer to match. If `FALSE`, the offer must also match the constraint expression.
`recipe`	The recipe for rewriting the constraint
`policies_to_pass_on`	Policies to be appended to the importer's policies and passed along to the target.

Proxy::IllegalRecipe Exception

```
exception IllegalRecipe{ConstraintRecipe recipe};
```

An error occurred while parsing the recipe.

Proxy::NotProxyOfferId Exception

```
exception NotProxyOfferId{OfferId id};
```

The offer identifier does not refer to a proxy offer.

Proxy::describe_proxy()

```
ProxyInfo describe_proxy(in OfferId id)
raises(IllegalOfferId,
       UnknownOfferId,
       NotProxyOfferId);
```

Obtains the description of a proxy offer.

Parameters

`id`	Identifier of the proxy offer of interest

Exceptions

`IllegalOfferId`	Offer Identifier is empty or has an invalid format.
`UnknownOfferId`	No offer was found with the given identifier
`NotProxyOfferId`	Offer identifier does not refer to a proxy offer

Proxy::export_proxy()

```
OfferId export_proxy(in Lookup target,
                     in ServiceTypeName type,
                     in PropertySeq properties,
                     in boolean if_match_all,
                     in ConstraintRecipe recipe,
                     in PolicySeq policies_to_pass_on)
raises(IllegalServiceType,
       UnknownServiceType,
       InvalidLookupRef,
       IllegalPropertyName,
       PropertyTypeMismatch,
       ReadonlyDynamicProperty,
       MissingMandatoryProperty,
       IllegalRecipe,
       DuplicatePropertyName,
       DuplicatePolicyName);
```

Creates a new proxy offer.

Parameters

`target`	The target `Lookup` interface
`type`	The service type for which this offer was exported
`properties`	A sequence of properties associated with this offer.
`if_match_all`	If `TRUE`, type conformance is all that is necessary for this offer to match. If `FALSE`, the offer must also match the constraint expression.
`recipe`	The recipe for rewriting the constraint.
`policies_to_pass_on`	Policies to be appended to teh importer's policies and passed along to the target.

Exceptions

`IllegalServiceType`	Service type name is empty or has invalid format.
`UnknownServiceType`	Service type was not found in the service type repository.
`InvalidLookupRef`	Target object reference is `nil`.

`IllegalPropertyName`	Property name is empty or has an invalid format.
`PropertyTypeMismatch`	Property value type does not match the property definition of the service type.
`ReadonlyDynamicProperty`	Read-only properties cannot have dynamic values.
`MissingMandatoryProperty`	No value was given for a mandatory property.
`IllegalRecipe`	An error occurred while parsing the constraint recipe.
`DuplicatePropertyname`	A property name appeared more than once in the list of properties.
`DuplicatePolicyName`	A policy name appeared more than once in the list of policies to pass on.

Proxy::withdraw_proxy()

```
void withdraw_proxy(in OfferId id)
    raises(IllegalOfferId,
           UnknownOfferId,
           NotProxyOfferId);
```

Removes a proxy offer.

Parameters

`id`	Identifier of the proxy offer to be withdrawn

Exceptions

`IllegalOfferId`	Offer identifier is empty or has an invalid format
`UnknownOfferId`	No offer was found with the given identifier.
`NotProxyOfferId`	Offer identifier does not refer to a proxy offer

CosTrading::OfferIterator Interface

```
interface OfferIterator
{
  unsigned long max_left()
  raises (UnknownMaxLeft);

  boolean next_n( in unsigned long n, out OfferSeq offers );

  void destroy();
};
```

Specifies methods to iterate through a list of offers.

OfferIterator::destroy()

```
void destroy();
```

Destroys the iterator object.

OfferInterator::max_left()

```
unsigned long max_left()
raises(UnknownMaxLeft);
```

Returns the number of offers remaining in the iterator.

Exceptions

`UnknownMaxLeft` cannot determine the number of remaining offers

OfferIterator::next_n()

```
boolean next_n(in unsigned long n,
               out OfferSeq offers);
```

Returns TRUE if offers contains more offer identifiers, and returns FALSE if offers is nil.

Parameters

n	Number of offers to return
ids	List of offers containing at most n elements

CosTrading::OfferIdIterator Interface

```
interface OfferIdIterator
{
  unsigned long max_left()
  raises (UnknownMaxLeft);

  boolean next_n(in unsigned long n, out OfferIdSeq ids);

  void destroy();
};
```

Specifies methods to iterate through a list of offer identifiers.

OfferIdInterator::destroy()

```
void destroy();
```

Destroys the iterator object.

OfferIdIterator::max_left()

```
unsigned long max_left()
raises(UnknownMaxLeft);
```

Returns the number of offer identifiers remaining in the iterator.

Exceptions

UnknownMaxLeft	Cannot determine the number of remaining offer identifiers

OfferIdIterator::next_n()

```
boolean next_n(in unsigned long n,
               out OfferIdSeq ids);
```

Returns TRUE if ids contains more offer identifiers, and returns FALSE if ids is nil.

Parameters

n	Number of offer identifiers to return
ids	List of offer identifiers containing at most n elements

CosTrading::Lookup Interface

```
interface Lookup :
          TraderComponents, SupportAttributes, ImportAttributes
{
  typedef Istring Preference;

  enum HowManyProps
  {
    none,
    some,
    all
  };

  union SpecifiedProps switch (HowManyProps)
  {
    case some: PropertyNameSeq prop_names;
  };

  exception IllegalPreference {Preference pref};
  exception IllegalPolicyName {PolicyName name};
  exception PolicyTypeMismatch {Policy the_policy};
  exception InvalidPolicyValue {Policy the_policy};

  void query(in ServiceTypeName type,
             in Constraint constr,
             in Preference pref,
             in PolicySeq policies,
             in SpecifiedProps desired_props,
             in unsigned long how_many,
             out OfferSeq offers,
             out OfferIterator offer_itr,
             out PolicyNameSeq limits_applied)
  raises (IllegalServiceType, UnknownServiceType,
          IllegalConstraint, IllegalPreference,
          IllegalPolicyName, PolicyTypeMismatch,
          InvalidPolicyValue, IllegalPropertyName,
          DuplicatePropertyName, DuplicatePolicyName);
};
```

Provides a single operation, query, for use by importers.

Lookup::Preference DataType

```
typedef Istring Preference;
```

A query preference expression. The preference is used to order the offers found by a query. The valid forms of a preference expression are:

min ***numeric-expression*** orders the offers in ascending order based on the numeric expression. Offers for which the expression cannot be evaluated (for example, if the offer does not contain a property that is used in the expression) are placed at the end of the sequence.

max ***numeric-expression*** orders the offers in descending order based on the numeric expression. Offers for which the expression cannot be evaluated (for example, if the offer does not contain a property that is used in the expression) are placed at the end of the sequence.

with ***boolean-expression*** orders the offers such that those for which the boolean expression are TRUE are included before any of those for which the expression is false, which are placed before any of those that cannot be evaluated.

random orders the offers in random order.

first orders the offers as they are encountered by the server.

If an empty preference expression is supplied, it is equivalent to a preference of `first`.

Lookup::HowManyProps Enum

```
enum HowManyProps
{
    none,
    some,
    all
```

```
};
```

The choices for indicating how many properties are returned with each offer. The members are defined as follows:

none	No properties should be returned.
some	Some properties should be returned.
all	All properties should be returned.

Lookup::SpecifiedProps Union

```
union SpecifiedProps switch(HowManyProps)
{
case some: PropertyNameSeq prop_names;
};
```

Determines which properties are to be returned for each matching offer found by the query operation. The union's discriminator can meaningfully be set to the other enumerated values none and all. If set to none, you are indicating that no properties should be returned. If set to all, then all properties will be returned. Set the value for some with a sequence of property names indicating which properties should be returned

Lookup::IllegalPolicyName Exception

```
exception IllegalPolicyName {PolicyName name};
```

The policy name is empty or does not conform the format supported by the trader. The invalid name is returned.

Lookup::IllegalPreference Exception

```
exception IllegalPreference {Preference pref};
```

An error occurred while parsing the preference expression. The invalid preference is returned.

Lookup::InvalidPolicyValue Exception

```
exception InvalidPolicyValue {Policy the_policy};
```

The policy has an invalid value.

Lookup::PolicyTypeMismatch Exception

```
exception PolicyTypeMismatch {Policy the_policy};
```

The policy value type specified does not match the type expected by the trader. The type expected by the trader is returned.

Lookup::query()

```
void query(in ServiceTypeName type,
           in Constraint constr,
           in Preference pref,
           in PolicySeq policies,
           in SpecifiedProps desired_props,
           in unsigned long how_many,
           out OfferSeq offers,
           out OfferIterator offer_itr,
           out PolicyNameSeq limits_applied)
raises(IllegalServiceType,
       UnknownServiceType,
       IllegalConstraint,
       IllegalPreference,
       IllegalPolicyName,
       PolicyTypeMismatch,
       InvalidPolicyValue,
       IllegalPropertyName,
       DuplicatePropertyName,
       DuplicatePolicyName);
```

Allows an *importer* to obtain references to objects that provide services meeting its requirements.

The importer can control the behavior of the search by supplying values for certain policies. The trader may override some or all of the values supplied by the importer. The following policies are known by the trader:

exact_type_match (boolean) if TRUE, only offers of exactly the service type specified by the importer are considered; if FALSE, offers of any service type that conforms to the importer's service type are considered

hop_count (unsigned long) indicates maximum number of hops across federation links that should be tolerated in the resolution of this query

link_follow_rule (FollowOption) indicates how the client wishes links to be followed in the resolution of this query

match_card (unsigned long) indicates the maximum number of matching offers to which the preference specification should be applied

return_card (unsigned long) indicates the maximum number of matching offers to return as a result of this query

search_card (unsigned long) indicates the maximum number of offers to be considered when looking for type conformance and constraint expression match

starting_trader (TraderName) specifies the remote trader at which the query starts

use_dynamic_properties (boolean) specifies whether to consider offers with dynamic properties

use_modifiable_properties (boolean) specifies whether to consider offers with modifiable properties

use_proxy_offers (boolean) specifies whether to consider proxy offers

Parameters

`type`	Specifies the service type that interests the importer. The service type limits the scope of the search to only those offers exported for this type, and optionally any subtype of this type.
`constr`	Limits the search to only those offers for which this expression is `TRUE`. The simplest constraint expression is "TRUE", which matches any offer.
`pref`	Specifies how the matched offers are t be ordered.
`policies`	Specifies the policies that govern the behavior of the query.
`desired_props`	Determines the properties that are to be included with each offer returned by the query. This parameter does not affect whether or not a service offer is returned. To exclude an offer that does not contain a desired property, include "exist *property-name*" in the constraint.
`how_many`	Indicates how many offers are to be returned in the `offers` parameter.
`offers`	Holds at most `how_many` offers. If the number of matching offers exceeds `how_many`, the `offer_itr` parameter will hold a reference to an iterator object through which the remaining offers can be obtained.
`offer_itr`	Will hold `nil` if no matching offers were found or if all of the matching offers were returned in `offers`; otherwise, holds a reference to an iterator. The object's `destroy` operation should be invoked when the object is no longer needed.
`limits_applied`	Holds the names of any policies that were overridden by the trader's maximum allowable settings.

Exceptions

`IllegalServiceType`	Service type name is empty or has an invalid format
`UnknownServiceType`	Service type was not found in service type repository
`IllegalConstraint`	An error occurred while parsing the constraint expression

IllegalPreference	An error occurred while parsing the preference expression
IllegalPolicyName	A policy name is empty or has an invalid format
PolicyTypeMismatch	A policy value type did not match the type expected by the trader
InvalidPolicyValue	A policy has an invalid value
IllegalPropertyName	A property name is empty or has an invalid format
DuplicatePropertyName	A property name appeared more than once in the list of desired properties
DuplicatePolicyName	A policy name appeared more than once in the list of policies

CosTrading::LinkAttributes Interface

LinkAttributes::max_link_follow_policy Attribute

```
readonly attribute FollowOption max_link_follow_policy;
```

Determines the most permissive behavior that will be allowed for any link.

CosTrading::Link Interface

```
interface Link :
          TraderComponents, SupportAttributes, LinkAttributes
{
  struct LinkInfo
  {
    Lookup target;
    Register target_reg;
    FollowOption def_pass_on_follow_rule;
    FollowOption limiting_follow_rule;
  };

  exception IllegalLinkName { LinkName name; };
  exception UnknownLinkName { LinkName name; };
  exception DuplicateLinkName { LinkName name; };

  exception DefaultFollowTooPermissive {
              FollowOption default_follow_rule;
              FollowOption limiting_follow_rule; };
  exception LimitingFollowTooPermissive {
              FollowOption limiting_follow_rule;
              FollowOption max_link_follow_policy; };

  void add_link( in LinkName name, in Lookup target,
                 in FollowOption default_follow_rule,
                 in FollowOption limiting_follow_rule )
  raises ( IllegalLinkName, DuplicateLinkName, InvalidLookupRef,
           DefaultFollowTooPermissive,
           LimitingFollowTooPermissive );

  void remove_link( in LinkName name )
  raises ( IllegalLinkName, UnknownLinkName );

  LinkInfo describe_link( in LinkName name )
  raises ( IllegalLinkName, UnknownLinkName );

  LinkNameSeq list_links();
```

```
  void modify_link( in LinkName name,
                    in FollowOption default_follow_rule,
                    in FollowOption limiting_follow_rule )
  raises ( IllegalLinkName, UnknownLinkName,
           DefaultFollowTooPermissive,
           LimitingFollowTooPermissive );
};
```

Provides structures, exceptions, and operations for managing links between traders.

Link::LinkInfo Data Structure

```
struct LinkInfo
{
    Lookup target;
    Register target_reg;
    FollowOption def_pass_on_follow_rule;
    FollowOption limiting_follow_rule;
};
```

A complete description of a link. The members hold the following information:

`target`	Lookup interface if link target
`target_reg`	`Register` interface of link
`def_pass_on_follow_rule`	Default link behavior for the link if no link-follow policy is specified by an importer durring a query
`limiting_follow_rule`	Most permisive link-follow behavior that the link is willing to tolerate

CosTrading::Link Exceptions

Link::DefaultFollowTooPermissive Exception

```
exception DefaultFollowTooPermissive
{
```

```
    FollowOption def_pass_on_follow_rule;
    FollowOption limiting_follow_rule;
};
```

Raised when the value for `def_pass_on_follow_rule` exceeds the value for `limiting_follow_rule`. Both values are passed back to the caller.

Link::DuplicateLinkName Exception

```
exception DuplicateLinkName {LinkName name};
```

Raised when a link already exists with the given name. The duplicated link name is passed back to the caller.

Link::IllegalLinkName Exception

```
exception IllegalLinkName {LinkName name};
```

Raised when the link name is empty or does not conform the format supported by the trader. The invalid link name is passed back to the caller.

Link::LimitingFollowTooPermissive Exception

```
exception LimitingFollowTooPermissive
{
    FollowOption limiting_follow_rule;
    FollowOption max_link_follow_policy;
};
```

The value for `limiting_follow_rule` exceeds the trader's `max_link_follow_policy` attribute.

Link::UnknownLinkName Exception

```
exception UnknownLinkName {LinkName name};
```

Raised when trader does not have a link with the given name. The invalid name is returned.

Link::add_link()

```
void add_link(in LinkName name, in Lookup target,
              in FollowOption def_pass_on_follow_rule,
              in FollowOption limiting_follow_rule)
raises(IllegalLinkName,
       DuplicateLinkName,
       InvalidLookupRef,
       DefaultFollowTooPermissive,
       LimitingFollowTooPermissive);
```

Adds a new, unidirectional link from this trader to another trader.

Parameters

`name`	Specifies the name of the new link.
`target`	Holds a reference to the `Lookup` interface of the target trader
`def_pass_on_follow_rule`	Specifies the default link behavior for the link if not link-follow policy is specified by an importer durring a query.
`limiting_follow_rule`	Specifies the most permisive link-follow behavior the the link is willing to follow.

Exceptions

`IllegalLinkName`	Link name is empty of has an invalid format.
`DuplicateLinkName`	Another link exists with the same name.
`InvalidLookupRef`	Targer object reference in `nil`.
`DefaultFollowTooPermisive`	The value for `def_pass_on_follow_rule` exceeds the value for `limiting_follow_rule`.
`LimitingFollowTooPermissive`	The value for `limiting_follow_rule` exceeds the trader's `max_link_follow_policy`.

Link::describe_link()

```
LinkInfo describe_link(in LinkName name)
raises(IllegalLinkName, UnknownLinkName);
```

Obtains a description of a link and returns it in a LinkInfo object.

Parameters

name	Name of the link of interest

Exceptions

IllegalLinkName The link name is empty or has an invalid format.

UnknownLinkName No link with the specified name exists.

Link::list_links()

```
LinkNameSeq list_links();
```

Reurns the names of all trading links within the trader.

Link::modify_link()

```
void modify_link(in LinkName name,
                 in FollowOption def_pass_on_follow_rule,
                 in FollowOption limiting_follow_rule)
raises(IllegalLinkName,
       UnknownLinkName,
       DefaultFollowTooPermissive,
       LimitingFollowTooPermissive);
```

Modifies the follow behavior of an existing link.

Parameters

name	Specifies the name of the link to be modified.

`def_pass_on_follow_rule`	Specifies the default link behavior for the link if no link-follow policy is specifed by an importer durring a query.
`limiting_follow_rule`	Describes the most permisive link-follow behavior that the link is willing to follow.

Exceptions

`IllegalLinkName`	Link name is empty of has an invalid format.
`UnknownLinkName`	The specified link name does not exist.
`DefaultFollowTooPermisive`	The value for `def_pass_on_follow_rule` exceeds the value for `limiting_follow_rule`.
`LimitingFollowTooPermissive`	The value for `limiting_follow_rule` exceeds the trader's `max_link_follow_policy`.

Link::remove_link()

```
void remove_link(in LinkName name)
    raises(IllegalLinkName, UnknownLinkName);
```

Removes an existing link.

Parameters

`name`	Name of the link to be removed

Exceptions

`IllegalLinkName`	The link name is empty or has an invalid format.
`UnknownLinkName`	No link exists witht the specified name.

CosTrading::ImportAttributes Interface

The read-only attributes of this interface provide the default and maximum values for policies that govern query operations.

Note: Performing a query is also known as *importing service offers*, therefore these attributes are called *import attributes*.

ImportAttributes::def_follow_policy Attribute

```
readonly attribute FollowOption def_follow_policy;
```

The default value for the `follow_policy` policy if it is not supplied.

ImportAttributes::def_hop_count Attribute

```
readonly attribute unsigned long def_hop_count;
```

The default value for the `hop_count` policy if it is not supplied.

ImportAttributes::def_match_card Attribute

```
readonly attribute unsigned long def_match_card;
```

The default value for the `match_card` policy if it is not supplied.

ImportAttributes::def_return_card Attribute

```
readonly attribute unsigned long def_return_card;
```

The default value for the `return_card` policy if it is not supplied.

ImportAttributes::def_search_card Attribute

```
readonly attribute unsigned long def_search_card;
```

The default value for the `search_card` policy if it is not supplied.

ImportAttributes::max_follow_policy Attribute

```
readonly attribute FollowOption max_follow_policy;
```

The maximum value for the `follow_policy` policy, which may override the value supplied by an importer.

ImportAttributes::max_hop_count Attribute

```
readonly attribute unsigned long max_hop_count;
```

The maximum value for the `hop_count` policy, which may override the value supplied by an importer.

ImportAttributes::max_list Attribute

```
readonly attribute unsigned long max_list;
```

The maximum size of any list returned by the trader. This may override the value supplied by a client to operations such as `query` and `next_n`.

ImportAttributes::max_match_card Attribute

```
readonly attribute unsigned long max_match_card;
```

The maximum value for the `match_card` policy, which may override the value supplied by an importer.

ImportAttributes::max_return_card Attribute

```
readonly attribute unsigned long max_return_card;
```

The maximum value for the `return_card` policy, which may override the value supplied by an importer.

ImportAttributes::max_search_card Attribute

```
readonly attribute unsigned long max_search_card;
```

The maximum value for the `search_card` policy, which may override the value supplied by an importer.

CosTrading::TraderComponents Interface

`interface TraderComponents`

Each of the five major interfaces of the `CosTrading` module inherit from this interface. By doing so, any of the *trader components* can be obtained using a reference to any of the other components.

A `nil` value will be returned by an attribute if the trader does not support that interface.

TraderComponents::admin_if Attribute

```
readonly attribute Admin admin_if;
```

TraderComponents::link_if Attribute

```
readonly attribute Link link_if;
```

TraderComponents::lookup_if Attribute

```
readonly attribute Lookup lookup_if;
```

TraderComponents::proxy_if Attribute

```
readonly attribute Proxy proxy_if;
```

TraderComponents::register_if Attribute

```
readonly attribute Register register_if;
```

CosTrading::Dynamic Module

Defines interfaces and types necessary to support dynamic properties. Dynamic properties allow an exporter to delegate a property's value to a third party. For example, rather than exporting an offer with a value of 54 for the property `weight`, you can provide a reference to an object that will dynamically compute the value for `weight`.

Naturally, there are performance issues when using dynamic properties, and therefore an importer may elect to exclude any offers containing dynamic properties.

To export an offer (or a proxy offer) with a dynamic property, you need to do the following:

- Define an object that implements the `DynamicPropEval` interface.
- Create an instance of the `DynamicProp` struct and insert that into the property's `CORBA::Any` value.
- Ensure that the lifetime of the `DynamicPropEval` object is such that it will be available whenever dynamic property evaluation is necessary.

CosTradingDynamic::DynamicProp Struct

```
struct DynamicProp
{
    DynamicPropEval eval_if;
    TypeCode returned_type;
    any extra_info;
};
```

Describes a dynamic property. This struct is inserted into a property's `CORBA::Any` value and provides all of the information necessary for the trader to accomplish dynamic property evaluation.

`eval_if`	Object reference for evaluation interface

`returned_type`	Value type expected for the property. The value of `returned_type` must match the value type of the property as defined by the service type.
`extra_info`	Additional information used for property evaluation. Orbix Trader supports primitive and user-defined types as values for `extra_info`.

CosTradingDynamic::DPEvalFailure Exception

```
exception DPEvalFailure
{
    CosTrading::PropertyName name;
    TypeCode returned_type;
    any extra_info;
};
```

Evaluation of a dynamic property failed.

`name`	Name of the property to be evaluated
`returned_type`	Value type expected for the property
`extra_info`	Additional information used for property evaluation

CosTradingDynamic:: DynamicPropEval Interface

```
interface DynamicPropEval
```

Defines a single operation for evaluating a dynamic property.

DynamicPropEval::evalDP()

```
any evalDP(in CosTrading::PropertyName name,
           in TypeCode returned_type,
           in any extra_info)
    raises(DPEvalFailure);
```

Evaluates a dynamic property and returns the objects properties.

Parameters

`name`	Name of the property to be evaluated
`returned_type`	Value type expected for the property
`extra_info`	Additional information used for property evaluation

Exceptions

`DPEvalFailure`	Evaluation of the property failed

CosTradingRepos Module

Contains the `ServiceTypeRepository` interface, which manages information about service types for the trading service.

A service type represents the information needed to describe a service, including an interface type defining the computational signature of the service, and zero or more properties that augment the interface. Each traded service, or service offer, is associated with a service type.

There are several components of a service type:

Interface: The interface repository identifier for an interface determines the computational signature of a service. If the trading service is configured to use the interface repository, and this identifier resolves to an `InterfaceDef` object in the interface repository, then the trading service will ensure that an object in an exported offer conforms to this interface.

Properties: Any number of properties can be defined for a service type. Properties typically represent behavioral, non-functional and non-computational aspects of the service.

Super types: Service types can be related in a hierarchy that reflects interface type inheritance and property type aggregation. This hierarchy provides the basis for deciding if a service of one type may be substituted for a service of another type.

When a new service type is added that has one or more super types, the service type repository performs a number of consistency checks. First, the repository ensures (if possible) that the interface of the new type conforms to the interface of the super type. Second, the repository checks for any property that has been redefined in the new service type to ensure that it has the same type as that of the super type, and that its mode is at least as strong as its mode in the super type.

CosTradingRepos:: ServiceTypeRepository Interface

`interface ServiceTypeRepository`

Contains types and operations for managing the repository.

ServiceTypeRepository::Identifier Alias

```
typedef CosTrading::Istring Identifier;
```

The interface repository identifier of an interface. For example, the identifier of this interface is `IDL:omg.org/CosTradingRepos/ServiceTypeRepository:1.0`.

ServiceTypeRepository::PropStructSeq Sequence

```
typedef sequence<PropStruct> PropStructSeq;
```

ServiceTypeRepository::ServiceTypeNameSeq Sequence

```
typedef sequence<CosTrading::ServiceTypeName> ServiceTypeNameSeq;
```

ServiceTypeRepository::ListOption Enum

```
enum ListOption
{
    all,
    since
};
```

Indicates which service types are of interest.

`all`	All service types
`since`	All service types since a particular incarnation

ServiceTypeRepository::PropertyMode Enum

```
enum PropertyMode
{
    PROP_NORMAL,
    PROP_READONLY,
    PROP_MANDATORY,
    PROP_MANDATORY_READONLY
};
```

Each property has a mode associated with it. The property mode places restrictions on an exporter when exporting and modifying service offers.

`PROP_NORMAL`	Property is optional
`PROP_READONLY`	Property is optional, but once a value has been supplied, it cannot be changed
`PROP_MANDATORY`	A value for this property must be supplied when the offer is exported, but can also be changed at some later time
`PROP_MANDATORY_READONLY`	A value for this property must be supplied when the offer is exported, and cannot be changed

ServiceType:Repository::IncarnationNumber Structure

```
struct IncarnationNumber
{
    unsigned long high;
    unsigned long low;
};
```

Represents a unique, 64-bit identifier that is assigned to each service type. This will be replaced by `long long` when that type is widely supported.

ServiceTypeRepository::PropStruct Structure

```
struct PropStruct
{
    CosTrading::PropertyName name;
    TypeCode value_type;
    PropertyMode mode;
};
```

A complete description of a property.

name	Name of the property
value_type	CORBA::TypeCode describing the type of values allowed for the property
mode	Determines whether a property is mandatory, and whether the property can be modified

ServiceTypeRepository::TypeStruct Structure

```
struct TypeStruct
{
    Identifier if_name;
    PropStructSeq props;
    ServiceTypeNameSeq super_types;
    boolean masked;
    IncarnationNumber incarnation;
};
```

A complete description of a service type.

if_name	Interface repository identifier for an interface
props	Defines the properties associated with this type
super_types	Service types from which this type inherits property definitions
masked	If TRUE, no new offers can be exported for this type
incarnation	Unique, 64-bit identifier for this type

ServiceTypeRepository::SpecifiedServiceTypes Union

```
union SpecifiedServiceTypes switch(ListOption)
{
case since: IncarnationNumber incarnation;
};
```

Provides two ways of retrieving the names of the service types managed by the repository. The union's discriminator can be set to `all` if you want to obtain all of the service type names.

`since`	Set this value with an incarnation number; only the names of those types whose incarnation numbers are greater than or equal to this value will be returned

ServiceTypeRepository::AlreadyMasked Exception

```
exception AlreadyMasked {CosTrading::ServiceTypeName name};
```

The service type cannot be masked if it is already masked.

ServiceTypeRepository::DuplicateServiceTypeName Exception

```
exception DuplicateServiceTypeName
{
CosTrading::ServiceTypeName name;
};
```

The same service type appeared more than once in the list of super types.

ServiceTypeRepository::HasSubTypes Exception

```
exception HasSubTypes
{
    CosTrading::ServiceTypeName the_type;
    CosTrading::ServiceTypeName sub_type;
};
```

A service type cannot be removed if it is the super type of any other type.

ServiceTypeRepository::InterfaceTypeMismatch Exception

```
exception InterfaceTypeMismatch
{
    CosTrading::ServiceTypeName base_service;
    Identifier base_if;
    CosTrading::ServiceTypeName derived_service;
    Identifier derived_if;
};
```

The interface of the new (*derived*) service type does not conform to the interface of a super type (*base service*).

ServiceTypeRepository::NotMasked Exception

```
exception NotMasked {CosTrading::ServiceTypeName name};
```

The service type cannot be unmasked if it is not currently masked.

ServiceTypeRepository::ServiceTypeExists Exception

```
exception ServiceTypeExists {CosTrading::ServiceTypeName name};
```

Another service type exists with the given name.

ServiceTypeRepository::ValueTypeRedefinition Exception

```
exception ValueTypeRedefinition
{
    CosTrading::ServiceTypeName type_1;
    PropStruct definition_1;
    CosTrading::ServiceTypeName type_2;
    PropStruct definition_2;
};
```

The definition of a property in the new service type (*type_1*) conflicts with the definition in a super type (*type_2*). This error can result if the `value_type` members do not match, or if the mode of the property is weaker than in the super type.

ServiceTypeRepository::incarnation Attribute

```
readonly attribute IncarnationNumber incarnation;
```

Determines the next incarnation number that will be assigned to a new service type. This could be used to synchronize two or more service type repositories, for example.

ServiceTypeRepository::add_type()

```
IncarnationNumber add_type(in CosTrading::ServiceTypeName name,
                           in Identifier if_name,
                           in PropStructSeq props,
                           in ServiceTypeNameSeq super_types)
raises(CosTrading::IllegalServiceType,
       ServiceTypeExists,
       InterfaceTypeMismatch,
       CosTrading::IllegalPropertyName,
       CosTrading::DuplicatePropertyName,
       ValueTypeRedefinition,
       CosTrading::UnknownServiceType,
       DuplicateServiceTypeName);
```

Adds a new service type and returns a unique identifier for the new type.

Parameters

`name`	Name to be used for the new type
`if_name`	Interface repository identifier for an interface
`props`	Properties defined for this interface interface
`super_types`	Zero or more super types from which this type will inherit interface and property definitions

Exceptions

CosTrading::IllegalServiceType	Service type name is empty or has an invalid format
ServiceTypeExists	Service type already exists with the same name
InterfaceTypeMismatch	Interface of the new type does not conform to the interface of a super type
CosTrading::IllegalPropertyName	Property name is empty or has an invalid format
CosTrading::DuplicatePropertyName	Same property name appears more than once in `props`
ValueTypeRedefinition	Property definition in `props` conflicts with a definition in a super type
CosTrading::UnknownServiceType	Super type does not exist
DuplicateServiceTypeName	Same super type name appears more than once in `super_types`

ServiceTypeRepository::describe_type()

```
TypeStruct describe_type(in CosTrading::ServiceTypeName name)
raises(CosTrading::IllegalServiceType,
       CosTrading::UnknownServiceType);
```

Gets the description of a service type and returns a `TypeStruct` with the description.

Parameters

`name`	Name of the type of interest

Exceptions

CosTrading::IllegalServiceType	Service type name is empty or has an invalid format
CosTrading::UnknownServiceType	Service type does not exist

ServiceTypeRepository::fully_describe_type()

```
TypeStruct fully_describe_type(in CosTrading::ServiceTypeName
                               name)
raises(CosTrading::IllegalServiceType,
       CosTrading::UnknownServiceType);
```

Obtains the *full* description of a service type. The super_types member of a full description contains the names of the types in the transitive closure of the super type relation. The props member includes all properties inherited from the transitive closure of the super types. A TypeStruct containing the full description is returned.

Parameters

name	Name of the type of interest

Exceptions

CosTrading::IllegalServiceType	Service type name is empty or has an invalid format
CosTrading::UnknownServiceType	Service type does not exist

ServiceTypeRepository::list_types()

```
ServiceTypeNameSeq list_types(in SpecifiedServiceTypes
                              which_types);
```

Lists the names of some or all of the service types in the repository.

Parameters

which_types	Specifies which types are of interest

ServiceTypeRepository::mask_type()

```
void mask_type(in CosTrading::ServiceTypeName name)
raises(CosTrading::IllegalServiceType,
       CosTrading::UnknownServiceType,
       AlreadyMasked);
```

Masks a service type so that offers can no longer be exported for it. Masking a service type is useful when the type is considered deprecated; in other words, no new offers should be allowed, but existing offers are still supported.

Parameters

`name`	Name of the type to be masked

Exceptions

`CosTrading::IllegalServiceType`	Service type name is empty or has an invalid format
`CosTrading::UnknownServiceType`	Service type does not exist
`AlreadyMasked`	Service type is already masked

ServiceTypeRepository::remove_type()

```
void remove_type(in CosTrading::ServiceTypeName name)
raises(CosTrading::IllegalServiceType,
       CosTrading::UnknownServiceType,
       HasSubTypes);
```

Removes an existing service type.

Parameters

`name`	Name of the type to be removed

Exceptions

`CosTrading::IllegalServiceType`	Service type name is empty or has an invalid format
`CosTrading::UnknownServiceType`	Service type does not exist
`HasSubTypes`	Service type cannot be removed if it is the super type of any other type

ServiceTypeRepository::unmask_type()

```
void unmask_type(in CosTrading::ServiceTypeName name)
raises(CosTrading::IllegalServiceType,
       CosTrading::UnknownServiceType,
       NotMasked);
```

Unmasks a masked service type so that offers can be exported for it.

Parameters

name	Name of the type to be unmasked

Exceptions

CosTrading::IllegalServiceType	Service type name is empty or has an invalid format
CosTrading::UnknownServiceType	Service type does not exist
NotMasked	Service type is not currently masked

Appendix A
System Exceptions

This appendix defines the system exceptions returned by Orbix.

BAD_CONTEXT	This exception is raised if a client invokes an operation but the passed context does not contain the context values required by the operation.
BAD_INV_ORDER	This exception indicates that the caller has invoked operations in the wrong order. For example, it can be raised by an ORB if an application makes an ORB-related call without having correctly initialized the ORB first.
BAD_OPERATION	This exception indicates that an object reference denotes an existing object, but that the object does not support the operation that was invoked.
BAD_PARAM	This exception is raised if a parameter passed to a call is out of range or otherwise considered illegal. For example, an ORB may raise this exception if null values or null pointers are passed to an operation (for language mappings where the concept of a null pointers or null values applies). BAD_PARAM can also be raised as a result of client generating requests with incorrect parameters using the DII.
BAD_TYPECODE	This exception is raised if the ORB encounters a malformed type code (for example, a type code with an invalid TCKind value).
COMM_FAILURE	This exception is raised if communication is lost while an operation is in progress, after the request was sent by the client, but before the reply from the server has been returned to the client.

DATA_CONVERSION	This exception is raised if an ORB cannot convert the representation of data as marshaled into its native representation or vice-versa. For example, DATA_CONVERSION can be raised if wide character codeset conversion fails, or if an ORB cannot convert floating point values between different representations.
FREE_MEM	This exceptioon is raised if the ORB failed in an attempt to free dynamic memory. For example, it is raised because of heap corruption or memory segments being locked.
IMP_LIMIT	This exception indicates that an implementation limit was exceeded in the ORB run time. For example, an ORB may reach the maximum number of references it can can hold simultaneously in an address space, the size of a parameter may have exceeded the allowed maximum, or an ORB may impose a maximum on the number of clients or servers that can run simultaneously.
INITIALIZE	This exception is raised if an ORB encounters a failure during its initialization, such as failure to acquire networking resources or detection of a configuration error.
INTERNAL	This exception indicates an interal failure in an ORB. For example, it is raised if an ORB detected corruption of its internal data structures.
INTF_REPOS	This exception is raised if an ORB cannot reach the interface repository, or some other failure relating to the interface repository is detected.
INV_FLAG	This exception indicates that an invalid flag was passed to an operation. For example, it is raised when creating a DII request.

INV_IDENT	This exception indicates that an IDL identifier is syntactically invalid. For example it may be raised if an identifier passed to the interface repository does not conform to IDL identifier syntax, or if an illegal operation name is used with the DII.
INV_OBJREF	This exception indicates that an object reference is internally malformed. For example, the repository ID may have incorrect syntax or the addressing information may be invalid. This exception is raised by ORB::string_to_object if the passed string does not decode correctly. An ORB implementation might detect calls via nil references (although it is not obliged to detect them). INV_OBJREF is used to indicate this.
INV_POLICY	This exception is raised when an invocation cannot be made due to an incompatibility between policy overrides that apply to the particular invocation.
INVALID_TRANSACTION	This exception indicates that the request carried an invalid transaction context. For example, this exception could be raised if an error occurred when trying to register a resource.
MARSHAL	This exception is raised if a request or reply from the network is structurally invalid. This error typically indicates a bug in either the client-side or server-side run time. For example, if a reply from the server indicates that the message contains 1000 bytes, but the actual message is shorter or longer than 1000 bytes, the ORB raises this exception. MARSHAL can also be caused by using the DII or DSI incorrectly. For example, it is raised if the type of the actual parameters sent does not agree with IDL signature of an operation.

NO_IMPLEMENT	This exception is raised if the operation that was invoked exists (it has an IDL definition) but no implementation for that operation exists. For example, NO_IMPLEMENT can be raised by an ORB if a client asks for an object's type definition from the interface repository, but no interface repository is provided by the ORB.
NO_MEMORY	This exception indicates that the ORB run time has run out of memory.
NO_PERMISSION	This exception is raised if an invocation fails because the caller has insufficient privileges.
NO_RESOURCES	This exception indicates that the ORB has encountered some general resource limitation. For example, the run time may have reached the maximum permissible number of open connections.
NO_RESPONSE	This exception is raised if a client attempts to retrieve the result of a deferred synchronous call but the response for the request is not yet available.
OBJ_ADAPTER	This exception typically indicates an administrative mismatch. For example, a server may have made an attempt to register itself with an implementation repository under a name that is already in use, or a name that is unknown to the repository. OBJ_ADAPTER is also raised by the POA to indicate problems with application-supplied servant managers.

OBJECT_NOT_EXIST	This exception is raised whenever an invocation on a deleted object is performed. It is an authoritative "hard" fault report. Anyone receiving it is allowed (even expected) to delete all copies of this object reference and to perform other appropriate "final recovery" style procedures. Bridges forward this exception to clients, also destroying any records they may hold (for example, proxy objects used in reference translation). The clients could in turn purge any of their own data structures.
PERSIST_STORE	This exception indicates a persistent storage failure. For example, it is raised if there is a failure to establish a database connection or corruption of a database.
REBIND	This exception is raised when the current effective RebindPolicy has a value of NO_REBIND or NO_RECONNECT and an invocation on a bound object reference results in a LocateReply message with status OBJECT_FORWARD or a Reply message with status LOCATION_FORWARD. This exception is also raised if the current effective RebindPolicy has a value of NO_RECONNECT and a connection must be re-opened. The invocation can be retried once the effective RebindPolicy is changed to TRANSPARENT or binding is re-established through an invocation of CORBA::Object::validate_connection().
TIMEOUT	This system exception is raised when no delivery has been made and the specified time-to-live period has been exceeded. It is a standard system exception because time-to-live QoS can be applied to any invocation.

TRANSACTION_MODE	The CosTransactions module adds the TRANSACTION_MODE exception that can be raised by the ORB when it detects a mismatch between the TransactionPolicy in the IOR and the current transaction mode.
TRANSACTION_REQUIRED	This exception indicates that the request carried a null transaction context, but an active transaction is required.
TRANSACTION_ROLLEDBACK	This exception indicates that the transaction associated with the request has already been rolled back or marked to roll back. The requested operation either could not be performed or was not performed because further computation on behalf of the transaction would be fruitless.
TRANSACTION_UNAVAILABLE	The CosTransactions module adds the TRANSACTION_UNAVAILABLE exception that can be raised by the ORB when it cannot process a transaction service context because its connection to the transaction service has been abnormally terminated.
TRANSIENT	This exception indicates that the ORB attempted to reach an object and failed. It is not an indication that an object does not exist. Instead, it simply means that no further determination of an object's status was possible because it could not be reached. For example, this exception is raised if an attempt to establish a connection fails because the server or the implementation repository is down.

UNKNOWN	This exception is raised if an operation implementation throws a non-CORBA exception (such as an exception specific to the implementation's programming language), or if an operation raises a user exception that does not appear in the operation's raises expression. UNKNOWN is also raised if the server returns a system exception that is unknown to the client. (This can happen if the server uses a later version of CORBA than the client and new system exceptions have been added to the later version.)

Appendix A

Index

B

C

D

E

F

G

H

I

J

K

L

M

N

O

P

Q

R

S

T

U

V

W

X

Y