SOMobjects Developer's Toolkit

Programmer's Reference, Volume II: Object Services

SOMobjects Version 3.0

Note: Before using this information and the product it supports, be sure to read the general information under **Notices** on page iii.

Second Edition (December 1996)

This edition of *Programmer's Reference for Object Services* applies to SOMobjects Developer's Toolkit for SOM Version 3.0 and to all subsequent releases of the product until otherwise indicated in new releases or technical newsletters.

Notices

Trademarks and Acknowledgements

Contents

About Programmer's Reference for Object Services

The *Programmer's Reference for Object Services* provides an implementation of standard interfaces defined by the Object Management Group (OMG). The SOMobjects Object Services are object-oriented class libraries for managing objects in distributed applications.

Who Should Use this Documentation

This documentation is for software developers using Object Services, as well as for developers who are providing specializations of object services interfaces.

You will find having the following background helpful:

- Familiarity with the OMG CORBA 1.1 and CORBA IDL specifications
- Familiarity with the OMG Common Object Services, in particular the:
  - Externalization Service
  - Naming Service
  - CosObject Identity Module (introduced in the Relationship Service)
- Knowledge of object-oriented principles
- C or C++ programming experience
- IBM SOM and DSOM knowledge, preferably with programming experience
- Familiarity with distributed systems management and object management concepts
- A careful examination of the information provided in *Programmer's Guide for Object Services*

Topics Covered

This documentation provides information about the SOMobjects Developer's Toolkit for SOM Version 3.0. Topics covered include:

- **Externalization Service**
- **Naming Service**
- **Object Services Server**

Typographic Conventions

This book uses the following typographic conventions:

Bold
Identifies commands, subroutines, keywords, files, structures, directories, and other items whose names are predefined by the system. Also identifies graphical objects such as buttons, labels, and icons that you select.

Italics
Identifies parameters and variables whose actual names or values you supply. Also identifies new terminology.

`Monospace`
Identifies examples of specific data values, examples of text similar to what you might see displayed, examples of portions of program code similar to what you might write as a programmer, messages from the system, or information you should actually type.

Related Publications

The following books contain information about, or related to, SOMobjects Object Services:

- *Common Object Services Specification Volume 1* (OMG Document Number 94-1-1)
- CORBA 1.1
- *Programmer's Guide for SOM and DSOM*
- *Programmer's Reference for SOM and DSOM*
- *Programmer's Guide for Object Services*
- *Programmer's Reference for Abstract Interface Definitions*

Relationship to Standards

SOMobjects Developer's Toolkit Object Services provide an implementation of standard interfaces that are defined by the OMG. Although it would be helpful, it is not necessary to be familiar with the OMG specifications themselves to understand SOMobjects Object Services. However, it is necessary to understand how those standards affected the structuring of the code and documentation for SOMobjects Object Services. This section looks at:

- Approaches to implementing standards
- SOMobjects Object Services' approach
- Structure used to document interfaces

For specific information about how the standard interfaces are implemented, refer to **Who Should Use This Documentation** on page vii in *Programmer's Guide for Object Services*.

Note: The following terms might be used interchangeably in this documentation:

interface* and *class — Typically *interface* is used to refer to an interface defined in CORBA IDL which are realized as *classes* in SOM.

object* and *instance — The term *object* is relatively common in object-oriented vernacular, whereas *instance* is used prodominately in C++ and SOMobjects to refer to an instance of an object.

operation* and *method — In CORBA IDL an interface introduces zero or more *operations*. Operations are commonly known as *methods* in SOMobjects and as *member functions* in C++.

Approaches to Implementing Standards

This section examines possible implementation approaches that can be used for the implementation of a standard interface. It is presented in generic terms without reference to the specific standards being implemented.

What Standards Provide

A standard for an object service provides interface definitions for one or more interfaces that comprise that service. In most cases, a service is composed of multiple interfaces that have a defined relationship to one another.

The interfaces are syntactically described using OMG CORBA IDL and the semantics of the interface are described using text descriptions. The use of IDL allows for a precise definition of the syntax; however, the semantic descriptions often require implementation-specific interpretation, which can vary from one implementation to another. In some cases, this is done on purpose and in others it is the result of insufficient description provided by the standard.

A standard may or may not be complete. That is, a standard may not provide sufficient definition of interfaces and operations for the service for use in a production environment. Consequently, extensions must be added to make it useful to a client. A standard almost always needs implementation-specific extensions.

Approaches to Implementing Standards

This section looks at various philosophies regarding the implementation of standard interfaces.

Pure Implementation: Provides an implementation of the specification without extensions. With this approach, the interfaces defined by the standard's IDL are implemented without adding operations to any of the interfaces and without extending the implementations through subtyping. This approach either requires the standard to provide sufficient interfaces and operations for the implementation, or the implementation must use mechanisms other than IDL- defined interfaces and operations to provide for inter-object interaction.

Because standards are seldom sufficiently defined for implementation and because IDL-defined operations are the normal way for distributed objects to communicate, there are few scenarios where this approach is practical.

Add Operations to Standard Interfaces: Provides an implementation of the specification by adding operations to the interfaces defined by the standard. With this approach, no new interfaces are introduced through subtyping. The additional operations can be used in the following ways:

- A way for two objects in the service to communicate in order to provide behavior defined by the standard, but the operations are not intended to be called by clients.
- A way to provide clients with access to behavior needed to make even a minimum implementation of the standard usable.
- An implementation-specific extension intended to give the client capabilities above and beyond what is defined by the standard.

The most likely scenario for the use of this approach would be the first case, where the additional operations are used within the service only and are not directly used by clients.

Subtype Standard Interfaces: Provides an implementation of the specification by subtyping the standard interfaces and defining additional operations in the new interface. In this approach, the interface defined by the standard is abstract and would never actually be instantiated as an object. Instead, the subtype interface or one of its descendents would be instantiated as an object. The additional operations can be used in the same manner as defined in **Add Operations to Standard Interfaces** on page xi.

When this approach is taken, an additional factor comes into play: where to provide the actual implementation for each of the operations. Here are a few of the approaches that can be taken:

- For the implementation of the class defined by the standard interface, do not implement any of the operations within that class. In the class that implements the subtype interface, override all of the operations in the standard interface. Provide the actual method implementations for operations defined in both the standard interface and subtype interface in the class for the subtype.

- For the implementation of the class defined by the standard interface, implement those methods that can be implemented without knowledge of the subtype. For operations whose methods need knowledge of the subtype, override the operations and provide the implementation in the class for the subtype.
- For the implementation of the class defined by the standard interface, implement all of the operations defined. Some of these methods will need to know specifics about the implementation of the subtype.

Of these choices, the last violates object-oriented principles, while either of the first two is an acceptable approach.

SOMobjects Object Services' Approach

Given the possible implementation approaches described in the previous section, this section describes the approach used by SOMobjects Object Services. How this affects the IDL is then described through a series of examples.

The approach SOMobjects Object Services takes is to subtype the standard interface and to provide all implementation of operations within the subclass. The IDL for the standard interface appears exactly as defined within the standard (with the exception being that some SOMobjects-specific information must be added), with no new interfaces or operations added. The IDL for the subtype inherits the interface defined for the standard, adds any additional operations needed, and overrides all of the operations defined for the standard interface. The name of the subtype interface will be a variation of the name of the standard interface, such as appending SOM to the beginning of it.

SOMobjects Developer's Toolkit sometimes introduces an abstract interface that it treats in the same way as the standard interfaces. These abstract interfaces are described in *Programmer's Reference for Abstract Interface Definitions*. Throughout these abstract interface descriptions, assume the same characteristics for the SOM-introduced abstract interfaces as you do for the OMG standard interfaces.

The following examples show the IDL for various cases encountered in SOMobjects Object Services.

Subtyping a Simple Interface

In this case, an interface is defined by the standard that is subtyped to define a class within which all implementation is done.

```
interface StandardInterface {

     void StandardOperation();

};


interface somExtendedInterface : StandardInterface {

     void AdditionalOperation();

     override:StandardOperation;

};
```

Subtyping an Interface that Inherits a Mixin

In this case, a standard introduces an interface that is intended to be a mixin interface. A mixin interface is not ever intended to be the principal interface for an object, but it is to be inherited into another interface. A standard may introduce a mixin interface that is only actually mixed in with one other interface in the standard. When this is the case, SOMobjects Object Services flatten this hierarchy and provide implementation for the operations defined in the mixin class in the subclass of the interface that inherits the mixin. The result looks something like this:

```
interface StandardMixin {

     void StandardMixinOperation();

};


interface StandardMixedInterface : StandardMixin {

     void StandardOperation();

};


interface somExtendedInterface : StandardInterface {

     void AdditionalOperation();

     override:StandardMixinOperation, StandardOperation;

};
```

Subtyping an Interface that Inherits a Non-Mixin Interface

Standards also introduce interfaces that inherit from each other that do not involve mixins. In this case, each standard interface is subtyped for implementation. The resulting structure appears as follows:

```
interface StandardBase {

     void StandardBaseOperation();

};


interface somExtendedBase : StandardBase {

     void AdditionalBaseOperation();

     override:StandardBaseOperation;

};


interface StandardInterface : StandardBase {

     void StandardOperation();

};
```

```
interface somExtendedInterface :
     StandardInterface, somStandardBase {

     void AdditionalOperation();

     override: StandardOperation;

};
```

In this case, **somExtendedInterface** inherits the implementation provided by **somExtendedBase**. This is the first case introduced where an implemented operation is inherited.

Overriding an Inherited Implementation

This example extends the previous example, overriding the implementation of an operation that has been inherited. The IDL would appear as follows:

```
interface StandardBase {

     void StandardBaseOperation1();
     void StandardBaseOperation2();

};


interface somExtendedBase : StandardBase {

     void AdditionalBaseOperation1();
     void AdditionalBaseOperation2();

     override: StandardBaseOperation1, StandardBaseOperation2;

};


interface StandardInterface : StandardBase {

     void StandardOperation();

};


interface somExtendedInterface :
     StandardInterface, somStandardBase {

     void AdditionalOperation();

     override: StandardOperation, AdditionalBaseOperation2,
          StandardBaseOperation2;

};
```

somExtendedInterface, the operations **AdditionalBaseOperation2** and **StandardBaseOperation2** are overridden. These overrides do not provide the initial implementation of the methods as they do in all other overrides presented here. They are used to extend, enhance, modify, and alter the behavior provided by the implementation of these operations in **somExtendedBase**.

Structure Used to Document Interfaces

Given the examples in the previous section, the structure used by this manual for documenting interfaces can be described. The following principles are followed:

- Specific knowledge of the syntax and semantics introduced by the standards is not required in order to understand this documentation.
- The standard interfaces are not specifically documented on their own as unique interfaces.
- Only interfaces that provide implementation are documented.
- Operations are treated as if they were initially introduced in the interface that provides the initial implementation rather than in the actual IDL interface where they are introduced.

The result of following these principles is documentation that is aligned with the hierarchy as implemented rather than aligned with the hierarchy as defined in IDL. This more closely aligns with the actual objects that users will create and use, providing a clearer understanding of how to use the services provided.

The examples in the previous section illustrate how interfaces and operations are introduced into the document. The IDL is repeated for ease in comparing the IDL to the documentation approach. For each IDL example, the classes that will be documented are listed along with the operations that will be documented within that class. Methods are treated as new methods if this is the initial implementation of the method; methods are treated as overrides if they are overriding a previous implementation.

Subtyping a Simple Interface

```
interface StandardInterface {

     void StandardOperation();

};


interface somExtendedInterface : StandardInterface {

     void AdditionalOperation();

     override:StandardOperation;

};
```

Documented Class	Treated as New Method	Treated as Overridden Method
somExtendedInterface	AdditionalOperation StandardOperation	

Table 1. Subtyping a Simple Interface

Subtyping an Interface that Inherits a Mixin

```
interface StandardMixin {

     void StandardMixinOperation();

};


interface StandardInterface : StandardMixin {

     void StandardOperation();

};


interface somExtendedInterface : StandardInterface {

     void AdditionalOperation();

     override: StandardMixinOperation, StandardOperation;

};
```

Documented Class	Treated as New Method	Treated as Overridden Method
somExtendedInterface	AdditionalOperation StandardMixinOperation StandardOperation	

Table 2. Subtyping a Mixin Interface

Subtyping an Interface that Inherits a Non-Mixin Interface

```
interface StandardBase {

     void StandardBaseOperation();

};


interface somExtendedBase : StandardBase {

     void AdditionalBaseOperation();

     override:StandardBaseOperation;

};


interface StandardInterface : StandardBase {

     void StandardOperation();

};
```

```
interface somExtendedInterface :
     StandardInterface, somExtendedBase {

     void AdditionalOperation();

     override:StandardOperation;

};
```

Documented Class	Treated as New Method	Treated as Overridden Method
somExtendedBase	AdditionalBaseOperation StandardBaseOperation	
somExtendedInterface	AdditionalOperation StandardOperation	

Table 3. Subtying an Interface that Inherits a Non-Mixin Interface

Overriding an Inherited Implementation

```
interface StandardBase {

     void StandardBaseOperation1();
     void StandardBaseOperation2();

};


interface somExtendedBase : StandardBase {

     void AdditionalBaseOperation1();
     void AdditionalBaseOperation2();

     override:StandardBaseOperation1, StandardBaseOperation2;

};


interface StandardInterface : StandardBase {

     void StandardOperation();

};


interface somExtendedInterface :
     StandardInterface, somExtendedBase {

     void AdditionalOperation();

     override:StandardOperation, AdditionalBaseOperation2,
          StandardBaseOperation2;

};
```

Documented Class	Treated as New Method	Treated as Overridden Method
somExtendedBase	AdditionalBaseOperation1 AdditionalBaseOperation2 StandardBaseOperation1 StandardBaseOperation2	
somExtendedInterface	AdditionalOperation StandardOperation	AdditionalBaseOperation2 StandardBaseOperation2

Table 4. Overriding an Inherited Implementation

Documentation of Attributes

Attributes are documented in terms of the **get** and **set** operations defined for them, and follow the same guidelines as described in the preceding sections regarding their treatment as either new or overridden methods. The following is an IDL example and its corresponding documentation:

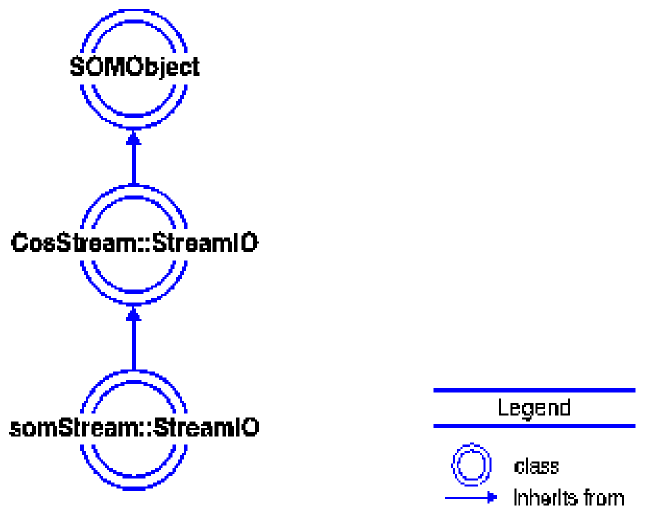
```
interface StandardInterface {

     attribute long StandardLong;

};


interface somExtendedInterface : StandardInterface {

     attribute long AdditionalLong;

     override: _get_StandardLong, _set_StandardLong;

};


interface somExtension : somExtendedInterface {

     override: _get_StandardLong, _set_StandardLong,
          _get_AdditionalLong, _set_AdditionalLong;

};
```

Documented Class	Treated as New Method	Treated as Overridden Method
somExtendedInterface	_get_StandardLong _set_StandardLong _get_AdditionalLong _set_AdditionalLong	
somExtension		_get_StandardLong _set_StandardLong _get_AdditionalLong _set_AdditionalLong

Table 5. Documenting Attributes

Chapter 1. Externalization Service

somStream::StreamIO Class



The **somStream::StreamIO** class is a subclass of the **CosStream::StreamIO** class. This class is base class for IBM supplied implementations of the OMG **CosStream::StreamIO** interface.

Intended Usage

This class is abstract and should not be instantiated by client applications. This class serves as the base for concrete implementations of the **StreamIO** interface.

File Stem

somestio

Base

CosStream::StreamIO

Metaclass

SOMClass

Ancestor Class

CosStream::StreamIO

SOMObject

New Methods

already_streamed Method

clear_buffer Method

get_buffer Method

read_boolean Method

read_char Method

read_double Method

read_float Method

read_long Method

read_short Method

read_string Method

read_unsigned_long Method

read_unsigned_short Method

read_octet Method

set_buffer Method

write_boolean Method

write_char Method

write_double Method

write_float Method

write_long Method

write_octet Method

write_string Method

write_unsigned_long Method

write_unsigned_short Method

Overridden Methods

somDefaultInit Method

somDestruct Method

Related Information

somStream::MemoryStreamIO Class

already_streamed Method

The **already_streamed** method determines if data for a specific parent class for a specific object has been written to, or read from, the stream.

IDL Syntax

boolean already_streamed(
in SOMObject *obj*,
in SOMObject *class_obj*);

Description

This method is useful for *diamond top* (multiple inheritance) situations.

Intended Usage

This method is intended to be used by specialized **StreamIO** implementations. It is not typically overridden.

This method is not defined in the OMG standard.

Parameters

obj
The specific object to check for inclusion in the stream.

class_obj
The specific class whose object is to be checked.

Return Value

True or false indicating whether the specified object is already in stream.

Example

The following example shows how the **already_streamed** method is used in the **externalize_to_stream** method of a **Car** class.

```
SOM_Scope void SOMLINK externalize_to_stream(Car somSelf,
     Environment *ev, CosStream_StreamIO stream) {

     CarData *somThis=CarGetData(somSelf);
     CarMethodDebug("Car", "externalize_to_stream");

     if (!_already_streamed(stream, ev, somSelf, _Car)) {
          Car_parent_Vehicle_externalize_to_stream(somSelf, ev,
               stream);
          _write_string(stream, ev, _name);
          _write_long(stream, ev, _size);
     }

}
```

Original Class

somStream::StreamIO Class

clear_buffer Method

The **clear_buffer** method erases the contents of the **StreamIO** buffer and resets the current position to the beginning.

IDL Syntax

void clear_buffer();

Description

The **clear_buffer** method erases the contents of the **StreamIO** buffer and resets the current position to the beginning. It makes the **StreamIO** essentially the same as a new one.

This must be overridden by subclasses. The implementation of **clear_buffer** should call the **read_octet Method**.

Intended Usage

This method is intended to be used by special client programs that need to reset the internal stream buffer. It must be overridden by specialized **StreamIO** classes with a specific implementation.

This method is not defined in the OMG standard.

Example

```
/*   myStreamIO needs to be declared as either
     somStream_MemoryStreamIO, somStream_StandardStreamIO, or
     somStream_StringStreamIO. */
...
_clear_buffer(myStreamIO, ev);
...
```

Original Class

somStream::StreamIO Class

get_buffer Method

The **get_buffer** method returns a copy of the **StreamIO** buffer.

IDL Syntax

somStream::seq_octet get_buffer();

Description

Returns copy of **StreamIO** buffer. The client is responsible for deallocating the memory returned in the sequence using the **somFree Method**.

This must be overridden by subclasses.

Intended Usage

This method is intended to be used by special client programs that need access to the internal stream buffer. It must be overridden by specialized **StreamIO** classes with a specific implementation.

This method is not defined in the OMG standard.

Return Value

A sequence of octets containing a copy of the **StreamIO** buffer.

Example

```
seq_octet inputBuffer;

/*   myStreamIO needs to be declared as either
     somStream_MemoryStreamIO, somStream_StandardStreamIO,
     or somStream_StringStreamIO. */
...
inputBuffer=_get_buffer(myStreamIO, ev);
...
```

Original Class

somStream::StreamIO Class

read_boolean Method

The **read_boolean** method reads a boolean from a stream.

IDL Syntax

boolean read_boolean();

Description

Streamable objects use this method to internalize their state data.

Intended Usage

This method is intended to be used by **Streamable** objects within their **internalize_from_-stream** method. It must be overridden by specialized **StreamIO** classes with a specific implementation.

Return Value

Returns the next value stored in the stream as a **boolean** value.

Example

```
CosStream_StreamIO myStream;

...
myBoolean=_read_boolean(myStream, ev);
```

Original Class

CosStream::StreamIO

read_char Method

The **read_char** method reads a char from a stream.

IDL Syntax

char read_char();

Description

Streamable objects use this method to internalize their state data.

Intended Usage

This method is intended to be used by **Streamable** objects within their **internalize_from_-stream** method. It must be overridden by specialized **StreamIO** classes with a specific implementation.

Return Value

Returns the next value stored in the stream as a **char** value.

Example

```
CosStream_StreamIO myStream;

...
myChar=_read_char(myStream, ev);
```

Original Class

CosStream::StreamIO

read_double Method

The **read_double** method reads a double from a stream.

IDL Syntax

double read_double();

Description

Streamable objects use this method to internalize their state data.

Intended Usage

This method is intended to be used by **Streamable** objects within their **internalize_from_-stream** method. It must be overridden by specialized **StreamIO** classes with a specific implementation.

Return Value

Returns the next value stored in the stream as a **double** value.

Example

```
CosStream_StreamIO myStream;

...
myDouble=_read_double(myStream, ev);
```

Original Class

CosStream::StreamIO

read_float Method

The **read_float** method reads a float from a stream.

IDL Syntax

float read_float();

Description

Streamable objects use this method to internalize their state data.

Intended Usage

This method is intended to be used by **Streamable** objects within their **internalize_from_-stream** method. It must be overridden by specialized **StreamIO** classes with a specific implementation.

Return Value

Returns the next value stored in the stream as a **float** value.

Example

```
CosStream_StreamIO myStream;

...
myFloat=_read_float(myStream, ev);
```

Original Class

CosStream::StreamIO

read_long Method

The **read_long** method reads a long integer from a stream.

IDL Syntax

long read_long();

Description

Streamable objects use this method to internalize their state data.

Intended Usage

This method is intended to be used by **Streamable** objects within their **internalize_from_-stream** method. It must be overridden by specialized **StreamIO** classes with a specific implementation.

Return Value

Returns the next value stored in the stream as a **long** value.

Example

```
CosStream_StreamIO myStream;

...
myLong=_read_long(myStream, ev);
```

Original Class

CosStream::StreamIO

read_octet Method

The **read_octet** method reads an octet from a stream.

IDL Syntax

octet read_octet();

Description

Streamable objects use this method to internalize their state data.

Intended Usage

This method is intended to be used by **Streamable** objects within their **internalize_from_-stream** method. It must be overridden by specialized **StreamIO** classes with a specific implementation.

Return Value

Returns the next value stored in the stream as an **octet** value.

Example

```
CosStream_StreamIO myStream;

...
myOctet=_read_octet(myStream, ev);
```

Original Class

CosStream::StreamIO

read_short Method

The **read_short** method reads a short integer from a stream.

IDL Syntax

short read_short();

Description

Streamable objects use this method to internalize their state data.

Intended Usage

This method is intended to be used by **Streamable** objects within their **internalize_from_-stream** method. It must be overridden by specialized **StreamIO** classes with a specific implementation.

Return Value

Returns the next value stored in the stream as a **short** value.

Example

```
CosStream_StreamIO myStream;

...
myShort=_read_short(myStream, ev);
```

Original Class

CosStream::StreamIO

read_string Method

The **read_string** method reads a string from a stream.

IDL Syntax

string read_string();

Description

Streamable objects use this method to internalize their state data.

Intended Usage

This method is intended to be used by **Streamable** objects within their **internalize_from_-stream** method. It must be overridden by specialized **StreamIO** classes with a specific implementation.

Return Value

Returns the next value stored in the stream as a **string** value.

The memory for the return value is owned by the caller and should be freed using **SOMFree** when it is no longer needed.

Example

```
CosStream_StreamIO myStream;

...
if (myString)
     SOMFree(myString,ev);
myString=_read_string(myStream, ev);
```

Original Class

CosStream::StreamIO

read_unsigned_long Method

The **read_unsigned_long** method reads an unsigned long integer from a stream.

IDL Syntax

unsigned long read_unsigned_long();

Description

Streamable objects use this method to internalize their state data.

Intended Usage

This method is intended to be used by **Streamable** objects within their **internalize_from_-stream** method. It must be overridden by specialized **StreamIO** classes with a specific implementation.

Return Value

Returns the next value stored in the stream as a **unsigned long** value.

Example

```
CosStream_StreamIO myStream;

...
myUnsignedLong=_read_unsigned_long(myStream, ev);
```

Original Class

CosStream::StreamIO

read_unsigned_short Method

The **read_unsigned_short** method reads an unsigned short integer from a stream.

IDL Syntax

unsigned short read_unsigned_short();

Description

Streamable objects use this method to internalize their state data.

Intended Usage

This method is intended to be used by **Streamable** objects within their **internalize_from_-stream** method. It must be overridden by specialized **StreamIO** classes with a specific implementation.

Return Value

Returns the next value stored in the stream as a **unsigned short** value.

Example

```
CosStream_StreamIO myStream;

...
myUnsignedShort=_read_unsigned_short(myStream, ev);
```

Original Class

CosStream::StreamIO

reset Method

The **reset** method resets the current position in the **StreamIO** buffer to the beginning of the buffer (first byte).

IDL Syntax

void reset();

Description

The **reset** method resets the current position in **StreamIO** buffer to the first byte. Subsequent reads or writes to the **StreamIO** begin with the first byte in the stream buffer. If the buffer has unused memory, the reset **method** shrinks the buffer using **SOMRealloc**. When the stream is reset, the **end_context** method is implicitly called.

This must be overridden by subclasses, and the subclass should call its parent **reset** method.

Intended Usage

This method is intended to be used by special client programs that need to reset the internal stream buffer. Normally it is invoked by the **clear_buffer** method. It must be overridden by specialized **StreamIO** classes with a specific implementation.

This method is not defined in the OMG standard.

Example

```
/*   myStreamIO needs to be declared as either
     somStream_MemoryStreamIO, somStream_StandardStreamIO,
     or somStream_StringStreamIO. */
...
_reset(myStreamIO,ev);
...
```

Original Class

somStream::StreamIO Class

set_buffer Method

The **set_buffer** method sets the **StreamIO** buffer to a copy of the buffer passed in.

IDL Syntax

void set_buffer(in somStream::seq_octet *buffer*);

Description

Sets the **StreamIO** buffer to a copy of the buffer parameter. The client retains ownership of the buffer parameter. The **StreamIO** current position is reset to the first byte in the buffer.

This must be overridden by subclasses.

Intended Usage

This method is intended to be used by special client programs that need to set the internal stream buffer. It must be overridden by specialized **StreamIO** classes with a specific implementation.

This method is not defined in the OMG standard.

Parameters

buffer
A sequence of octets to be copied into the **StreamIO** buffer.

Example

```
seq_octet inputBuffer;

/*   myStreamIO needs to be declared as either
     somStream_MemoryStreamIO, somStream_StandardStreamIO, or
     somStream_StringStreamIO. */
...

/* Allocate a fill the inputBuffer with valid stream data */
inputBuffer._buffer=SOMMalloc(datalen);
..

_set_buffer(myStreamIO, ev, &inputBuffer);
```

Original Class

somStream::StreamIO Class

write_boolean Method

The **write_boolean** method writes a boolean into a stream.

IDL Syntax

void write_boolean(in boolean *item*);

Description

Streamable objects use this method to externalize their state data.

Intended Usage

This method is intended to be used by **Streamable** objects within their **externalize_to-stream** method. It must be overridden by specialized **StreamIO** classes with a specific implementation.

Parameters

item
The data to be written into the stream, of data type boolean.

Example

```
CosStream_StreamIO myStream;

...
_write_boolean(myStream, ev, myBoolean);
```

Original Class

CosStream::StreamIO

write_char Method

The **write_char** method writes a char into a stream.

IDL Syntax

void write_char(in char *item*);

Description

Streamable objects use this method to externalize their state data.

Intended Usage

This method is intended to be used by **Streamable** objects within their **externalize_to-stream** method. It must be overridden by specialized **StreamIO** classes with a specific implementation.

Parameters

item
The data to be written into the stream, of data type char.

Example

```
CosStream_StreamIO myStream;

...
_write_char(myStream, ev, myChar);
```

Original Class

CosStream::StreamIO

write_double Method

The **write_double** method writes a double into a stream.

IDL Syntax

void write_double(in double *item*);

Description

Streamable objects use this method to externalize their state data.

Intended Usage

This method is intended to be used by **Streamable** objects within their **externalize_to-stream** method. It must be overridden by specialized **StreamIO** classes with a specific implementation.

Parameters

item
The data to be written into the stream, of data type **double**.

Example

```
CosStream_StreamIO myStream;

...
_write_double(myStream, ev, myDouble);
```

Original Class

CosStream::StreamIO

write_float Method

The **write_float** method writes a float into a stream.

IDL Syntax

void write_float(in float *item*);

Description

Streamable objects use this method to externalize their state data.

Intended Usage

This method is intended to be used by **Streamable** objects within their **externalize_to-stream** method. It must be overridden by specialized **StreamIO** classes with a specific implementation.

Parameters

item
The data to be written into the stream, of data type float.

Example

```
CosStream_StreamIO myStream;

...
_write_float(myStream, ev, myFloat);
```

Original Class

CosStream::StreamIO

write_long Method

The **write_long** method writes a long integer into a stream.

IDL Syntax

void write_long(in long *item*);

Description

Streamable objects use this method to externalize their state data.

Intended Usage

This method is intended to be used by **Streamable** objects within their **externalize_to-stream** method. It must be overridden by specialized **StreamIO** classes with a specific implementation.

Parameters

item
The data to be written into the stream, of data type long.

Example

```
CosStream_StreamIO myStream;

...
_write_long(myStream, ev, myLong);
```

Original Class

CosStream::StreamIO

write_octet Method

The **write_octet** method writes an octet into a stream.

IDL Syntax

void write_octet(in octet *item*);

Description

Streamable objects use this method to externalize their state data.

Intended Usage

This method is intended to be used by **Streamable** objects within their **externalize_to-stream** method. It must be overridden by specialized **StreamIO** classes with a specific implementation.

Parameters

item
The data to be written into the stream, of data type **octet**.

Example

```
CosStream_StreamIO myStream;

...
_write_octet(myStream, ev, myOctet);
```

Original Class

CosStream::StreamIO

write_short Method

The **write_short** method is writes a short integer into a stream.

IDL Syntax

void write_short(in short *item*);

Description

Streamable objects use this method to externalize their state data.

Intended Usage

This method is intended to be used by **Streamable** objects within their **externalize_to-stream** method. It must be overridden by specialized **StreamIO** classes with a specific implementation.

Parameters

item
The data to be written into the stream, of data type short.

Example

```
CosStream_StreamIO myStream;

...
_write_short(myStream, ev, myShort);
```

Original Class

CosStream::StreamIO

write_string Method

The **write_string** method writes a string into a stream.

IDL Syntax

void write_string(in string *item*);

Description

Streamable objects use this method to externalize their state data.

Intended Usage

This method is intended to be used by **Streamable** objects within their **externalize_to-stream** method. It must be overridden by specialized **StreamIO** classes with a specific implementation.

Parameters

item
The data to be written into the stream, of data type **string**.

Example

```
CosStream_StreamIO myStream;

...
_write_string(myStream, ev, myString);
```

Original Class

CosStream::StreamIO

write_unsigned_long Method

The **write_unsigned_long** method writes unsigned long integer into a stream.

IDL Syntax

void write_unsigned_long(in unsigned long *item*);

Description

Streamable objects use this method to externalize their state data.

Intended Usage

This method is intended to be used by **Streamable** objects within their **externalize_to-stream** method. It must be overridden by specialized **StreamIO** classes with a specific implementation.

Parameters

item
The data to be written into the stream, of data type unsigned long.

Example

```
CosStream_StreamIO myStream;

...
_write_unsigned_long(myStream, ev, myUnsignedLong);
```

Original Class

CosStream::StreamIO

write_unsigned_short Method

The **write_unsigned_short** method writes an unsigned short integer into a stream.

IDL Syntax

void write_unsigned_short(in unsigned short *item*);

Description

Streamable objects use this method to externalize their state data.

Intended Usage

This method is intended to be used by **Streamable** objects within their **externalize_to-stream** method. It must be overridden by specialized **StreamIO** classes with a specific implementation.

Parameters

item
The data to be written into the stream, of data type unsigned short.

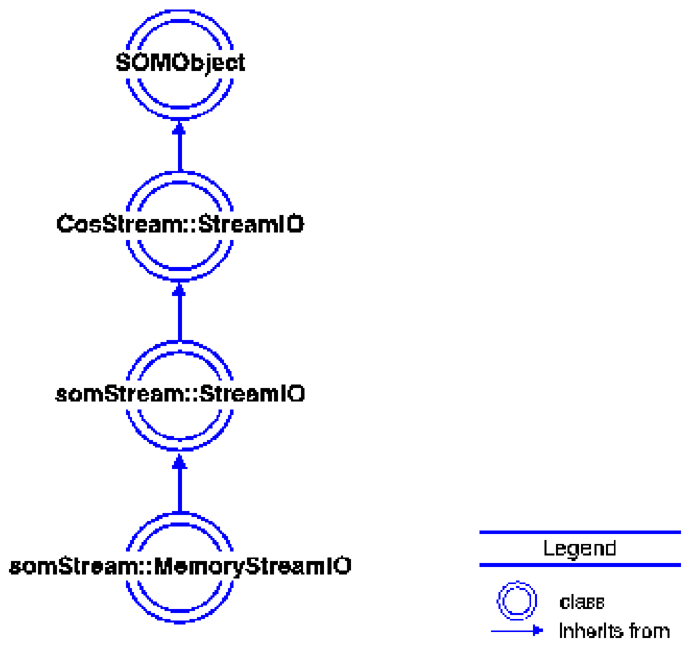
Example

```
CosStream_StreamIO myStream;
unsigned short myUnsignedShort;


...
_write_unsigned_short(myStream, ev, myUnsignedShort);
```

Original Class

CosStream::StreamIO

somStream::MemoryStreamIO Class



The **somStream::MemoryStreamIO** implementation stores the data in the format native to the process in which the buffer resides. The code page of the character data, the endian format, and the floating point format can vary.

The **somStream::MemoryStreamIO** class is a subclass of the **somStream::StreamIO** class. This class implements the **read_<*type*>**, **write_<*type*>** and other abstract methods introduced in **somStream::StreamIO Class**. This class is a complete and usable implementation of the OMG **CosStream::StreamIO** interface.

File Stem

somestio

Base

somStream::StreamIO

Metaclass

SOMClass

Ancestor Class

somStream::StreamIO Class

CosStream::StreamIO

SOMObject Class

New Methods

None.

Overridden Methods

clear_buffer Method

get_buffer Method

read_boolean Method

read_char Method

read_double Method

read_float Method

read_long Method

read_octet Method

read_short Method

read_string Method

read_unsigned_long Method

read_unsigned_short Method

read_octet Method

set_buffer Method

somDefaultInit Method

somDestruct Method

write_boolean Method

write_char Method

write_double Method

write_float Method

write_long Method

write_octet Method

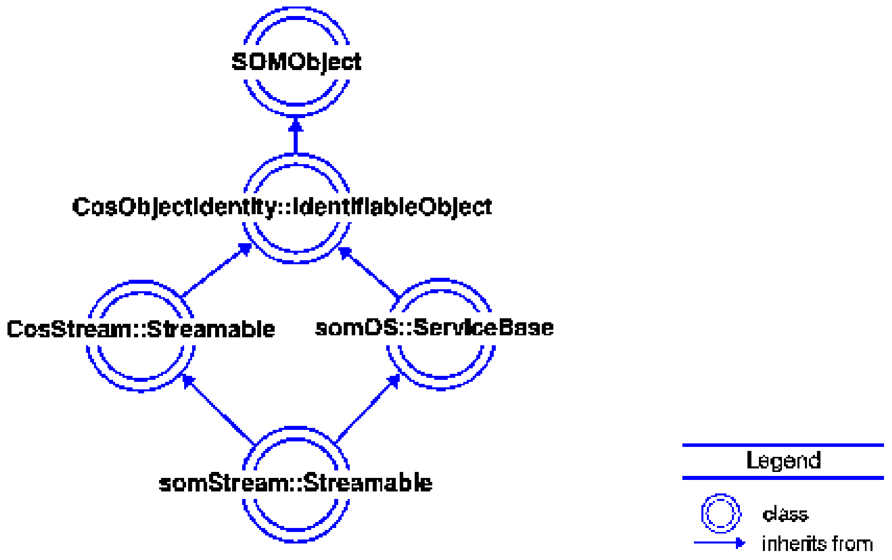
write_string Method

write_unsigned_long Method

write_unsigned_short Method

See **somStream::StreamIO Class** for a complete specification of these methods.

somStream::Streamable Class



The **somStream::Streamable** class is an implementation of the **CosStream::Streamable** interface. This class provides the methods necessary to convert the state data of an object into and out of this externalized form.

The **somStream::Streamable** class also inherits from the **somOS::ServiceBase** class, which supplies the implementation for **IdentifiableObject**.

File Stem

somestio

Base

somOS::ServiceBase

CosStream::Streamable

Metaclass

SOMClass

Ancestor Class

somOS::ServiceBase

CosObjectIdentity::IdentifiableObject

CosStream::Streamable

SOMObject Class

New Methods

externalize_to_stream Method

internalize_from_stream Method

Overridden Methods

somDefaultInit Method

somDestruct Method

externalize_to_stream Method

The **externalize_to_stream** method writes the state data of a streamable object to a stream.

IDL Syntax

void externalize_to_stream (in StreamIO *stream*);

Description

The **externalize_to_stream** method writes the object's state data to a stream. This is done through various calls to the **write_<*type*>** methods. The amount of data to be externalized and the order in which it is externalized are determined by the object.

This method must be overridden by the object provider. The implementation should call each parent **externalize_to_stream** method.

Intended Usage

This method is intended to be used by client applications. It is not typically overridden.

Parameters

stream
A stream object.

Original Class

CosStream::Streamable

internalize_from_stream Method

The **internalize_from_stream** method reads the state data of an object from a stream.

IDL Syntax

```
void internalize_from_stream (
        in StreamIO stream,
        in CosLifeCycle::FactoryFinder ff);
```

Description

The **internalize_from_stream** method reads the state data of an object from a stream. This is done through various calls to the **read_*<type>*** methods. The amount of data to be internalized and the order in which it is internalized are determined by the object.

This method must be overridden by the object provider. The implementation should call each parent **internalize_from_stream** method.

Intended Usage

This method is intended to be used by client applications. It is not typically overridden.

Parameters

stream
A stream object.

ff
A LifeCycle factory finder. This parameter is provided only for use in calling the **read_object** method.

Original Class

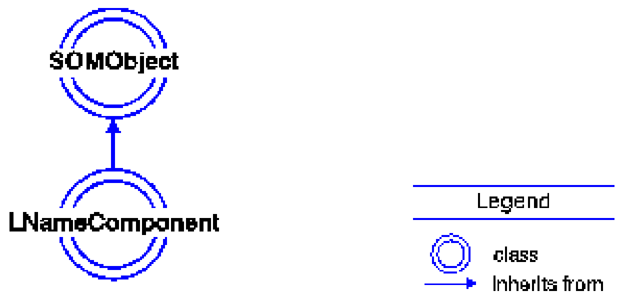
CosStream::Streamable

Chapter 2. Naming Service

The Naming Service provides support for the *OMG Naming Specification* and IBM Extended Naming enhancements. The services introduces two primary interfaces — **CosNaming::NamingContext** and **ExtendedNaming::ExtendedNamingContext** — and a number of secondary interfaces. Most of these are implemented in classes introduced in the **FileXNaming** module — in particular **CosNaming::NamingContext** and **ExtendedNaming::ExtendedNamingContext** are implemented in **FileXNaming::FileENC**. **LName** and **LNameComponent** are implemented separately.

The **ExtendedNaming::ExtendedNamingContext** interface introduces methods for searching a naming context based on the property-values assigned to a name-binding. The BNF for the constraint expression is provided in **Appendix A, BNF for Naming Constraint Language**.

LNameComponent Class



The **LNameComponent** class provides support for OMG library name components.

Intended Usage

The *OMG Names Library* implements names as *pseudo-objects*. This provides a convenient way of constructing **CosNaming::Name**s, which otherwise have to be assembled into a structure. The Names Library, also known as an **LName**, makes it easier to build a compound name by constructing individual name components — **LNameComponent**s — which can then be inserted into the **LName**.

A client makes calls on pseudo-objects the same way as on ordinary objects. Pseudo-object references cannot be passed across IDL interfaces.

The **LNameComponent** class is part of the OMG names library. The names library consists of two interfaces: the **LNameComponent**, and the **LName**. Name components consist of two elements: an **ID**, and a **kind**. This class defines methods to manipulate these two elements in the **LNameComponent**, and for inserting **LNameComponent**s into an **LName**.

Also, see “LName Class” on page 43

File Stem

lnamec

Base Classes

SOMObject Class

Ancestor Classes

None.

New Methods

destroy Method
get_id Method
get_kind Method
set_id Method
set_kind Method

Exceptions

CORBA 1.1 standard exceptions and the following user exceptions:

- **LNameComponent::NotSet** is raised to indicate that the attribute has not been set.

Related Information

See **LName Class** on page 43.

Creating a Library Name Component

To create a Library Name Component pseudo-object use the following function:

CosNaming::LNameComponent create_lname_component();

The returned pseudo-object can then be operated on using the operations defined for the **LNameComponent** class.

destroy Method

Destroys the library name component.

IDL Syntax

```
void destroy();
```

Description

Destroys the library name component. It invokes the **somFree Method** to free the object.

Intended Usage

This method is intended to be used by client applications. It is not typically overridden.

Exceptions

CORBA 1.1 standard exceptions.

Original Class

LNameComponent Class

get_id Method

Retrieves the *ID* element of a library name component.

IDL Syntax

```
string get_id();
```

Description

A name component has the two elements ***ID*** and ***kind***. The **get_id** method retrieves the *ID* element of a library name component.

Intended Usage

This method is intended to be used by client applications. It is not typically overridden.

Return Value

A string is returned, which is the *ID* element of the target library name component.

Exceptions

CORBA 1.1 standard exceptions and the following user exceptions:

LNameComponent::NotSet is raised if the *ID* element has not been set in the target library name component.

Original Class

LNameComponent Class

get_kind Method

Retrieves the *kind* element of a library name component.

IDL Syntax

string get_kind();

Description

A name component has the two elements ***id*** and ***kind***. The **get_kind** method retrieves the *kind* element of a library name component.

Intended Usage

This method is intended to be used by client applications. It is not typically overridden.

Return Value

A string is returned representing the *kind* element of the target library name component.

Exceptions

CORBA 1.1 standard exceptions and the following user exceptions:

LNameComponent::NotSet is raised if the *kind* element has not been set in the target library name component.

Original Class

LNameComponent Class

set_id Method

Sets the *ID* element of a library name component.

IDL Syntax

void set_id(in string *ID*);

Description

Sets the *ID* of a library name component.

Intended Usage

This method is intended to be used by client applications. It is not typically overridden.

Parameters

ID
A string to be used for the *ID* element of the name component.

Exceptions

CORBA 1.1 standard exceptions.

Original Class

LNameComponent Class

set_kind Method

Sets the *kind* element of a library name component.

IDL Syntax

void set_kind(in string *kind*);

Description

Sets the *kind* element of a library name component.

Intended Usage

This method is intended to be used by client applications. It is not typically overridden.

Parameters

kind
A string to be used for the kind.

Return Value

void

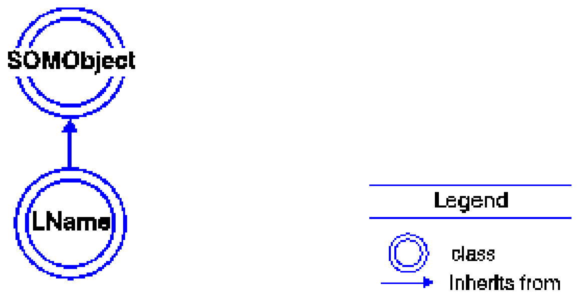
Exceptions

CORBA 1.1 standard exceptions.

Original Class

LNameComponent Class

LName Class



The **LName** class provides support for OMG library names.

Intended Usage

The *OMG Names Library* implements names as *pseudo-objects*. A client makes calls on pseudo-objects in the same way it makes calls on ordinary objects. Pseudo-object references cannot be passed across IDL interfaces.The Names Library supports operations to convert a Library Name into a value that can be passed to the Name Service through the **NamingContext** interface.

The **LName** class is part of the OMG names library. The names library consists of two interfaces: the **LNameComponent**, and the **LName**. Names consist of one or more *name components*. Each component, except the last, is used to identify names of subcontexts. The last component denotes a bound object. This class defines methods to manipulate name components for a name.

File Stem

lname

Base Classes

SOMObject Class

Ancestor Classes

None.

New Methods

delete_component Method
destroy Method
equal Method
from_idl_form Method
get_component Method
insert_component Method
less_than Method
num_components Method
to_idl_form Method

Exceptions

CORBA 1.1 standard exceptions and the following user exceptions:

LName::NoComponent is raised for **insert** methods, with indicator *i* representing the ith **LNameComponent** if the LNameComponent *i* –1 is undefined and if **LNameComponent** *i* is greater than 1. This exception is also raised for **get** and **delete** methods if component *i* does not exist.

LName::OverFlow is raised if resources cannot be allocated.

Related Information

See **LNameComponent Class** on page 36.

Creating a Library Name

To create a Library Name pseudo-object, use the following function:

LName create_lname();

The returned pseudo-object can then be operated on using the operations defined for the **LName** class.

delete_component Method

Deletes a library name component from a library name.

IDL Syntax

CosNaming::LNameComponent delete_component(in unsigned long *i*);

Description

Deletes the *ith* name component from a library name. The first position is position 1. After a delete operation has been performed without error, the compound name has one fewer component, and components previously identified as *i+1...n* are now identified as *i...n-1*.

Intended Usage

This method is intended to be used by client applications. It is not typically overridden.

Parameters

i
The *ith* position for the delete; 1-origin.

Return Value

This method returns the deleted **LNameComponent.**

Exceptions

CORBA 1.1 standard exceptions and the following user exceptions:

LName::NoComponent is raised if component *i* does not exist.

Original Class

LName Class

destroy Method

Destroys a library name.

IDL Syntax

void destroy();

Description

Destroys the library name. It invokes the **somFree Method** to free the object.

Intended Usage

This method is intended to be used by client applications. It is not typically overridden.

Exceptions

CORBA 1.1 standard exceptions.

Original Class

LName Class

equal Method

Determines equality of another library name.

IDL Syntax

boolean equal(in CosNaming::LName *lname*);

Description

Determines equality of another library name. Two library names are equal if both have the same number of components and if the *id* and *kind* elements are identical for every component.

Intended Usage

This method is intended to be used by client applications. It is not typically overridden.

Parameters

lname
The library name to be used for comparison.

Return Value

This method returns a Boolean value, indicating TRUE if equal, or FALSE if not.

Exceptions

CORBA 1.1 standard exceptions.

Original Class

LName Class

from_idl_form Method

Translates from an IDL form into a library name.

IDL Syntax

void from_idl_form(in CosNaming::Name *name*);

Description

CORBA specifies a library name as a pseudo-object; therefore, it cannot be passed across an IDL interface. The **from_idl_form** method sets the component's *ID* and *kind* elements of the target library name from the *name*.

The **CosNaming::NamingContext** interface and the **ExtendedNaming::Extended-NamingContext** interface define operations that return an **CosNaming::Name** structure. Because the library name is a pseudo-object, this method sets the *ID* and *kind* elements of the library name.

Intended Usage

This method is intended to be used by client applications. It is not typically overridden.

Parameters

name
The name to be used for the translation.

Exceptions

CORBA 1.1 standard exceptions.

Original Class

LName Class

get_component Method

Retrieves a library name component from a library name.

IDL Syntax

CosNaming::LNameComponent get_component(in unsigned long *i*);

Description

Retrieves the *ith* library name component from a library name. The first position is position 1.

Clients typically use this method to retrieve a component from a library name object. Then they use the **get_id** and **get_kind** methods to extract the *ID* and *kind* elements of the component.

Intended Usage

This method is intended to be used by client applications. It is not typically overridden.

Parameters

i
The *ith* position for the retrieval; 1-origin.

Return Value

An **LNameComponent** is returned, which is the requested library name component.

Exceptions

CORBA 1.1 standard exceptions and the following user exceptions:

- **LName::NoComponent** is raised if component *i* does not exist.

Original Class

LName Class

insert_component Method

Inserts a library name component into a library name.

IDL Syntax

```
CosNaming::LName insert_component(
        in unsigned long i,
        in CosNaming::LNameComponent lname_comp);
```

Description

Inserts a library name component into a library name. The library name component is inserted after the specified position. The first position is position 1.

This method is used to insert a library name component into a library name. After a library name object is created, clients can add more components to the library name.

Intended Usage

This method is intended to be used by client applications. It is not typically overridden.

Parameters

i
The *ith* position for the insert; 1-origin. The component is inserted immediately preceding this position.

lname_comp
The library name component to be inserted.

Return Value

An **LName** is returned, which is the target library name.

Exceptions

CORBA 1.1 standard exceptions and the following user exceptions:

LName::NoComponent is raised if the **LNameComponent** *i* –1 is undefined and **LNameComponent** *i* is greater than 1.

LName::OverFlow is raised if resources cannot be allocated.

Original Class

LName Class

less_than Method

Determines order of another library name.

IDL Syntax

boolean less_than(in CosNaming::LName *lname*);

Description

Tests for the order of the library name in relation to library name *lname*. Returns TRUE if:

- The number of components in the target library name is less than the number in the passed argument *lname*.
- The number of components in the target equals the number in the passed argument ln and there is it least one component whose *ID* and *kind* elements are lexically less than the same fields of the corresponding component in the library name *lname*.

Intended Usage

This method is intended to be used by client applications. It is not typically overridden.

Parameters

lname
The library name to be used for comparison.

Return Value

The method returns a Boolean value, indicating TRUE if the target library name is less than the library name passed as an argument.

Exceptions

CORBA 1.1 standard exceptions.

Original Class

LName Class

num_components Method

Retrieves the number of library name components in a library name.

IDL Syntax

```
unsigned long num_components();
```

Description

Retrieves the number of library name components in a library name.

Intended Usage

This method is intended to be used by client applications. It is not typically overridden.

Return Value

An unsigned long is returned, which is the number of library name components.

Exceptions

CORBA 1.1 standard exceptions.

Original Class

LName Class

to_idl_form Method

Produces a **CosNaming::Name** for a library name suitable for transmitting to an IDL-defined interface.

IDL Syntax

CosNaming::Name to_idl_form();

Description

Produces a **CosNaming::Name** for the target library name. This operation produces a structure that can be passed across an IDL request.

Several operations on naming contexts have arguments of type IDL structure **CosNaming::Name**. Because library name is a pseudo-object, it cannot be passed across an IDL interface. Clients can use this method to convert a library name to a **CosNaming::Name** structure and pass it across IDL interfaces.

Intended Usage

This method is intended to be used by client applications. It is not typically overridden.

Return Value

This method returns a **CosNaming::Name** from the target **LName**.

Exceptions

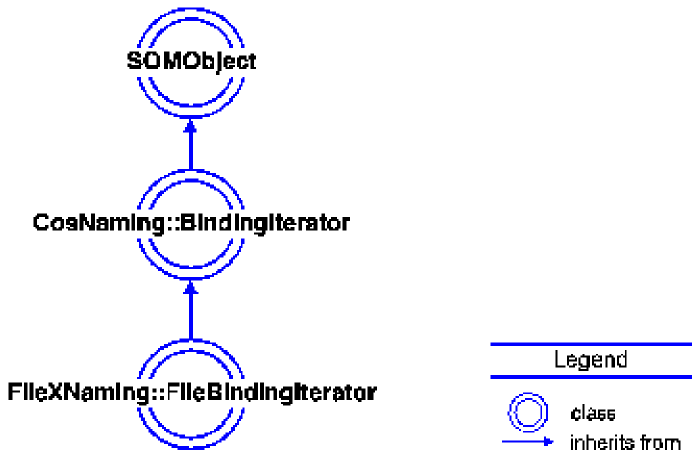
CORBA 1.1 standard exceptions and the following user exceptions:

CosNaming::InvalidName

Original Class

LName Class

FileXNaming::FileBindingIterator Class



The **FileXNaming::FileBindingIterator** class provides support for OMG binding iteration.

Intended Usage

This class is instantiated and returned as an out parameter in the **CosNaming::Naming-Context::list** method if the targeted naming context contains more name-object bindings than requested.

File Stem

xnamingf

Base Classes

SOMObject Class

Ancestor Classes

CosNaming::BindingIterator Class

Types

```
typedef string Istring;

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

enum BindingType {nobject, ncontext};

struct Binding {
        Name binding_name;
        BindingType binding_type;
};
typedef sequence <Binding> BindingList;
```

New Methods

destroy Method
next_n Method
next_one Method

Overridden Methods

somDefaultInit Method
somDestruct Method

Exceptions

CORBA 1.1 standard exceptions.

destroy Method

Destroys the iterator.

IDL Syntax

void destroy();

Description

Destroys the iterator and frees up allocated memory. It invokes the **somFree Method** to free the object.

Intended Usage

This method is intended to be used by client applications. It is not typically overridden.

Exceptions

CORBA 1.1 standard exceptions.

Original Class

CosNaming::BindingIterator

next_n Method

Retrieves at most the specified number of name-object bindings.

IDL Syntax

```
boolean next_n(
        in unsigned long how_many,
        out CosNaming::BindingList blist);
```

Description

Returns *how_many* bindings in the *blist* parameter. This method will return fewer bindings if less than how_many bindings remain in the iterator. With the **next_n** operation, clients can iterate through the bindings. Returns FALSE if there are no more bindings to return.

Intended Usage

This method is intended to be used by client applications. It is not typically overridden.

Parameters

how_many
The maximum number of bindings to be returned.

blist
The returned **BindingList**.

Return Value

This method returns a Boolean value where FALSE indicates to the client that there are no more bindings and where TRUE indicates more bindings exist.

Exceptions

CORBA 1.1 standard exceptions.

Original Class

CosNaming::BindingIterator

next_one Method

Retrieves the next name-object binding.

IDL Syntax

boolean next_one(out CosNaming::Binding *binding*);

Description

Returns the next name-object binding in the *binding* parameter. A FALSE is returned if there are no more bindings.

Intended Usage

This method is intended to be used by client applications. It is not typically overridden.

Parameters

binding
The returned Binding.

Return Value

This method returns a Boolean value where FALSE indicates to the client that there are no more bindings.

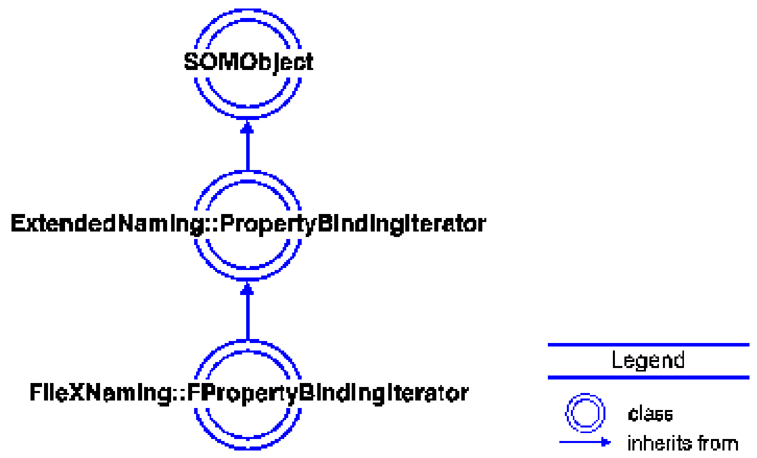
Exceptions

CORBA 1.1 standard exceptions.

Original Class

CosNaming::BindingIterator

FileXNaming::FPropertyBindingIterator Class


SOMObject
ExtendedNaming::PropertyBindingIterator
FileXNaming::FPropertyBindingIterator
Legend
class
inherits from

The **FileXNaming::PropertyBindingIterator** class provides support for property binding iteration.

Intended Usage

This class is instantiated and returned through the **ExtendedNaming::ExtendedNaming-Context::list_properties** method if an extended naming context contains more property bindings than requested.

File Stem

xnamingf

Base Classes

SOMObject Class

Ancestor Classes

ExtendedNaming::PropertyBindingIterator Class

Types

```
typedef struct PropertyBinding_struct {
        CosNaming::Istring property_name;
        boolean sharable;
} PropertyBinding;
typedef sequence<PropertyBinding> PropertyBindingList;
```

New Methods

destroy Method
next_n Method
next_one Method

Overridden Methods

somDefaultInit Method
somDestruct Method

Exceptions

CORBA 1.1 standard exceptions.

destroy Method

Destroys the iterator.

IDL Syntax

void destroy();

Description

Destroys the iterator. It invokes the **somFree Method** to free the object.

Intended Usage

This method is intended to be used by client applications. It is not typically overridden.

This method is not defined in the OMG standard.

Exceptions

CORBA 1.1 standard exceptions.

Original Class

ExtendedNaming::PropertyBindingIterator Interface

Related Information

list_properties Method

next_n Method

Retrieves a specified maximum number of property bindings.

IDL Syntax

boolean next_n(
  in unsigned long *how_many*,
  out ExtendedNaming::PropertyBindingList *pblist*);

Description

Returns *how_many* bindings in the *pblist* parameter.This method is used, in standard CORBA fashion, to obtain the next several property bindings from the extended naming context with which the targeted **FPropertyBindingIterator** is associated. Calling programs should check the return value for decision making for further invocations on the iterator. The method returns FALSE if there are no more property bindings to obtain.

Intended Usage

This method is intended to be used by client applications. It is not typically overridden.

This method is not defined in the OMG standard.

Parameters

how_many
Maximum number of bindings to return.

pblist
The returned **PropertyBindingList**.

Return Value

This method returns a Boolean value where FALSE indicates to the client that there are no more property bindings and where TRUE indicates more property bindings exist.

Exceptions

CORBA 1.1 standard exceptions.

Original Class

ExtendedNaming::PropertyBindingIterator Interface

Related Information

list_properties Method

next_one Method

Retrieves the next property binding.

IDL Syntax

boolean next_one(out ExtendedNaming::PropertyBinding *pbinding*);

Description

Returns the next property binding in the *pbinding* parameter. This method is used, in standard CORBA fashion, to obtain the next property binding from the extended naming context for which the targeted **PropertyBindingIterator** is associated. Calling programs should check the return value for decision making for further invocations on the iterator. The method returns FALSE if there are no more bindings to obtain.

Intended Usage

This method is intended to be used by client applications. It is not typically overridden.

This method is not defined in the OMG standard.

Parameters

pbinding
The returned **PropertyBinding**.

Return Value

This method returns a Boolean value where FALSE indicates to the client that there are no more property bindings and where TRUE indicates more property bindings exist.

Exceptions

CORBA 1.1 standard exceptions.

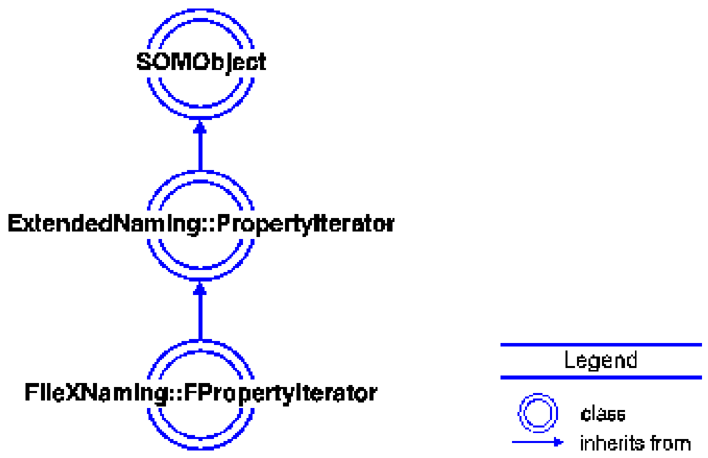
Original Class

ExtendedNaming::PropertyBindingIterator Interface

Related Information

list_properties Method

FileXNaming::FPropertyIterator Class


SOMObject
ExtendedNaming::PropertyIterator
FileXNaming::FPropertyIterator
Legend
class
inherits from

The **FileXNaming::FPropertyIterator** class provides support for extended naming property iteration.

Intended Usage

This class is instantiated and outputted through the **ExtendedNaming::Extended-NamingContext::get_properties** or **ExtendedNaming::ExtendedNaming-Context::get_all_properties** methods if an extended naming context contains more properties than requested.

File Stem

xnamingf

Base Classes

SOMObject Class

Ancestor Classes

ExtendedNaming::PropertyIterator Class

Types

```
typedef struct Property_struct {
        PropertyBinding binding ;
        any value;
} Property;
typedef sequence<Property> PropertyList;
```

New Methods

destroy Method
next_n Method
next_one Method

Overridden Methods

somDefaultInit Method
somDestruct Method

Exceptions

CORBA 1.1 standard exceptions.

destroy Method

Destroys the iterator.

IDL Syntax

```
void destroy();
```

Description

Destroys the iterator. It invokes the **somFree Method** to free the object.

Intended Usage

This method is intended to be used by client applications. It is not typically overridden.

This method is not defined in the OMG standard.

Exceptions

CORBA 1.1 standard exceptions.

Original Class

ExtendedNaming::PropertyIterator Interface

Related Information

get_properties Method
get_all_properties Method

next_n Method

Retrieves a specified maximum number of properties.

IDL Syntax

boolean next_n(
    in unsigned long *how_many*,
    out ExtendedNaming::PropertyList *plist*);

Description

Returns a specified maximum number of properties in the *pl* parameter.This method is used, in standard CORBA fashion, to obtain the next several properties from the extended naming context with which the targeted **PropertyIterator** is associated. Calling programs should check the return value for decision making for further invocations on the iterator. The method returns FALSE if there are no more properties to obtain, indicating to the calling program that it should not invoke the method again.

Intended Usage

This method is intended to be used by client applications. It is not typically overridden.

This method is not defined in the OMG standard.

Parameters

how_many
The maximum number of bindings.

plist
The returned **PropertyList**.

Return Value

This method returns a Boolean value where FALSE indicates to the client that there are no more bindings and where TRUE indicates more bindings exist.

Exceptions

CORBA 1.1 standard exceptions.

Original Class

ExtendedNaming::PropertyIterator Interface

Related Information

get_properties Method
get_all_properties Method

next_one Method

Retrieves the next property.

IDL Syntax

```
boolean next_one(out ExtendedNaming::Property property);
```

Description

Returns the next property in the *property* parameter. This method is used, in standard CORBA fashion, to obtain the next property from the extended naming context for which the targeted **PropertyIterator** is associated with. Calling programs should check the return value for decision making for further invocations on the iterator. The method returns FALSE if there are no more properties to obtain.

Intended Usage

This method is intended to be used by client applications. It is not typically overridden.

This method is not defined in the OMG standard.

Parameters

property
The returned **Property**.

Return Value

This method returns a Boolean value where FALSE indicates to the client that there are no more bindings and where TRUE indicates more bindings exist.

Exceptions

CORBA 1.1 standard exceptions.

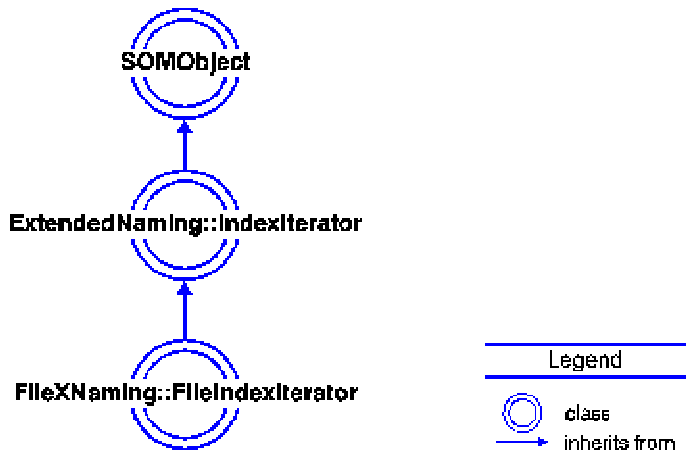
Original Class

ExtendedNaming::PropertyIterator Interface

Related Information

get_properties Method
get_all_properties Method

FileXNaming::FileIndexIterator Class



The **FileXNaming::FileIndexIterator** class provides support for ExtendedNaming index iteration.

Intended Usage

This class is instatiated and returned from the **ExtendedNaming::ExtendedNaming-Context::list_indexes** method if an extended naming context contains more indexes than requested.

File Stem

xnamingf

Base Classes

SOMObject Class

Ancestor Classes

ExtendedNaming::IndexIterator Class

Types

```
typedef struct IndexDescriptor_struct {
        CosNaming::Istring property_name;
        TypeCode property_type;
        unsigned long distance;
} IndexDescriptor;
typedef sequence<IndexDescriptor> IndexDescriptorList;
```

New Methods

destroy Method
next_n Method
next_one Method

Overridden Methods

somDefaultInit Method
somDestruct Method

Exceptions

CORBA 1.1 standard exceptions.

destroy Method

Destroys the iterator.

IDL Syntax

```
void destroy();
```

Description

Destroys the iterator. It invokes the **somFree Method** to free the object.

Intended Usage

This method is intended to be used by client applications. It is not typically overridden.

This method is not defined in the OMG standard.

Exceptions

CORBA 1.1 standard exceptions.

Original Class

ExtendedNaming::IndexIterator Interface

Related Information

remove_index Method

next_n Method

Retrieves a specified maximum number of index descriptors.

IDL Syntax

```
boolean next_n(
        in unsigned long how_many,
        out ExtendedNaming::IndexDescriptorList idxlist);
```

Description

Returns a specified maximum number of bindings.This method is used, in standard CORBA fashion, to obtain the next several index descriptors from the extended naming context for which the targeted **IndexIterator** is associated. Calling programs should check the return value for decision making for further invocations on the iterator. The method returns FALSE if there are no more index descriptors to obtain, indicating to the calling program that it should not invoke the method again.

Intended Usage

This method is intended to be used by client applications. It is not typically overridden.

This method is not defined in the OMG standard.

Parameters

how_many
The maximum number of bindings.

idxlist
The returned **IndexDescriptorList**.

Return Value

This method returns a Boolean value where FALSE indicates to the client that there are no more bindings and where TRUE indicates more bindings exist.

Exceptions

CORBA 1.1 standard exceptions.

Original Class

ExtendedNaming::IndexIterator Interface

Related Information

remove_index Method

next_one Method

Retrieves the next index descriptor.

IDL Syntax

boolean next_one(out ExtendedNaming::IndexDescriptor *idx*);

Description

Returns the next index descriptor in the *p* parameter. This method is used, in standard CORBA fashion, to obtain the next index descriptor from the extended naming context with which the targeted **IndexIterator** is associated. Calling programs should check the return value for decision making for further invocations on the iterator. The method returns FALSE if there are no more index descriptors to obtain.

Intended Usage

This method is intended to be used by client applications. It is not typically overridden.

This method is not defined in the OMG standard.

Parameters

idx
The returned **IndexDescriptor**.

Return Value

This method returns a Boolean value where FALSE indicates to the client that there are no more bindings and where TRUE indicates more bindings exist.

Exceptions

CORBA 1.1 standard exceptions.

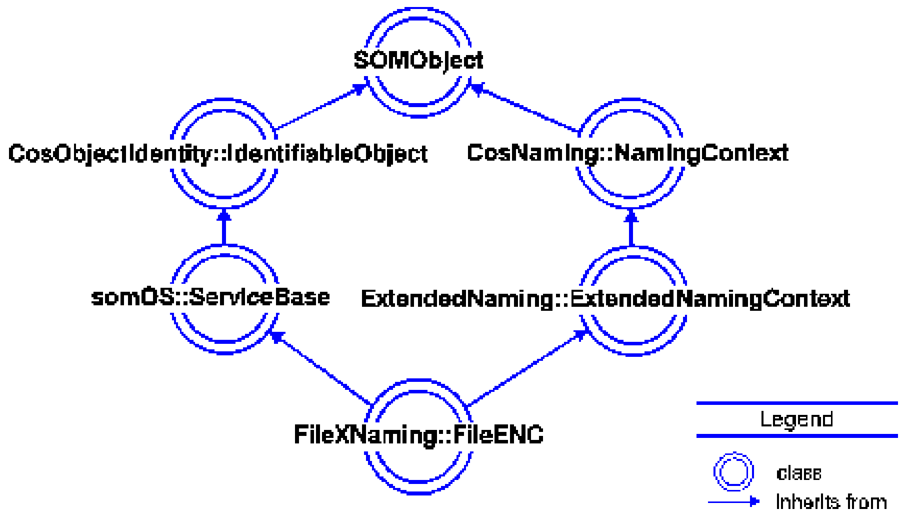
Original Class

ExtendedNaming::IndexIterator Interface

Related Information

remove_index Method

FileXNaming::FileENC Class



The **FileXNaming::FileENC** class provides support for the **CosNaming::NamingContext** and the extensions introduced in the **ExtendedNaming::ExtendedNamingContext** interfaces.

Intended Usage

The **FileXNaming::FileENC** class is the concrete implementation of the **Extended-Naming::ExtendedNamingContext** interface.

File Stem

xnamingf

Base Classes

SOMObject Class

somOS::ServiceBase Class

Ancestor Classes

CosObjectIdentity::IdentifiableObject

CosNaming::NamingContext

ExtendedNaming::ExtendedNamingContext

Types

```
typedef string Istring;

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

enum BindingType {nobject, ncontext};
```

```
struct Binding {
        Name binding_name;
        BindingType binding_type;
};
typedef sequence <Binding> BindingList;

typedef string Constraint;
typedef sequence<CosNaming::Istring> IList;

typedef struct PropertyBinding_struct {
        CosNaming::Istring property_name;
        boolean sharable;
} PropertyBinding;
typedef sequence<PropertyBinding> PropertyBindingList;

typedef struct Property_struct {
        PropertyBinding binding ;
        any value;
} Property;
typedef sequence<Property> PropertyList;

typedef struct IndexDescriptor_struct {
        CosNaming::Istring property_name;
        TypeCode property_type;
        unsigned long distance;
} IndexDescriptor;
typedef sequence<IndexDescriptor> IndexDescriptorList;
```

New Methods

add_index Method
add_properties Method
add_property Method
bind Method
bind_context_with_properties Method
bind_with_properties Method
find_all Method
find_any Method
find_any_name_binding Method
get_all_properties Method
get_features_supported Method
get_properties Method
get_property Method
list Method
list_indexes Method
list_properties Method
rebind_with_properties Method
rebind_context_with_properties Method
remove_all_properties Method
remove_index Method
remove_properties Method
remove_property Method
resolve_with_all_properties Method
resolve_with_properties Method
resolve_with_property Method

_get_allowed_object_types Method
_get_allowed_property_names Method
_get_allowed_property_types Method

Overridden Methods

somDefaultInit Method
somDestruct Method

Exceptions

CORBA 1.1 standard exceptions and the following user exceptions:

- **CosNaming::NamingContext::AlreadyBound** is raised to indicate that an object is already bound to the name. Re-binding operations unbind the name, then rebinds the name without raising this exception.
- **CosNaming::NamingContext::CannotProceed{NamingContext ctx; Name rest_of_name;};** is raised to indicate that the implementation has given up for some reason. The client may be able to continue the operation using the returned naming context.
- **CosNaming::NamingContext::InvalidName** is raised to indicate that the name is invalid. A name with a length of zero is invalid. (This exception may be raised upon further implementation restrictions.)
- **CosNaming::NamingContext::NotFound{NotFoundReason why; Name rest_of_name;};** is raised to indicate that the name does not identify a binding. If a compound name is passed as an argument for the **bind** operation, it traverses multiple contexts. A **NotFound** exception is raised if any of the intermediate contexts cannot be resolved.
- **ExtendedNaming::ExtendedNamingContext::InvalidPropertyName** indicates that the property name is invalid. A property name with length of zero is invalid.
- **ExtendedNaming::ExtendedNamingContext::NotSupported indicates that** the implementation does not support this method.
- **ExtendedNaming::ExtendedNamingContext::ConflictingPropertyName** indicates the property name is in conflict.
- **ExtendedNaming::ExtendedNamingContext::PropertyNotFound{CosNaming Istring property_name;}** indicates that a property was not found.
- **ExtendedNaming::ExtendedNamingContext::IllegalConstraintExpression** indicates that a constraint expression could not be parsed.
- **ExtendedNaming::ExtendedNamingContext::BindingNotFound;** indicates that a requested binding was not found.

add_index Method

Identifies a property to be indexed.

IDL Syntax

void add_index(in ExtendedNaming::IndexDescriptor *idx*);

Description

Identifies a property to be indexed. The index applies to any name-object bindings in the targeted extended naming context or sub-extended naming contexts up to a depth of *distance*, whose property name and property type are specified in *idx*. If *distance* is set to 0 this operation builds an index for only the targeted context. If *distance* is set to something greater than 0, this operation operates recursively on all sub-contexts down to the depth specified. Any properties added later to bindings in the target extended naming context or relevant sub-extended naming contexts of this property name and type are automatically added to the index.

Intended Usage

This method is intended to be used by client applications. It is not typically overridden.

This method is not defined in the OMG standard.

Parameters

idx
The index descriptor to be added.

Return Value

void

Exceptions

CORBA 1.1 standard exceptions and the following user exceptions:

ExtendedNaming::ExtendedNamingContext::NotSupported{}; is raised to indicate that implementation does not support this method.

Original Class

ExtendedNaming::ExtendedNamingContext Interface

add_properties Method

Adds properties to name-object binding.

IDL Syntax

```
void add_properties(
        in CosNaming::Name name,
        in ExtendedNaming::PropertyList props);
```

Description

Adds properties to name-object binding. Adds or updates multiple properties, specified in *props*, associated with a name-object binding specified by *name*, in a target extended naming context. If a property already exists, the property is updated. If a property does not already exist, a new property is associated with the binding (added).

Note: The sharable flag inside a property's **PropertyBinding** is not supported at this time.

Intended Usage

This method is intended to be used by client applications. It is not typically overridden.

This method is not defined in the OMG standard.

Parameters

name
The name of the name-object binding.

props
The **PropertyList** to be added.

Exceptions

CORBA 1.1 standard exceptions and the following user exceptions:

- **CosNaming::NamingContext::NotFound** is raised to indicate that the name does not identify a binding. If a compound name is passed as an argument for the **bind** operation, it traverses multiple contexts. A **NotFound** exception is raised if any of the intermediate contexts cannot be resolved.
- **CosNaming::NamingContext::CannotProceed** is raised to indicate that the implementation has given up for some reason. The client may be able to continue the operation using the returned naming context.
- **CosNaming::NamingContext::InvalidName** is raised to indicate that the name is invalid. A name with a length of zero is invalid. (This exception may be raised upon further implementation restrictions.)
- **ExtendedNaming::ExtendedNamingContext::InvalidPropertyName;** is raised to indicate that the property name is invalid. A property name with a length of zero is invalid.
- **ExtendedNaming::ExtendedNamingContext::NotSupported** is raised to indicate implementation does not support this method.
- **ExtendedNaming::ExtendedNamingContext::ConflictingPropertyName;** is raised to indicate that the property is in conflict.

Original Class

ExtendedNaming::ExtendedNamingContext Interface

add_property Method

Adds a property to name-object binding.

IDL Syntax

```
void add_property(
        in CosNaming::Name name,
        in ExtendedNaming::Property prop);
```

Description

Adds a property to binding. Adds or updates a single property, specified as *prop,* associated with a name-object binding specified by *name*, in a target extended naming context. If the property already exists the property is updated with the specified property, *prop*. If the property does not already exist, then specified property is associated with the binding (added).

Note: The sharable flag inside a property's **PropertyBinding** is not supported at this time.

Intended Usage

This method is intended to be used by client applications. It is not typically overridden.

This method is not defined in the OMG standard.

Parameters

name
The name of the binding.

prop
The **Property** to be added.

Exceptions

CORBA 1.1 standard exceptions and the following user exceptions:

- **CosNaming::NamingContext::NotFound** is raised to indicate that the name does not identify a binding. If a compound name is passed as an argument for the **bind** operation, it traverses multiple contexts. A **NotFound** exception is raised if any of the intermediate contexts cannot be resolved.
- **CosNaming::NamingContext::CannotProceed** is raised to indicate that the implementation has given up for some reason. The client may be able to continue the operation using the returned naming context.
- **CosNaming::NamingContext::InvalidName** is raised to indicate that the name is invalid. A name with a length of zero is invalid. (This exception may be raised upon further implementation restrictions.)
- **ExtendedNaming::ExtendedNamingContext::InvalidPropertyName** is raised to indicate that the property name is invalid. A property name with a length of zero is invalid.
- **ExtendedNaming::ExtendedNamingContext::NotSupported** is raised to indicate implementation does not support this method.
- **ExtendedNaming::ExtendedNamingContext::ConflictingPropertyName** is raised to indicate that the property name is in conflict.

Original Class

ExtendedNaming::ExtendedNamingContext Interface

bind Method

Creates a binding in a naming context.

IDL Syntax

```
void bind(
        in CosNaming::Name name,
        in SOMObject obj);
```

Description

Creates a binding to an object in a naming context. Binding a name and object into a naming context creates a name-object association relative to the target naming context. Once an object is bound, it can be found through the **resolve** operation. Naming contexts that are bound using **bind** do not participate in name resolution when compound names are resolved — **bind_context** should be used to bind naming context objects. This method runs **resolve** to traverse a compound name.

Intended Usage

This method is intended to be used by client applications. It is not typically overridden.

Parameters

name
The name for the binding.

obj
The **SOMObject** to be bound.

Exceptions

CORBA 1.1 standard exceptions and the following user exceptions:

- **CosNaming::NamingContext::NotFound** is raised to indicate that the name does not identify a binding. If a compound name is passed as an argument for the **bind** operation, it traverses multiple contexts. A **NotFound** exception is raised if any of the intermediate contexts cannot be resolved.
- **CosNaming::NamingContext::CannotProceed** is raised to indicate that the implementation has given up for some reason. The client may be able to continue the operation using the returned naming context.
- **CosNaming::NamingContext::InvalidName** is raised to indicate that the name is invalid. A name with a length of zero is invalid. (This exception may be raised upon further implementation restrictions.)
- **CosNaming::NamingContext::AlreadyBound** is raised to indicate that an object is already bound to the name. Re-binding operations unbind the name, then rebinds the name without raising this exception.

Original Class

CosNaming::NamingContext

bind_context Method

Creates a naming context binding.

IDL Syntax

```
void bind_context(
        in CosNaming::Name name,
        in CosNaming::NamingContext naming_context);
```

Description

Creates a naming context binding. Binding a name and a naming context object into a naming context creates a name-object association relative to the target naming context. Naming contexts that are bound using **bind_context** participate in name resolution when compound names are resolved.

Intended Usage

This method is intended to be used by client applications. It is not typically overridden.

Parameters

name
The name for the binding.

naming_context
The naming context object to be bound.

Exceptions

CORBA 1.1 standard exceptions and the following user exceptions:

- **CosNaming::NamingContext::NotFound** is raised to indicate that the name does not identify a binding. If a compound name is passed as an argument for the bind operation, it traverses multiple contexts. A **NotFound** exception is raised if any of the intermediate contexts cannot be resolved.
- **CosNaming::NamingContext::CannotProceed** is raised to indicate that the implementation has given up for some reason. The client may be able to continue the operation using the returned naming context.
- **CosNaming::NamingContext::InvalidName** is raised to indicate that the name is invalid. A name with a length of zero is invalid. (This exception may be raised upon further implementation restrictions.)
- **CosNaming::NamingContext::AlreadyBound** is raised to indicate that an object is already bound to the name. Re-binding operations unbind the name, then rebinds the name without raising this exception.

Original Class

CosNaming::NamingContext

bind_context_with_properties Method

Creates a naming context object binding and associate properties.

IDL Syntax

```
void bind_context_with_properties(
        in CosNaming::Name name,
        in ExtendedNaming::ExtendedNamingContext obj,
        in ExtendedNaming::PropertyList props);
```

Description

Binds a naming context with properties. Operates just like the **CosNaming::-NamingContext::bind_context** operation in that it binds the specified naming context into the target extended naming context. In addition, it defines properties associated with the binding in *props*. Naming contexts bound using this operation participate in name resolution when compound names are resolved.

Intended Usage

This method is intended to be used by client applications. It is not typically overridden.

This method is not defined in the OMG standard.

Parameters

name
The name of the binding.

obj
The naming context object to be bound.

props
The **PropertyList** to associated with the binding.

Exceptions

CORBA 1.1 standard exceptions and the following user exceptions:

- **CosNaming::NamingContext::NotFound** is raised to indicate that the name does not identify a binding. If a compound name is passed as an argument for the **bind** operation, it traverses multiple contexts. A **NotFound** exception is raised if any of the intermediate contexts cannot be resolved.
- **CosNaming::NamingContext::CannotProceed** is raised to indicate that the implementation has given up for some reason. The client continues the operation using the returned naming context.
- **CosNaming::NamingContext::InvalidName** is raised to indicate that the name is invalid. A name with a length of zero is invalid. (This exception may be raised upon further implementation restrictions.)
- **CosNaming::NamingContext::AlreadyBound** is raised to indicate that an object is already bound to the name. Rebinding operations unbind the name, then rebind the name without raising this exception.
- **ExtendedNaming::ExtendedNamingContext::InvalidPropertyName** is raised to indicate that the property name is invalid. A property name with a length of zero is invalid.
- **ExtendedNaming::ExtendedNamingContext::NotSupported** is raised to indicate implementation does not support this method.
- **ExtendedNaming::ExtendedNamingContext::ConflictingPropertyName** is raised to indicate that the property name is in conflict.

Original Class

ExtendedNaming::ExtendedNamingContext Interface

bind_new_context Method

Creates a new naming context in the same server as the target naming context on which the operation was invoked and binds it to a supplied name.

IDL Syntax

CosNaming::NamingContext bind_new_context(in CosNaming::Name *name*);

Description

Creates a new naming context in the same process as the target naming context on which the operation was invoked and binds it to a supplied name. The new naming context is implemented in the same naming server as the target naming context in which it was bound.

Intended Usage

This method is intended to be used by client applications. It is not typically overridden.

Parameters

name
The name for the naming context object binding.

Return Value

This operation returns a new **CosNaming::NamingContext** bound to the supplied name.

Exceptions

CORBA 1.1 standard exceptions and the following user exceptions:

- **CosNaming::NamingContext::NotFound** is raised to indicate that the name does not identify a binding. If a compound name is passed as an argument for the **bind** operation, it traverses multiple contexts. A **NotFound** exception is raised if any of the intermediate contexts cannot be resolved.
- **CosNaming::NamingContext::CannotProceed** is raised to indicate that the implementation has given up for some reason. The client may be able to continue the operation using the returned naming context.
- **CosNaming::NamingContext::InvalidName** is raised to indicate that the name is invalid. A name with a length of zero is invalid. (This exception may be raised upon further implementation restrictions.)
- **CosNaming::NamingContext::AlreadyBound** is raised to indicate that an object is already bound to the name.

Original Class

CosNaming::NamingContext

bind_with_properties Method

Creates a binding and associates properties to the binding.

IDL Syntax

```
void bind_with_properties(
        in CosNaming::Name name,
        in SOMObject obj,
        in ExtendedNaming::PropertyList plist);
```

Description

Binds an object with properties. Operates just like the **CosNaming::NamingContext::bind** operation in that it binds the specified *obj* into the target extended naming context. In addition, it defines properties to be associated with the binding in *prop* (combination of **add_properties** and **bind**). A property is replaced if it already exists.

Intended Usage

This method is intended to be used by client applications. It is not typically overridden.

This method is not defined in the OMG standard.

Parameters

name
The name of the name-object binding.

obj
The **SOMObject** to be bound.

plist
The **PropertyList** to associated with the binding.

Exceptions

CORBA 1.1 standard exceptions and the following user exceptions:

- **CosNaming::NamingContext::NotFound** is raised to indicate that the name does not identify a binding. If a compound name is passed as an argument for the **bind** operation, it traverses multiple contexts. A **NotFound** exception is raised if any of the intermediate contexts cannot be resolved.
- **CosNaming::NamingContext::CannotProceed** is raised to indicate that the implementation has given up for some reason. The client may be able to continue the operation using the returned naming context.
- **CosNaming::NamingContext::InvalidName** is raised to indicate that the name is invalid. A name with a length of zero is invalid. (This exception may be raised upon further implementation restrictions.)
- **CosNaming::NamingContext::AlreadyBound** is raised to indicate that an object is already bound to the name. Rebinding operations unbind the name, then rebind the name without raising this exception.
- **ExtendedNaming::ExtendedNamingContext::InvalidPropertyName** is raised to indicate that the property name is invalid. A property name with a length of zero is invalid.
- **ExtendedNaming::ExtendedNamingContext::NotSupported** is raised to indicate implementation does not support this method.
- **ExtendedNaming::ExtendedNamingContext::ConflictingPropertyName** is raised to indicate that the property name is in conflict.

Original Class

ExtendedNaming::ExtendedNamingContext Interface

destroy Method

Destroys a naming context.

IDL Syntax

```
void destroy();
```

Description

Destroys the naming context if the context is empty.

The naming context cannot contain bindings for this operation to succeed. It is the responsibility of the client to ensure that all bindings have been removed from the naming context before invoking this method . Use the **unbind** method to remove any bindings in the naming context; for more information, refer to the “unbind Method” on page 118.

This method invokes the **somFree Method** to free the object.

Intended Usage

This method is intended to be used by client applications. It is not typically overridden.

Exceptions

CosNaming::NamingContext::NotEmpty is raised is raised if the naming context contains any bindings.

Original Class

CosNaming::NamingContext

find_all Method

Retrieves all bindings satisfying property search constraints.

IDL Syntax

```
void find_all(
        in ExtendedNaming::Constraint constraint,
        in unsigned long distance,
        in unsigned long how_many,
        out CosNaming::BindingList blist,
        out CosNaming::BindingIterator biterator);
```

Description

Outputs each **CosNaming::Binding** that satisfies the property search constraint specified in *constraint*. It searches up to a depth of *distance* for all bindings that satisfy the given constraint and puts them into the binding list, *blist*. If *distance* is set to 0, this operation searches only the targeted context. Up to *how_many* name-object bindings are placed into the binding list. If more than *how_many* objects are found to satisfy the constraint, the remaining name-object bindings are placed into the binding iterator, *biterator*.

Intended Usage

This method is intended to be used by client applications. It is not typically overridden.

This method is not defined in the OMG standard.

Parameters

constraint
The search constraint. This constraint is a string that must be formed in conformance with the grammar specified in **Appendix A, BNF for Naming Constraint Language**.

distance
The search depth.

how_many
The maximum number of **Bindings** to put into *blist*.

blist
The outputted **BindingList.**

biterator
The outputted **BindingIterator**.

Exceptions

CORBA 1.1 standard exceptions and the following user exceptions:

- **CosNaming::NamingContext::CannotProceed** is raised to indicate that the implementation has given up for some reason. The client may be able to continue the operation using the returned naming context.
- **ExtendedNaming::ExtendedNamingContext::NotSupported** is raised to indicate implementation does not support this method.
- **ExtendedNaming::ExtendedNamingContext::IllegalConstraintExpression** is raised to indicate that a constraint expression could not be parsed.
- **ExtendedNaming::ExtendedNamingContext::BindingNotFound** is raised to indicate that the search failed.

Original Class

ExtendedNaming::ExtendedNamingContext Interface

Related Information

FileXNaming::FileBindingIterator Class

find_any Method

Retrieves the first bound object that satisfies the given search constraint.

IDL Syntax

```
SOMObject find_any(
        in ExtendedNaming::Constraint constraint,
        in unsigned long distance);
```

Description

Returns the first bound **SOMObject** satisfying the property search constraint specified in *constraint*. The returned **SOMObject** contains properties that satisfy the constraint. It searches up to a depth of *distance* for a binding that satisfies the given constraint. If *distance* is set to 0, this operation searches only the targeted context.

Intended Usage

This method is intended to be used by client applications. It is not typically overridden.

This method is not defined in the OMG standard.

Parameters

constraint
The search constraint. This constraint is a string that must be formed in conformance with the grammar specified in **Appendix A, BNF for Naming Constraint Language**.

distance
The search depth in the name graph.

Return Value

A **SOMObject** is returned, which satisfies the property search constraint.

Exceptions

CORBA 1.1 standard exceptions and the following user exceptions:

- **CosNaming::NamingContext::CannotProceed** is raised to indicate that the implementation has given up for some reason. The client may be able to continue the operation using the returned naming context.
- **ExtendedNaming::ExtendedNamingContext::NotSupported** is raised to indicate that implementation does not support this method.
- **ExtendedNaming::ExtendedNamingContext::IllegalConstraintExpression** is raised to indicate that a constraint expression could not be parsed.
- **ExtendedNaming::ExtendedNamingContext::BindingNotFound** is raised to indicate that the search failed.

Original Class

ExtendedNaming::ExtendedNamingContext Interface

find_any_name_binding Method

Retrieves a name-object binding satisfying property search constraints.

IDL Syntax

```
void find_any_name_binding(
        in ExtendedNaming::Constraint constraint,
        in unsigned long distance,
        out CosNaming::Binding binding);
```

Description

Returns a **CosNaming::Binding** satisfying the property search constraint specified in *constraint*. The retrieved **CosName::Binding** is any name-object binding that contains properties that satisfy *constraint*. It searches up to a depth of *distance* for a binding that satisfies the given constraint. If *distance* is set to 0, this operation searches only the targeted context.

Intended Usage

This method is intended to be used by client applications. It is not typically overridden.

This method is not defined in the OMG standard.

Parameters

constraint
The search constraint. This constraint is a string that must be formed in conformance with the grammar specified in **Appendix A, BNF for Naming Constraint Language**.

distance
The search depth in the Naming Service graph.

binding
The outputted **Binding**.

Exceptions

CORBA 1.1 standard exceptions and the following user exceptions:

- **CosNaming::NamingContext::CannotProceed** is raised to indicate that the implementation has given up for some reason. The client may be able to continue the operation using the returned naming context.
- **ExtendedNaming::ExtendedNamingContext::NotSupported** is raised to indicate implementation does not support this method.
- **ExtendedNaming::ExtendedNamingContext::IllegalConstraintExpression** is raised to indicate that a constraint expression could not be parsed.
- **ExtendedNaming::ExtendedNamingContext::BindingNotFound** is raised to indicate that a requested binding was not found.

Original Class

ExtendedNaming::ExtendedNamingContext Interface

get_all_properties Method

Retrieves all properties for a name-object binding.

IDL Syntax

```
void get_all_properties(
        in CosNaming::Name name,
        in unsigned long how_many,
        out ExtendedNaming::PropertyList props,
        out ExtendedNaming::PropertyIterator rest);
```

Description

Returns all properties for a name-object binding. Returns the properties that are associated with the name-object binding, specified by *name*, in the target extended naming context. If the name-object binding contains more than *how_many* properties, then the remaining properties are put in *rest*. Clients can iterate through the interator to retrieve the remaining properties.

Intended Usage

This method is intended to be used by client applications. It is not typically overridden.

This method is not defined in the OMG standard.

Parameters

name
The name of the name-object binding.

how_many
The maximum number of properties to put into *props*.

props
The returned properties.

rest
The returned **PropertyIterator**.

Exceptions

CORBA 1.1 standard exceptions and the following user exceptions:

- **CosNaming::NamingContext::NotFound** is raised to indicate that the name does not identify a binding. If a compound name is passed as an argument for the **bind** operation, it traverses multiple contexts. A **NotFound** exception is raised if any of the intermediate contexts cannot be resolved.
- **CosNaming::NamingContext::CannotProceed** is raised to indicate that the implementation has given up for some reason. The client may be able to continue the operation using the returned naming context.
- **CosNaming::NamingContext::InvalidName** is raised to indicate that the name is invalid. A name with a length of zero is invalid. (This exception may be raised upon further implementation restrictions.)
- **ExtendedNaming::ExtendedNamingContext::InvalidPropertyName** is raised to indicate that the property name is invalid. A property name with a length of zero is invalid.

Original Class

ExtendedNaming::ExtendedNamingContext Interface

Related Information

ExtendedNaming::PropertyIterator Interface

get_features_supported Method

Retrieves the supported features.

IDL Syntax

unsigned short get_features_supported();

Description

Returns the supported features of an extended naming context. Gets a bit vector that specifies the features this extended naming context implementation supports: 0 properties, 1 shared property, 2 searching, 3 indexing, 4 restrictions on object types, 5 restrictions on property types, 6 restrictions on property names, 7 - 15 not used.

Intended Usage

This method is intended to be used by client applications. It is not typically overridden.

This method is not defined in the OMG standard.

Return Value

An unsigned short bit vector is returned indicating supported features.

The return value contains 16 bits ordered as follows:

Individual bits are set TRUE if:

Bit	Description
0	The naming context implements support for properties
1	The naming context implements support for property-sharing
2	The naming context implements support for searching on properties
3	The naming context implements support for creating indexes that are applied during search operations
4	The naming context implements restrictions on the types of objects that can be bound — see the **allowed_object_types** attribute to determine which object types can be bound
5	The naming context implements restrictions on what property types can be created — see the **allowed_property_types** attribute to determine which property types can be used
6	The naming context implements restrictions on what property names can be used — see the **allowed_property_names** attribute to determine which property names can be used
7-15	unused

Exceptions

CORBA 1.1 standard exceptions.

Original Class

ExtendedNaming::ExtendedNamingContext Interface

get_properties Method

Retrieves property values for the specified property name.

IDL Syntax

```
void get_properties(
        in CosNaming::Name name,
        in unsigned long how_many,
        in ExtendedNaming::IList inames,
        out ExtendedNaming::PropertyList props,
        out ExtendedNaming::PropertyIterator rest);
```

Description

Returns a set of properties for a name-object binding. Returns the properties, with their property names specified as *inames*, associated with the name-object binding specified by *name* in the target extended naming context. If the name-object binding contains more than *how_many* properties, the remaining properties are put in *rest*. Clients can iterate through the iterator to retrieve the remaining properties.

Intended Usage

This method is intended to be used by client applications. It is not typically overridden.

This method is not defined in the OMG standard.

Parameters

name
The name of the name-object binding.

how_many
The maximum number of properties to put in *props*.

inames
The list of property names to be retrieved.

props
The returned properties.

rest
The returned **PropertyIterator**.

Exceptions

CORBA 1.1 standard exceptions and the following user exceptions:

- **CosNaming::NamingContext::NotFound** is raised to indicate that the name does not identify a binding. If a compound name is passed as an argument for the **bind** operation, it traverses multiple contexts. A **NotFound** exception is raised if any of the intermediate contexts cannot be resolved.
- **CosNaming::NamingContext::CannotProceed** is raised to indicate that the implementation has given up for some reason. The client may be able to continue the operation using the returned naming context.
- **CosNaming::NamingContext::InvalidName** is raised to indicate that the name is invalid. A name with a length of zero is invalid. (This exception may be raised upon further implementation restrictions.)
- **ExtendedNaming::ExtendedNamingContext::InvalidPropertyName** is raised to indicate that the property name is invalid. A property name with a length of zero is invalid.

- **ExtendedNaming::ExtendedNamingContext::PropertyNotFound{CosNaming Istring property_name;};** is raised to indicate that a property was not found.

Original Class

ExtendedNaming::ExtendedNamingContext Interface

Related Information

ExtendedNaming::PropertyIterator Interface, which can be instantiated by this method.

get_property Method

Retrieves the value of the specified property name.

IDL Syntax

```
void get_property(
        in CosNaming::Name name,
        in CosNaming::Istring pname,
        out ExtendedNaming::Property prop);
```

Description

Returns a property (value of the property) for a name-object binding. Returns the property, with its property name specified as *pname*, associated with the name-object binding specified by *name* in the target extended naming context.

Intended Usage

This method is intended to be used by client applications. It is not typically overridden.

This method is not defined in the OMG standard.

Parameters

name
The name of the name-object binding.

pname
The property name to be returned.

prop
The returned property.

Exceptions

CORBA 1.1 standard exceptions and the following user exceptions:

- **CosNaming::NamingContext::NotFound** is raised to indicate that the name does not identify a binding. If a compound name is passed as an argument for the **bind** operation, it traverses multiple contexts. A **NotFound** exception is raised if any of the intermediate contexts cannot be resolved.
- **CosNaming::NamingContext::CannotProceed** is raised to indicate that the implementation has given up for some reason. The client may be able to continue the operation using the returned naming context.
- **CosNaming::NamingContext::InvalidName** is raised to indicate that the name is invalid. A name with a length of zero is invalid. (This exception may be raised upon further implementation restrictions.)
- **ExtendedNaming::ExtendedNamingContext::InvalidPropertyName** is raised to indicate that the property name is invalid. A property name with a length of zero is invalid.
- **ExtendedNaming::ExtendedNamingContext::PropertyNotFound{CosNaming Istring property_name;};** is raised to indicate that a property was not found.

Original Class

ExtendedNaming::ExtendedNamingContext Interface

list Method

Retrieves all of the bindings in a naming context.

IDL Syntax

```
void list(
        in unsigned long how_many,
        out CosNaming::BindingList blist,
        out CosNaming::BindingIterator biterator);
```

Description

Returns all of the bindings in a naming context. Returns at most *how_many* number of bindings in *blist*. If the naming context contains additional bindings, a **BindingIterator** is returned, and the calling program can iterate through the remaining bindings. If the naming context does not contain additional bindings, the **BindingIterator** is a NIL object reference.

Intended Usage

This method is intended to be used by client applications. It is not typically overridden.

Parameters

how_many
The maximum number bindings to install into the **BindingList**.

blist
The returned **BindingList**.

biterator
The returned **BindingIterator**.

Exceptions

CORBA 1.1 standard exceptions.

Original Class

CosNaming::NamingContext

Related Information

FileXNaming::FPropertyBindingIterator Class

list_indexes Method

Retrieves all defined indexes.

IDL Syntax

```
void list_indexes(
        in unsigned long how_many,
        out Extended::Naming IndexDescriptorList idxlist,
        out IndexIterator rest);
```

Description

Returns all indexes defined in the target extended naming context. If any bindings in the target extended naming context have properties that are part of indexes in a parent context, those indexes are not listed. Up to *how_many* indexes are placed into the *idxlist*. If more than *how_many* indexes are found, the remaining indexes are put into the *rest*.

Intended Usage

This method is intended to be used by client applications. It is not typically overridden.

This method is not defined in the OMG standard.

Parameters

how_many
The maximum number of indexes to return.

idxlist
The returned **IndexDescriptorList.**

rest
The returned **IndexIterator**.

Exceptions

CORBA 1.1 standard exceptions and the following user exceptions:

ExtendedNaming::ExtendedNamingContext::NotSupported is raised to indicate that implementation does not support this method.

Original Class

ExtendedNaming::ExtendedNamingContext Interface

Related Information

ExtendedNaming::IndexIterator Interface

list_properties Method

Retrieves all **PropertyBindings** for a name-object binding.

IDL Syntax

```
void list_properties(
        in CosNaming::Name name,
        in unsigned long how_many,
        out ExtendedNaming::PropertyBindingList pblist,
        out ExtendedNaming::PropertyBindingIterator rest);
```

Description

Returns all **PropertyBindings** for a name-object binding. Returns all of the **PropertyBindings** (a structural part of an **ExtendedNaming::Property**) that are associated with a name-object binding specified by *name,* in the target extended naming context. If the name-object binding contains more than *how_many* **PropertyBindings**, the remaining **PropertyBindings** are put in *rest*.

Intended Usage

This method is intended to be used by client applications. It is not typically overridden.

This method is not defined in the OMG standard.

Parameters

name
The name of the name-object binding.

how_many
The maximum number of **PropertyBindings** to return.

pblist
The returned **PropertyBindingList**.

rest
The returned **PropertyBindingIterator**.

Exceptions

CORBA 1.1 standard exceptions and the following user exceptions:

- **CosNaming::NamingContext::NotFound** is raised to indicate that the name does not identify a binding. If a compound name is passed as an argument for the **bind** operation, it traverses multiple contexts. A **NotFound** exception is raised if any of the intermediate contexts cannot be resolved.
- **CosNaming::NamingContext::CannotProceed** is raised to indicate that the implementation has given up for some reason. The client may be able to continue the operation using the returned naming context.
- **CosNaming::NamingContext::InvalidName** is raised to indicate that the name is invalid. A name with a length of zero is invalid. (This exception may be raised upon further implementation restrictions.)

Original Class

ExtendedNaming::ExtendedNamingContext Interface

Related Information

FileXNaming::FPropertyBindingIterator Class

new_context Method

Creates an unbound new naming context in the same process as the target naming context on which the operation was invoked.

IDL Syntax

CosNaming::NamingContext new_context();

Description

Creates an unbound new naming context in the same process as the target naming context on which the operation was invoked. The new naming context is not bound to any name.

See **bind_new_context Method** on page 84..

Intended Usage

This method is intended to be used by client applications. It is not typically overridden.

Return Value

This operation returns a **FileXNaming::FileENC** (derived from **CosNaming::Naming-Context**) that is unbound.

Exceptions

CORBA 1.1 standard exceptions.

Original Class

CosNaming::NamingContext

rebind Method

Recreates a binding in a naming context even if the name is already bound in the naming context.

IDL Syntax

```
void rebind(
        in CosNaming::Name name,
        in SOMObject obj);
```

Description

Recreates a name binding in a naming context, even if the name is already bound in the naming context. Rebinding a name and object into a naming context recreates a name-object association relative to the target naming context. Naming contexts that are bound using **rebind** do not participate in name resolution process when compound names are resolved.

If an object is already bound with the same name, the bound object is replaced by the passed argument *obj*. If the name-object binding does not exist, the **rebind** method behaves like the **bind** method.

Clients can use the **rebind** method to replace an existing binding. They can use this method instead of the **unbind** and **bind** methods.

Intended Usage

This method is intended to be used by client applications. It is not typically overridden.

Parameters

name
The name to be re-bound.

obj
The **SOMObject** to be re-bound.

Exceptions

CORBA 1.1 standard exceptions and the following user exceptions:

- **CosNaming::NamingContext::NotFound** is raised to indicate that the name does not identify a binding. If a compound name is passed as an argument for the bind operation, it traverses multiple contexts. A **NotFound** exception is raised if any of the intermediate contexts cannot be resolved.
- **CosNaming::NamingContext::CannotProceed** is raised to indicate that the implementation has given up for some reason. The client may be able to continue the operation using the returned naming context.
- **CosNaming::NamingContext::InvalidName** is raised to indicate that the name is invalid. A name with a length of zero is invalid. (This exception may be raised upon further implementation restrictions.)

Original Class

CosNaming::NamingContext

rebind_context Method

Recreates a binding to a naming context, even if the *name* is already bound in the naming context.

IDL Syntax

void rebind_context(
in CosNaming::Name *name*,
in CosNaming::NamingContext *naming_context*);

Description

Recreates a binding to a naming context, even if the *name* is already bound in the naming context. Re-binding a name and a naming context object into a naming context recreates a name-object association relative to the target naming context. Naming contexts that are bound using **rebind_context** participate in name resolution when compound names are resolved.

This method is used to bind or replace a subcontext. If a context is already bound in a context, the **bind** operation raises the **AlreadyBound** exception. However, the **rebind** method replaces the bound object with the passed object.

Intended Usage

This method is intended to be used by client applications. It is not typically overridden.

Parameters

name
The name to be re-bound.

naming_context
The **NamingContext** object to be re-bound to the name.

Exceptions

CORBA 1.1 standard exceptions and the following user exceptions:

- **CosNaming::NamingContext::NotFound** is raised to indicate that the name cannot be resolved into a naming context to perform binding. If a compound name is passed as an argument for the bind operation, it traverses multiple contexts. A **NotFound** exception is raised if any of the intermediate contexts cannot be resolved.
- **CosNaming::NamingContext::CannotProceed** is raised to indicate that the implementation has given up for some reason. The client may be able to continue the operation using the returned naming context.
- **CosNaming::NamingContext::InvalidName** is raised to indicate that the name is invalid. A name with a length of zero is invalid. (This exception may be raised upon further implementation restrictions.)

Original Class

CosNaming::NamingContext

rebind_context_with_properties Method

Recreates a naming context object binding and associates properties.

IDL Syntax

void rebind_context_with_properties(
in CosNaming::Name *name*,
in ExtendedNaming::ExtendedNamingContext *obj*,
in ExtendedNaming::PropertyList *props*);

Description

Rebinds a naming context with properties. Operates just like the **CosNaming::-NamingContext::rebind_context** operation in that it rebinds the specified naming context into the target extended naming context. In addition, it defines the properties in **PropertyList** *props* to be associated with the binding. If a property is already associated with the binding, it replaces the existing property with the new property. If the property is not already associated with the binding, a new property is associated. Existing properties associated with the binding that are not specified in *props* remain intact. Naming contexts bound using this operation participate in name resolution when compound names are resolved.

Intended Usage

This method is intended to be used by client applications. It is not typically overridden.

This method is not defined in the OMG standard.

Parameters

name
The name of the binding.

obj
The naming context to be bound.

props
The **PropertyList** to associated with the binding.

Exceptions

CORBA 1.1 standard exceptions and the following user exceptions:

- **ExtendedNaming::ExtendedNamingContext::InvalidPropertyName** is raised to indicate that the property name is invalid. A property name with a length of zero is invalid.
- **ExtendedNaming::ExtendedNamingContext::NotSupported** is raised to indicate that implementation does not support this method.
- **ExtendedNaming::ExtendedNamingContext::ConflictingPropertyName** is raised to indicate that the property name is in conflict.
- **ExtendedNaming::ExtendedNamingContext::PropertyNotFound{CosNaming Istring property_name;};** is raised to indicate that a property was not found.
- **ExtendedNaming::ExtendedNamingContext::IllegalConstraintExpression** is raised to indicate that a constraint expression could not be parsed.
- **ExtendedNaming::ExtendedNamingContext::BindingNotFound** is raised to indicate that a requested binding was not found.

Original Class

ExtendedNaming::ExtendedNamingContext Interface

rebind_with_properties Method

Recreates a name-object binding and associate properties.

IDL Syntax

void rebind_with_properties(
    in CosNaming::Name *name*,
    in SOMObject *obj*,
    in ExtendedNaming::PropertyList *props*);

Description

Rebinds an object with properties. Operates just like the **CosNaming::NamingContext::-rebind** in that the specified **SOMObject** *obj* is rebound into the target extended naming context. In addition, it defines the properties in *prop* to be associated with the binding. If a property is already associated with the binding, it replaces the existing property with the new property. If the property is not already associated with the binding, a new property is then associated. Existing properties associated with the binding that are not specified in *prop* remain intact.

Intended Usage

This method is intended to be used by client applications. It is not typically overridden.

This method is not defined in the OMG standard.

Parameters

name
The name of the name-object binding for rebinding.

obj
The **SOMObject** to be bound.

props
The **PropertyList** to associated with the binding.

Exceptions

CORBA 1.1 standard exceptions and the following user exceptions:

- **CosNaming::NamingContext::NotFound** is raised to indicate that the name does not identify a binding. If a compound name is passed as an argument for the **bind** operation, it traverses multiple contexts. A **NotFound** exception is raised if any of the intermediate contexts cannot be resolved.
- **CosNaming::NamingContext::InvalidName** is raised to indicate that the name is invalid. A name with a length of zero is invalid. (This exception may be raised upon further implementation restrictions.)
- **ExtendedNaming::ExtendedNamingContext::InvalidPropertyName** is raised to indicate that the property name is invalid. A property name with a length of zero is invalid.
- **ExtendedNaming::ExtendedNamingContext::NotSupported** is raised to indicate that implementation does not support this method.
- **ExtendedNaming::ExtendedNamingContext::ConflictingPropertyName** is raised to indicate that the property name is in conflict.

Original Class

ExtendedNaming::ExtendedNamingContext Interface

remove_all_properties Method

Removes all properties associated with name-object binding.

IDL Syntax

void remove_all_properties(in CosNaming::Name *name*);

Description

Removes all properties associated with name-object binding. Resolves *name* in the target extended naming context and removes all properties associated with the binding.

Intended Usage

This method is intended to be used by client applications. It is not typically overridden.

This method is not defined in the OMG standard.

Parameters

name
The name of the name-object binding.

Exceptions

CORBA 1.1 standard exceptions and the following user exceptions:

- **CosNaming::NamingContext::NotFound** is raised to indicate that the name does not identify a binding. If a compound name is passed as an argument for the **bind** operation, it traverses multiple contexts. A **NotFound** exception is raised if any of the intermediate contexts cannot be resolved.
- **CosNaming::NamingContext::CannotProceed** is raised to indicate that the implementation has given up for some reason. The client may be able to continue the operation using the returned naming context.
- **CosNaming::NamingContext::InvalidName** is raised to indicate that the name is invalid. A name with a length of zero is invalid. (This exception may be raised upon further implementation restrictions.)
- **ExtendedNaming::ExtendedNamingContext::NotSupported** is raised to indicate implementation does not support this method.

Original Class

ExtendedNaming::ExtendedNamingContext Interface

remove_index Method

Removes a specified index.

IDL Syntax

void remove_index(in ExtendedNaming::IndexDescriptor *idx*);

Description

Removes a specified index from the target extended naming context. The *distance* is ignored in *idx*.

Intended Usage

This method is intended to be used by client applications. It is not typically overridden.

This method is not defined in the OMG standard.

Parameters

idx
The index to be removed.

Exceptions

CORBA 1.1 standard exceptions and the following user exceptions:

- **CosNaming::NamingContext::NotFound** is raised to indicate that the name does not identify a binding. If a compound name is passed as an argument for the **bind** operation, it traverses multiple contexts. A **NotFound** exception is raised if any of the intermediate contexts cannot be resolved.
- **CosNaming::NamingContext::CannotProceed** is raised to indicate that the implementation has given up for some reason. The client may be able to continue the operation using the returned naming context.
- **ExtendedNaming::ExtendedNamingContext::InvalidPropertyName** is raised to indicate that the property name is invalid. A property name with a length of zero is invalid.

Original Class

ExtendedNaming::ExtendedNamingContext Interface

remove_properties Method

Removes a set of properties associated with name-object binding.

IDL Syntax

```
void remove_properties(
        in CosNaming::Name name,
        in ExtendedNaming::IList plist);
```

Description

Removes a set of properties associated with name-object binding. Resolves *name* in the target extended naming context and removes the properties whose property names are specified by *plist*.

Intended Usage

This method is intended to be used by client applications. It is not typically overridden.

This method is not defined in the OMG standard.

Parameters

name
The name of the name-object binding.

plist
A list of property names for removal.

Exceptions

CORBA 1.1 standard exceptions and the following user exceptions:

- **CosNaming::NamingContext::NotFound{NotFoundReason why; Name rest_of_name;}** is raised to indicate that the name does not identify a binding. If a compound name is passed as an argument for the **bind** operation, it traverses multiple contexts. A **NotFound** exception is raised if any of the intermediate contexts cannot be resolved.
- **CosNaming::NamingContext::CannotProceed** is raised to indicate that the implementation has given up for some reason. The client may be able to continue the operation using the returned naming context.
- **CosNaming::NamingContext::InvalidName** is raised to indicate that the name is invalid. A name with a length of zero is invalid. (This exception may be raised upon further implementation restrictions.)
- **ExtendedNaming::ExtendedNamingContext::InvalidPropertyName** is raised to indicate that the property name is invalid. A property name with a length of zero is invalid.
- **ExtendedNaming::ExtendedNamingContext::PropertyNotFound{CosNaming Istring property_name;};** is raised to indicate that a property was not found.
- **ExtendedNaming::ExtendedNamingContext::NotSupported** is raised to indicate implementation does not support this method.

Original Class

ExtendedNaming::ExtendedNamingContext Interface

remove_property Method

Removes a property associated with name-object binding.

IDL Syntax

```
void remove_property(
        in CosNaming::Name name,
        in CosNaming::Istring prop);
```

Description

Removes a property associated with name-object binding. Resolves *name* in the target extended naming context and removes the property whose property name is specified by *prop*.

Intended Usage

This method is intended to be used by client applications. It is not typically overridden.

This method is not defined in the OMG standard.

Parameters

name
The name of the name-object binding.

prop
The property name.

Exceptions

CORBA 1.1 standard exceptions and the following user exceptions:

- **CosNaming::NamingContext::NotFound** is raised to indicate that the name does not identify a binding. If a compound name is passed as an argument for the **bind** operation, it traverses multiple contexts. A **NotFound** exception is raised if any of the intermediate contexts cannot be resolved.
- **CosNaming::NamingContext::CannotProceed** is raised to indicate that the implementation has given up for some reason. The client may be able to continue the operation using the returned naming context.
- **CosNaming::NamingContext::InvalidName** is raised to indicate that the name is invalid. A name with a length of zero is invalid. (This exception may be raised upon further implementation restrictions.)
- **ExtendedNaming::ExtendedNamingContext::InvalidPropertyName** is raised to indicate that the property name is invalid. A property name with a length of zero is invalid.
- **ExtendedNaming::ExtendedNamingContext::PropertyNotFound{CosNaming Istring property_name;};** is raised to indicate that a property was not found.
- **ExtendedNaming::ExtendedNamingContext::NotSupported** is raised to indicate that implementation does not support this method.

Original Class

ExtendedNaming::ExtendedNamingContext Interface

resolve Method

Retrieves a **SOMObject** bound to a name.

IDL Syntax

SOMObject resolve(in CosNaming::Name *name*);

Description

Retrieves the object bound to name n in the target naming context. Because names can be compound, name resolution can traverse multiple naming contexts.The given name must exactly match the bound name. The Naming Service does not return the type of object. Clients are responsible for narrowing the resolved object to the appropriate type. Clients typically cast the returned object to a more specialized interface.

Intended Usage

This method is intended to be used by client applications. It is not typically overridden.

Parameters

name
The name for the name-object binding.

Return Value

This operation returns a **SOMObject** bound to the supplied *name*.

Exceptions

CORBA 1.1 standard exceptions and the following user exceptions:

CosNaming::NamingContext::NotFound is raised to indicate that the name does not identify a binding. If a compound name is passed as an argument for the bind operation, it traverses multiple contexts. A **NotFound** exception is raised if any of the intermediate contexts cannot be resolved.

CosNaming::NamingContext::CannotProceed is raised to indicate that the implementation has given up for some reason. The client may be able to continue the operation using the returned naming context.

CosNaming::NamingContext::InvalidName is raised to indicate that the name is invalid. A name with a length of zero is invalid. (This exception may be raised upon further implementation restrictions.)

Original Class

CosNaming::NamingContext

resolve_with_all_properties Method

Resolves a name-object binding (returns an object associated with a name) and obtains all associated properties.

IDL Syntax

SOMObject resolve_with_all_properties(
in CosNaming::Name *name*,
in unsigned long *how_many*,
out ExtendedNaming::PropertyList *props*,
out ExtendedNaming::PropertyIterator *rest*);

Description

Resolves a name-object binding (returns an object associated with a name) and outputs all associated properties. Operates just like the **CosNaming::NamingContext::resolve** operation in that it resolves the specified name-object binding, specified by *name*, in the target extended naming context. In addition, it outputs all properties associated with name-object binding. If the name-object binding contains more than *how_many* properties, the remaining properties are put in *rest*. This method is a combination of the **resolve** method and **get_all_properties** method.

Intended Usage

This method is intended to be used by client applications. It is not typically overridden.

This method is not defined in the OMG standard.

Parameters

name
The name of the name-object binding.

how_many
The maximum number of properties to put into *props*.

props
The returned properties.

rest
The returned **PropertyIterator**.

Return Value

A **SOMObject** is returned, which is the resolved object.

Exceptions

CORBA 1.1 standard exceptions and the following user exceptions:

- **CosNaming::NamingContext::NotFound** is raised to indicate that the name does not identify a binding. If a compound name is passed as an argument for the **bind** operation, it traverses multiple contexts. A **NotFound** exception is raised if any of the intermediate contexts cannot be resolved.
- **CosNaming::NamingContext::CannotProceed** is raised to indicate that the implementation has given up for some reason. The client may be able to continue the operation using the returned naming context.
- **CosNaming::NamingContext::InvalidName** is raised to indicate that the name is invalid. A name with a length of zero is invalid. (This exception may be raised upon further implementation restrictions.)

Original Class

ExtendedNaming::ExtendedNamingContext Interface

Related Information

ExtendedNaming::PropertyIterator Interface

resolve_with_properties Method

Resolves a name-object binding (returns an object associated with a name) and obtains a set of associated properties.

IDL Syntax

```
SOMObject resolve_with_properties(
        in CosNaming::Name name,
        in unsigned long how_many,
        in ExtendedNaming::IList inames,
        out ExtendedNaming::PropertyList props,
        out ExtendedNaming::PropertyIterator rest);
```

Description

Resolves a name-object binding (returns an object associated with a name) and outputs a set of associated properties. Operates just like the **CosNaming::NamingContext::resolve** operation in that it resolves the specified name-object binding, specified by *name*, in the target extended naming context. In addition, it defines properties to be returned, with their property names specified as *inames*. If the name-object binding contains more than *how_many* properties, the remaining properties are put in *rest* (combination of **resolve** and **get_properties**).

Intended Usage

This method is intended to be used by client applications. It is not typically overridden.

This method is not defined in the OMG standard.

Parameters

name
The name of the name-object binding.

how_many
The maximum number of properties to put into *props.*

inames
List of property names.

props
The returned properties.

rest
The returned **PropertyIterator.**

Return Value

A **SOMObject** is returned, which is the resolved object.

Exceptions

CORBA 1.1 standard exceptions and the following user exceptions:

- **CosNaming::NamingContext::NotFound** is raised to indicate that the name does not identify a binding. If a compound name is passed as an argument for the **bind** operation, it traverses multiple contexts. A **NotFound** exception is raised if any of the intermediate contexts cannot be resolved.
- **CosNaming::NamingContext::CannotProceed** is raised to indicate that the implementation has given up for some reason. The client may be able to continue the operation using the returned naming context.

- **CosNaming::NamingContext::InvalidName** is raised to indicate that the name is invalid. A name with a length of zero is invalid. (This exception may be raised upon further implementation restrictions.)
- **ExtendedNaming::ExtendedNamingContext::InvalidPropertyName** is raised to indicate that the property name is invalid. A propertExtendedNaming::ExtendedNamingContexty name with a length of zero is invalid.
- **ExtendedNaming::ExtendedNamingContext::PropertyNotFound{CosNaming Istring property_name;};** is raised to indicate that a property was not found.

Original Class

ExtendedNaming::ExtendedNamingContext Interface

Related Information

ExtendedNaming::PropertyIterator Interface

resolve_with_property Method

Resolves a name-object binding (returns an object associated with a name) and obtains an associated property value.

IDL Syntax

SOMObject resolve_with_property(
in CosNaming::Name *name*,
in CosNaming::Istring *prop*,
out any *pvalue*);

Description

Resolves a name-object binding (returns an object associated with a name) and outputs the associated property value. Operates just like the **CosNaming::NamingContext::resolve** operation in that it resolves the specified name-object binding, specified by *name*, in the target extended naming context. In addition, it retrieves the value of the property *prop* associated with *name*.

Applications can use this method to resolve a name and to obtain the value of the specified property name in one invocation. You can achieve the same functionality with the **resolve** and **get_property** methods.

Intended Usage

This method is intended to be used by client applications. It is not typically overridden.

This method is not defined in the OMG standard.

Parameters

name
The name of the name-object binding.

prop
The property name.

pvalue
The returned property value.

Return Value

A **SOMObject** is returned, which is the resolved object.

Exceptions

CORBA 1.1 standard exceptions and the following user exceptions:

- **CosNaming::NamingContext::NotFound** is raised to indicate that the name does not identify a binding. If a compound name is passed as an argument for the **bind** operation, it traverses multiple contexts. A **NotFound** exception is raised if any of the intermediate contexts cannot be resolved.
- **CosNaming::NamingContext::CannotProceed** is raised to indicate that the implementation has given up for some reason. The client may be able to continue the operation using the returned naming context.
- **CosNaming::NamingContext::InvalidName** is raised to indicate that the name is invalid. A name with a length of zero is invalid. (This exception may be raised upon further implementation restrictions.)
- **ExtendedNaming::ExtendedNamingContext::InvalidPropertyName** is raised to indicate that the property name is invalid. A property name with a length of zero is invalid.

- **ExtendedNaming::ExtendedNamingContext::PropertyNotFound{CosNaming Istring property_name;};** is raised to indicate that a property was not found.

Original Class

ExtendedNaming::ExtendedNamingContext Interface

unbind Method

Removes a name-SOMObject binding.

IDL Syntax

void unbind(in CosNaming::Name *name*);

Description

Removes a binding from a context.

Intended Usage

This method is intended to be used by client applications. It is not typically overridden.

Parameters

name
The name for the name-object binding.

Exceptions

CORBA 1.1 standard exceptions and the following user exceptions:

CosNaming::NamingContext::NotFound is raised to indicate that the name does not identify a binding. If a compound name is passed as an argument for the **bind** operation, it traverses multiple contexts. A **NotFound** exception is raised if any of the intermediate contexts cannot be resolved.

CosNaming::NamingContext::CannotProceed is raised to indicate that the implementation has given up for some reason. The client may be able to continue the operation using the returned naming context.

CosNaming::NamingContext::InvalidName is raised to indicate that the name is invalid. A name with a length of zero is invalid. (This exception may be raised upon further implementation restrictions.)

Original Class

CosNaming::NamingContext

_get_allowed_object_types Method

Retrieves a list of types of objects that can be bound.

IDL Syntax

```
sequence<TypeCode> _get_allowed_object_types();
```

Description

Retrieves a list of types of objects that can be bound into the target extended naming context. An empty list implies no restrictions. This implementation places no restrictions on object types.

Intended Usage

This method is intended to be used by client applications. It is not typically overridden.

This method is not defined in the OMG standard.

Return Value

An **sequence<TypeCode>** is returned containing the allowed object types.

Exceptions

CORBA 1.1 standard exceptions.

Original Class

ExtendedNaming::ExtendedNamingContext Interface

_get_allowed_property_names Method

Retrieves a list of names of properties that can be added.

IDL Syntax

sequence<string>_get_allowed_property_names();

Description

Retrieves a list of names of properties that can be added to the target extended naming context. An empty list implies no restrictions.

Intended Usage

This method is intended to be used by client applications. It is not typically overridden.

This method is not defined in the OMG standard.

Return Value

An **sequence<string>** is returned indicating the allowed property names.

Exceptions

CORBA 1.1 standard exceptions.

Original Class

ExtendedNaming::ExtendedNamingContext Interface

_get_allowed_property_types Method

Retrieves a list of the types of the properties that can be added.

IDL Syntax

sequence<TypeCode> _get_allowed_property_types();

Description

Retrieves a list of the types of the properties that can be added to the target extended naming context. An empty list implies no restrictions. This implementation places no restrictions on the *type* of the allowed property.

Intended Usage

This method is intended to be used by client applications. It is not typically overridden.

This method is not defined in the OMG standard.

Return Value

An **sequence<TypeCode>** is returned indicating the allowed property types.

Exceptions

CORBA 1.1 standard exceptions.

Original Class

ExtendedNaming::ExtendedNamingContext Interface

Chapter 3. Object Services Server

The Object Services Server is responsible for instituting persistent object references and managing object metastate on behalf of the SOMobjects object services. The Object Services Server is composed of several parts: the server-object, an object services base class and its specializations, and the server program. The object services base class and its specializations contain operations that allow the Object Services Server to exchange metastate information with the object and to properly initialize it during the object lifecycles. The service base class also provides the implementation for the Object Identity Service. This chapter describes the **somOS::Server** class, of which the server-object is an instance, and the **somOS::ServiceBase** class and its specializations.

somOS Module

The **somOS** module contains five classes:

- **somOS::Server** to maintain object references and metastate persistently and to reactivate passivated objects automatically
- **somOS::ServiceBase** to contain operations that allow the **somOS::Server** to exchange metastate with the object, to properly initialize and uninitialize it during the object lifecycles, and to provide the notion of object identity
- **somOS::ServiceBasePRef** to automatically create and destroy persistent references for objects
- **somOS::ServiceBaseCORBA** to support objects that have a persistent reference, but do not have persistent state
- **somOS::ServiceBasePRefCORBA**, which is similar to **ServiceBaseCORBA**, but for which creating a persistent reference should be automatic.

The following is the structure for holding the metastate of a single service:

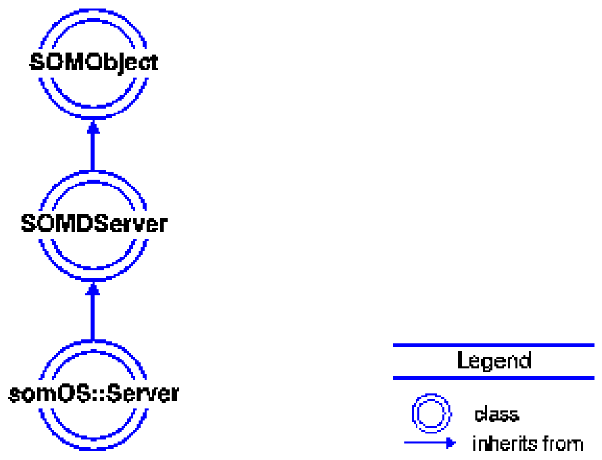
```
typedef struct metastate_struct {
    service_id_e svc_id;
    unsigned short version_major;
    unsigned short version_minor;
    any service_metastate;
} metastate_struct_t;
```

where:

```
typedef enum service_id {
    somOSNaming,
    somOSEvents,
    somOSLifeCycle,
    somOSPersistence,
    somOSSecurity,
    somOSObjectIdentity,
    somOSTransactions,
    somOSConcurrecny,
    somOSExternalization,
    somOSAttributePersistence,
    somOSLastEnum
} service_id_e;
```

Note: The above enum type `service_id_e` defines one constant for each standard OMG service. However, you can extend it to define a constant for a nonstandard service by adding a new constant just before `somOSLastEnum`. The Interface Repository must be updated based on the modified IDL. The Object Services Server then appropriately handles the metastate of the new service.

somOS::Server Class



The **somOS::Server class** is a specialization of **SOMDServer**. It maintains object references persistently, maintains persistent metastate, and automatically reactivates passivated objects.

Intended Usage

The **somOS::Server** maintains persistent references and metastate for objects that are explicitly registered with it. The references are created, deleted, and queried using **make_persistent_ref**, **delete_ref**, and **has_persistent_ref**, respectively. An object's metastate is stored and restored using **store_metastate** and **restore_metastate**. The metastate of a single service for an object is stored using **store_service_metastate**. Objects are passivated by calling **passivate_object** and **passivate_all_objects**.

The **somOS::Server** participates in the exportation and importation of object references. This is accomplished by specializing **SOMDServer** and providing unique implementations of the **somdRefFromSOMObj** and **somdSOMObjFromRef** methods.

File Stem

somos

Base Class

SOMDServer

Metaclass

SOMMSingleInstance

Ancestor Classes

SOMDServer

New Methods

delete_ref Method
has_persistent_ref Method
make_persistent_ref Method
passivate_all_objects Method
passivate_object Method
restore_metastate Method

store_metastate Method

store_service_metastate Method

Overridding Methods

somdRefFromSOMObj Method

somdSOMObjFromRef Method

delete_ref Method

Deletes a persistent reference of an object.

IDL Syntax

void delete_ref(in SOMobject *referenced_object*);

Description

To delete a persistent reference that has been created using **make_persistent_ref**. It does not delete the in-memory object. If there is no reference to an object then the user-defined exception **SysAdminException::ExNotfound** is raised.

Parameters

referenced_object
A pointer to a **SOMObject** class object.

Exceptions

One of the following user-defined exceptions is raised if the method fails:

SysAdminException::ExNotfound
Persistent reference does not exist.

SysAdminException::ExFailed
Failed for internal reason.

SysAdminException::ExFileIO
Failed because of file I/O.

Example

```
/*      Let "ostest" be a subclass of somOS::ServiceBase. The ostest
        class must be registered with the server. */

#include <somd.h>
#include <somos.h>
#include "ostest.h"

Environment ev;
SOMDServer OSServer;
ostest temp, pers;

SOM_InitEnvironment ( &ev );
SOMD_Init ( &ev );

temp = _somdCreate ( &ev, "ostest", FALSE );
OSServer = _GetInstanceManager ( temp, &ev );
        // Get the instance manager of the object.
pers = _make_persistent_ref ( OSServer, &ev, temp );
_delete_ref ( OSServer, &ev, pers );
        // Destroy the persistent reference of the object in
        // somOS::Server. Now, methods such as passivate() and
        // store_metastate() cannot be called because there is no
        // persistent reference.
```

Original Class

somOS::Server Class

Related Information

make_persistent_ref Method
has_persistent_ref Method

has_persistent_ref Method

Queries the server for a specified object as to whether the server is maintaining this object persistently.

IDL Syntax

boolean has_persistent_ref(in SOMObject *referenced_object*);

Description

The **has_persistent_ref** method returns TRUE, if the Object Services Server maintains the given object persistently, that is, the Server persistently maintains the metastate which is used to reactivate the object, if necessary.

Intended Usage

To query whether the server is maintaining an object persistently.

Parameters

referenced_object
A pointer to a SOMObject object.

Return Value

Returns TRUE if the object is persistently maintained by the server; otherwise, FALSE is returned.

Example

```
/*      Let "ostest" be a subclass of somOS::ServiceBase which defines a
        new method "Hello" which prints "Hello, World!" message. The
        ostest class must be registered with the server. */

#include <somd.h>
#include <somos.h>
#include "ostest.h"

Environment ev;
SOMDServer OSServer;
ostest temp, pers;
boolean Result;

SOM_InitEnvironment ( &ev );
SOMD_Init ( &ev );

temp = _somdCreate ( &ev, "ostest", FALSE );

OSServer = _GetInstanceManager ( temp, &ev );
        // Get the instance manager of the object.

Result = _has_persistent_ref ( OSServer, &ev, temp );
        // Result is FLASE because temp is a transient (DSOM) reference.

pers = _make_persistent_ref ( OSServer, &ev, temp );
Result = _has_persistent_ref ( OSServer, &ev, pers );
        // Result is TRUE.

_delete_ref ( OSServer, &ev, pers );
Result = _has_persistent_ref ( OSServer, &ev, pers );
        // Result is FALSE.
```

Original Class

somOS::Server Class

Related Information

make_persistent_ref Method
delete_ref Method

make_persistent_ref Method

Makes a persistent object reference for an object.

IDL Syntax

SOMObject make_persistent_ref(in SOMObject *referenced_object*);

Description

A persistent reference is created for *referenced_object*. If a persistent reference already exists, no action is taken, and the operation is considered successful.

Intended Usage

This method must be called prior to calling other methods for maintaining persistent metastate; for example, before the **store_metastate** method.

Parameters

referenced_object
A pointer to a **SOMObject** class object.

Return Value

The **make_persistent_ref** method returns **OBJECT_NIL** if, and only if, the operation is unsuccessful. The object pointer returned by the **make_persistent_ref** method must be used in the subsequent method invocations on the object. In particular, the object pointer returned by **somNewNoInit** or **somdCreate** must be discarded after the object has been registered with the server using **make_persistent_ref**.

Exceptions

The user-defined exception **SysAdminException::ExExists** occurs if a reference already exists for the input object. Note that the operation is considered successful in this case.

The following user-defined exceptions may be raised if the method fails:

SysAdminException::ExNotfound
Unable to find class name of object

SysAdminException::ExFileIO
Failed for internal reason

SysAdminException::ExFailed
Failed because of file I/O

Example

```
/*      Let "ostest" be a subclass of somOS::ServiceBase which defines a
        new method "Hello" which prints "Hello, World!" message. The
        ostest class must be registered with the server. */

#include <somd.h>
#include <somos.h>
#include "ostest.h"

Environment ev;
SOMDServer OSServer;
ostest temp, pers;

SOM_InitEnvironment ( &ev );
SOMD_Init ( &ev );

temp = _somdCreate ( &ev, "ostest", FALSE );
        // temp is a transient (DSOM) reference.
```

```
OSServer = _GetInstanceManager ( temp, &ev );
        // Get the instance manager of the object.

pers = _make_persistent_ref ( OSServer, &ev, temp );

        // Verification of the call.
if ( _is_nil ( pers, &ev ) )
        // pers is_nil if, and only if, the call fails.
        {
            printf("make_persistent_ref failed\n");
            somdExceptionFree(&ev);
            return;
        }
release( temp, &ev );
        // Release the transient proxy because it is not needed.

_Hello ( pers, &ev );
        // pers is a persistent reference and must be used subsequently.

_somdTargetFree ( pers, &ev );
        // Destroy the remote object but not its proxy or
        // persistent reference.

_Hello ( pers, &ev );
        // Object will be reactivated and the hello() method
        // will be executed on the reactivated object.
```

Original Class

somOS::Server Class

Related Information

delete_ref Method
has_persistent_ref Method

passivate_all_objects Method

Passivates all the in-memory objects that previously have been registered with the server.

IDL Syntax

void passivate_all_objects();

Description

The **passivate_all_objects** method passivates all objects.

Intended Usage

Passivates all objects. This method is useful when you are bringing down a server; allows active objects in the server a chance to store themselves prior to in-memory state loss.

Exceptions

One of the following user-defined exceptions is raised if the method fails:

SysAdminException::ExNotfound
Failed because a persistent reference could not be found, probably because **make_persistent_ref** has not been called.

SysAdminException::ExFailed
Failed for internal reason.

SysAdminException::ExFileIO
Failed because of file I/O.

Example

```
/*      Let "ostest" be a subclass of somOS::ServiceBase which defines
        a new method "Hello" which prints "Hello, World!" message.
        The ostest class must be registered with the server. */
#include <somd.h>
#include <somos.h>
#include "ostest.h"

Environment ev;
SOMDServer OSServer;
ostest temp, pers, temp2, pers2;

SOM_InitEnvironment ( &ev );
SOMD_Init ( &ev );

temp  = _somdCreate ( &ev, "ostest", FALSE );
temp2 = _somdCreate ( &ev, "ostest", FALSE );
OSServer = _GetInstanceManager ( temp, &ev );
        // Get the instance manager of the object.
pers  = _make_persistent_ref ( OSServer, &ev, temp );
pers2 = _make_persistent_ref ( OSServer, &ev, temp2 );

_passivate_all_objects ( OSServer, &ev );
        // ALL objects registered with OSServer are passivated.
if (ev._major != NO_EXCEPTION)
        return; // Failed to passivate all objects.

_Hello ( pers, &ev );
_Hello ( pers2, &ev );
        // Objects will be reactivated and the hello() method
        // will be executed on reactivated objects.
```

Original Class

somOS::Server Class

Related Information

passivate_object Method

passivate_object Method

Passivates an object that previously has been registered with the server.

IDL Syntax

void passivate_object(in SOMObject *referenced_object*);

Description

The **passivate_object** method passivates an in-memory object by capturing its metastate, storing the captured metastate on the persistent storage, performing **uninit_for_object_passivation** on the object, and destroying the in-memory object.

Parameters

referenced_object
A pointer to a **SOMObject** class object.

Exceptions

One of the following user-defined exceptions is raised if the method fails:

SysAdminException::ExNotfound
Failed because a persistent reference could not be found, probably because **make_persistent_ref** has not been called.

SysAdminException::ExFailed
Failed for internal reason.

SysAdminException::ExFileIO
Failed because of file I/O.

Example

```
/*      Let "ostest" be a subclass of somOS::ServiceBase which defines a
        new method "Hello" which prints "Hello, World!" message. The
        ostest class must be registered with the server. */
#include <somd.h>
#include <somos.h>
#include "ostest.h"

Environment ev;
SOMDServer OSServer;
ostest temp, pers;
boolean Result;

SOM_InitEnvironment ( &ev );
SOMD_Init ( &ev );

temp = _somdCreate ( &ev, "ostest", FALSE );
OSServer = _GetInstanceManager ( temp, &ev );
        // Get the instance manager of the object.
pers = _make_persistent_ref ( OSServer, &ev, temp );

_passivate_object ( OSServer, &ev, pers );
        // pers must have a persistent reference prior to this call.

_Hello ( pers, &ev );
        // Object will be reactivated and the hello() method will be
        // executed on the reactivated object.
```

Original Class

somOS::Server Class

Related Information

passivate_all_objects Method

restore_metastate Method

Restores the metastate of an object that has been previously stored.

IDL Syntax

void restore_metastate(in SOMObject *referenced_object*);

Description

The **restore_metastate** method finds the metastate of the specified object that has been previously stored and calls **reinit** on the specified object to reinitialize the object using the metastate. The **reinit** method on the specified object usually initializes the state of the object needed by the object services associated with it.

Intended Usage

This method should be called on an object after the metastate of the object has been stored. This method should be called whenever it is appropriate to bring the object's state back to the one stored persistently.

Parameters

reference_object
A pointer to a **SOMObject** class object.

Exceptions

One of the following user-defined exceptions is raised if the method fails:

SysAdminException::ExNotfound
Failed because a persistent reference could not be found, probably because **make_persistent_ref** has not been called.

SysAdminException::ExFailed
Failed for internal reason.

SysAdminException::ExFileIO
Failed because of file I/O.

Example

```
/*      Let "ostest" be a subclass of somOS::ServiceBase. The ostest
        class must be registered with the server. */
#include <somd.h>
#include <somos.h>
#include "ostest.h"

Environment ev;
SOMDServer OSServer;
ostest temp, pers;

SOM_InitEnvironment ( &ev );
SOMD_Init ( &ev );

temp  = _somdCreate ( &ev, "ostest", FALSE );
OSServer = _GetInstanceManager ( temp, &ev );
        // Get the instance manager of the object.
pers  = _make_persistent_ref ( OSServer, &ev, temp );
        // Object must have persistent reference before
        // store_metastate() can be called.
...

        // Store the metastate using, e.g., store_metastate()
        // or passivate_object().
```

```
...

_restore_metastate ( OSServer, &ev, pers );
        // OSServer calls gets metastate from persistent storage
        // and calls reinit on object pers to reinitialize it.
```

Original Class

somOS::Server Class

Related Information

store_metastate Method

somdRefFromSOMObj Method

Returns an object reference corresponding to the specified SOM object.

IDL Syntax

SOMDObject somdRefFromSOMObj(in SOMObject *somobj*);

Description

The **somdRefFromSOMObj** method is overridden to return special object references for objects registered with the **somOS::Server**.

Intended Usage

The **somdRefFromSOMObj** method creates a reference to a SOM object in a server, to be exported to a client as a proxy. This method is called by DSOM as part of converting the results of a local method call into a result message for a remote client, whenever the result contains a pointer to an object local to the server. Although this method usually is not called directly, you may need to call it directly under the following circumstance. Assume that an object is registered with the server, although it is not exported. In this case, the **somdRefFromSOMObj** method can be invoked on the **somOS::Server** directly. Because the **somOS::Server** does not distinguish whether the request is initiated by the SOMOA or some other object, the result is the same.

Parameters

receiver
A pointer to a **SOMDServer** object.

env
A pointer to the **Environment** structure for the method caller.

somobj
A pointer to the SOM object for which a DSOM reference is to be created.

Return Value

The **somdRefFromSOMObj** method returns a dsom reference for the specified SOM object. If the object is registered with the **somOS::Server**, it contains special reference data needed to maintain the lifecycle of persistent objects.

Original Class

Server

Related Information

somdSOMObjFromRef Method
somdRefFromSOMObj Method in **SOMDServer** class

somdSOMObjFromRef Method

Returns the SOM object that corresponds to the specified object reference.

IDL Syntax

SOMObject somdSOMObjFromRef(in SOMDObject *objref*);

Description

The **somdSOMObjFromRef** method is overridden to handle the special **somOS::Server** object references. If the reference belongs to **somOS::Serve**r, it returns the SOM object associated with it. If the object is passivated, it reactivates the object and returns the reactivated object. Otherwise, this method on the parent is called to return the associated SOM object.

Intended Usage

When an object is used in an **in** or **inout** argument to a method request entering the server process from another process, the SOMOA invokes **somdSOMObjFromRef** on the **somOS::Server**. This serves as the dual to exporting a reference.

Parameters

receiver
A pointer to a **SOMDServer** object.

env
A pointer to the **Environment** structure for the method caller.

objref
A pointer to the DSOM object reference to the SOM object.

Return Value

The **somdSOMObjFromRef** method returns the SOM object (possibly reactivated) associated with the supplied object reference.

Original Class

Server

Related Information

somdRefFromSOMObj Method
somdSOMObjFromRef Method in **SOMDServer** class

store_metastate Method

Stores the metastate of an object.

IDL Syntax

void store_metastate(in SOMObject *referenced_object*);

Description

The **store_metastate** calls **capture** on the specified object to obtain its metastate and stores the metastate on the persistent storage. The **capture** method of the specified object usually captures the metastate of all the services associated with it.

Intended Usage

This method should be called on an object after the object has been registered with the **somOS::Serve**r; for example, by using **make_persistent_ref** or an instance of **somOS::ServiceBase** class. It should be called if it is considered appropriate to persistently store the metastate of object services associated with the object.

Parameters

referenced_object
A pointer to a SOMObject class object.

Exceptions

One of the following user-defined exceptions is raised if the method fails:

SysAdminException::ExNotfound
Failed because a persistent reference could not be found, probably because **make_persistent_ref** has not been called.

SysAdminException::ExFailed
Failed for internal reason.

SysAdminException::ExFileIO
Failed because of file I/O.

Example

```
/*      Let "ostest" be a subclass of somOS::ServiceBase. The ostest
        class must be registered with the server. */
#include <somd.h>
#include <somos.h>
#include "ostest.h"

Environment ev;
SOMDServer OSServer;
ostest temp, pers;

SOM_InitEnvironment ( &ev );
SOMD_Init ( &ev );

temp  = _somdCreate ( &ev, "ostest", FALSE );
OSServer = _GetInstanceManager ( temp, &ev );
        // Get the instance manager of the object.
pers  = _make_persistent_ref ( OSServer, &ev, temp );
        // Object must have persistent reference before
        // store_metastate() can be called.

_store_metastate ( OSServer, &ev, pers );
        // OSServer calls capture on object pers to get its metastate
        // and stores it persistently.
```

Original Class

somOS::Server Class

Related Information

restore_metastate Method

store_service_metastate Method

Stores the metastate of a single service of an object to the persistent storage.

IDL Syntax

void store_service_metastate(
in SOMObject *referenced_object*,
in service_id_e *somos_service_id*,
in any *service_metadata*);

Description

The **store_service_metastate** method persistently stores the given metastate of a single service for the specified object.

Intended Usage

Call this method on an object after the object has been registered with the **somOS::Server**; for example, using **make_persistent_ref** for an instance of **somOS::ServiceBase class**. Call it when considered appropriate to persistently store the metastate of an object service associated with the object. It is the caller's responsibility to free the memory allocated to store the metadata.

Parameters

referenced_object
A pointer to an object reference.

somos_service_id
The service for which to store the metastate.

service_metadata
A pointer to the metastate data of the service.

Exceptions

One of the following user-defined exceptions is raised if the method fails:

SysAdminException::ExNotfound
Failed because a persistent reference could not be found, probably because **make_persistent_ref** has not been called.

SysAdminException::ExFailed
Failed for internal reason.

SysAdminException::ExFileIO
Failed because of file I/O.

Example

```
/*      Let "ostest" be a subclass of somOS::ServiceBase. The ostest
        class must be registered with the server. */

#include <somd.h>
#include <somos.h>
#include "ostest.h"

Environment ev;
SOMDServer OSServer;
ostest temp, pers;
somOS_service_id_e id = somOS_somOSNaming;
any service_metadata;

SOM_InitEnvironment ( &ev );
```

```
SOMD_Init ( &ev );

temp  = _somdCreate ( &ev, "ostest", FALSE );

OSServer = _GetInstanceManager ( temp, &ev );
        // Get the instance manager of the object.

pers  = _make_persistent_ref ( OSServer, &ev, temp );
        // Object must have persistent reference before
        // store_service_metastate() can be called.

service_metadata._type = TC_short;
service_metadata._value = (short *) SOMMalloc(sizeof(short));
        // Allocate space for metadata.
service_metadata._value = 5;
        // Initialize metadata.

_store_service_metastate ( OSServer, &ev, pers, id,
                                   &service_metadata );
        // service_metadata is store as the metadata for service id
        // for object pers.

SOMFree(service_metadata._value);
```

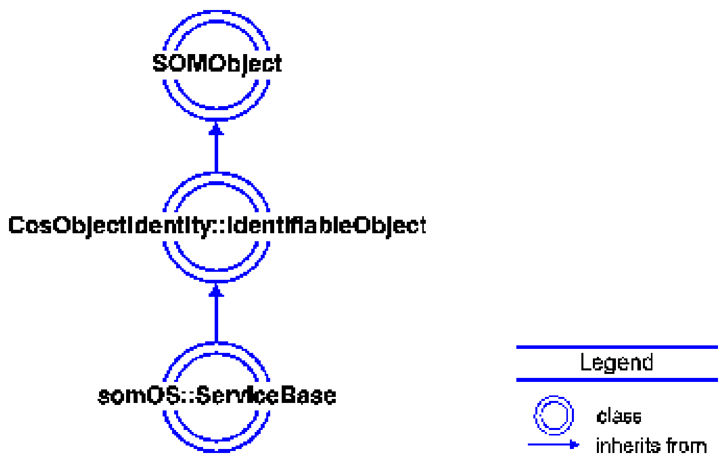
Original Class

somOS::Server Class

Related Information

store_metastate Method
restore_metastate Method

somOS::ServiceBase Class



The **somOS::ServiceBase** class is the base-class for the Object Services Server. The **somOS::ServiceBase** contains operations that allow the **somOS::Server** to exchange metastate with the object, to properly initialize and uninitialize it during the object lifecycles, and to provide the notion of object identity.

File Stem

somos

Base Class

SOMObject

Metaclass

SOMClass

Ancestor Classes

CosObjectIdentity::IdentifiableObject

Types

typedef sequence <metastate_struct_t> metastate_t;
typedef unsigned long ObjectIdentifer

New Methods

capture Method
init_for_object_copy Method
init_for_object_creation Method
init_for_object_reactivation Method
is_identical Method
reinit Method
uninit_for_object_destruction Method
uninit_for_object_move Method
uninit_for_object_passivation Method
_get_constant_random_id Method

Overridding Methods

somDestruct Method

capture Method

Obtains the metastate of an object.

IDL Syntax

void capture(inout metastate_t *metadata*);

Description

The **capture** method obtains the metastate of an object. The metastate of an object is defined by the metastate of the services associated with the object.

Instances of **somOS::ServiceBase** have private metastate that provides the object identity characteristics needed to support the implementation of OMG-defined **CosObjectIdentity::IdentifiableObject**. The **somOS::ServiceBase** provides an implementation of **capture** that captures its object identity metastate. Therefore, it is important that subclasses of **somOS::ServiceBase** that override **capture** also call their parent's implementation of **capture**.

Intended Usage

The **capture** method should be overridden and all subclasses of **somOS::ServiceBase** should call their parents implementation of **capture**. This methods enables the object's metastate to be captured in *metadata*. *metadata* is a sequence of **metastate_struct_t**. Each element of the sequence contains the metastate of a service. The *service_metastate* field of a service is filled by the **capture** method. The **capture** method must explicitly allocate memory to hold the **service_metastate** data (including type code). The server frees this memory.

Parameters

metadata
A partially initialized sequence to hold the metastate of the object.

Example

The following example is for class programmers:

```
/*      In a subclass called ostest of somOS::ServiceBase ... */
#include <somd.h>
#include "ostest.ih"
SOM_Scope void  SOMLINK capture(ostest somSelf,  Environment *ev,
                              somOS_ServiceBase_metastate_t* meta_data)
{

        ostestData *somThis = ostestGetData(somSelf);
        ostestMethodDebug("ostest","capture");
        ostest_parent_somOS_ServiceBase_capture(somSelf, ev, meta_data);

        //  Insert code here to capture metadata in meta_data. For
        //  example, ...
        meta_data->_buffer[somOS_somOSNaming].service_metastate._type =
             TypeCode_copy(TC_long);
        meta_data->_buffer[somOS_somOSNaming].service_metastate._value =
             SOMMalloc( sizeof ( long ));

        *((long *) (meta_data->_buffer[somOS_somOSNaming].
             service_metastate._value)) = somThis->MyVariable;
        ...

}
```

Note: If you are a client programmer, do not call this this method directly.

Original Class

somOS::ServiceBase

Related Information

reinit Method

GetInstanceManager Method

Obtains a reference to the instance manager of an object.

IDL Syntax

SOMObject GetInstanceManager ( );

Description

The **GetInstanceManager** method obtains a reference to the instance manager of an object. If the object does not have an instance manager, OBJECT_NIL is returned.

Intended Usage

A client that needs to register an object and maintain its life cycle with the Object Services Server must obtain a reference to the server (instance manager) using the **GetInstanceManager** method. This method usually is not overridden by the class programmer.

Return Value

Returns an object reference to the instance manager of the input object.

Example

```
/*      Let "ostest" be a subclass of somOS::ServiceBase which defines a
        new method "Hello" which prints "Hello, World!" message. The
        ostest class must be registered with the server. */

#include <somd.h>
#include <somos.h>
#include "ostest.h"

Environment ev;
SOMDServer OSServer;
ostest temp, pers;

SOM_InitEnvironment ( &ev );
SOMD_Init ( &ev );

temp = _somdCreate ( &ev, "ostest", FALSE );
        // temp is a transient (DSOM) reference.

OSServer = _GetInstanceManager ( temp, &ev );

if (_is_nil(OSServer, &ev))
               return;

        // Return if there is no instance manager associated with
        // temp object.
```

Original Class

somOS::ServiceBase

init_for_object_copy Method

Enables the object to perform initialization on itself after it has been created as a part of the **copy** operation.

IDL Syntax

SOMObject init_for_object_copy();

Description

The **init_for_object_copy** method performs initialization when an object is copied.

Intended Usage

The **init_for_object_copy** method is an abstract method that must be overridden if special initilization is required when an object is copied. SOMobjects Version 3.0 does not implement this method. However, you may want to supply this method when the object copy feature is added in the future.

Return Value

Returns the object reference of the copied object.

Original Class

somOS::ServiceBase

Related Information

somDestruct Method

init_for_object_creation Method

Enables the object to perform initialization on itself after it is created the first time and establishes the instance's identtity.

IDL Syntax

SOMObject init_for_object_creation();

Description

The **init_for_object_creation** method performs initialization at the time of object creation, including establishing the instance's identity. Note that the identity of the object is not persistent unless **make_persistent_ref** is called on the **somOS::Server** for the instance.

Intended Usage

The **init_for_object_creation** method must be overridden to perform proper initialization at the time of object creation. For example, you can register the object in any frameworks or containers to which the object should belong. You also can allocate any memory the object requires or create and initialize a persistent reference for the object.

It is important that subclasses of **somOS::ServiceBase** that override **init_for_object_creation** ensure that their parent's **init_for_object_creation** is called in the override.

Return Value

Returns the object reference of the newly created object.

Example

The following example is for class programmers:

```
/* In the ostest subclass of somOS::ServiceBase ... */
#include <somd.h>
#include "ostest.ih"

SOM_Scope SOMObject  SOMLINK init_for_object_creation(ostest somSelf,
                                                     Environment *ev)
{

        ostestData *somThis = ostestGetData(somSelf);
        ostestMethodDebug("ostest","init_for_object_creation");

        ostest_parent_somOS_ServiceBase_init_for_object_creation
             (somSelf, ev);

        // Insert code here to perform initialization. For example,
        ...
        somThis->MyVariable = (long *) SOMMalloc ( sizeof(long) );
        *(somThis->MyVariable) = VALUE_AT_THE_CREATE_TIME;
        ...
        return (somSelf);
}
```

The following example is for client programmers:

```
#include <somd.h>
#include <somos.h>
#include "ostest.h"

Environment ev;
ostest temp, pers;
```

```
SOM_InitEnvironment ( &ev );
SOMD_Init ( &ev );

temp = _somdCreate ( &ev, "ostest", FALSE );
        // temp is a transient (DSOM) reference.
pers = _init_for_object_creation( temp, &ev );
        // Subsequently use pers and discard temp,
        // if the method creates a persistent reference.
```

Original Class

somOS::ServiceBase

Related Information

uninit_for_object_destruction Method

init_for_object_reactivation Method

Enables the object to perform initialization after it has been re-activated.

IDL Syntax

SOMObject init_for_object_reactivation();

Description

The **init_for_object_reactivation** method performs reinitialization at the time of object reactivation.

Intended Usage

The **init_for_object_reactivation** method is an abstract method. It must be overridden to perform proper reinitialization at the time of object reactivation. For example, you can allocate any memory the object requires. Unlike the **init_for_object_creation** method, you may not have to register the object with frameworks or create a persistent reference. The server calls this method when an object is reactivated; for example, when a method is called on an already passivated object.

Return Value

Returns the object reference of the re-activated object.

Example

The following example is for class programmers:

```
/* In the ostest subclass of somOS::ServiceBase ... */

#include <somd.h>
#include "ostest.ih"

SOM_Scope SOMObject  SOMLINK init_for_object_reactivation(ostest
                                                 somSelf, Environment *ev)
{

        ostestData *somThis = ostestGetData(somSelf);
        ostestMethodDebug("ostest","init_for_object_reactivation");

        // Insert code here to perform reactivation. For example, ...
        somThis->MyVariable = (long *) SOMMalloc ( sizeof(long) );
        *(somThis->MyVariable) = VALUE_AT_THE_REACTIVATION_TIME;
        ...

        ostest_parent_somOS_ServiceBase_init_for_object_reactivation
             (somSel, ev);

        return ( somSelf );

}
```

Note: If you are a client programmer, do not call this method directly.

Original Class

somOS::ServiceBase

Related Information

uninit_for_object_passivation Method

is_identical Method

Determines if two objects are identical.

IDL Syntax

boolean is_identical(in IdentifiableObject *other_object*);

Description

Determines if two objects are identical.

Parameters

other_object
Identity comparison requires two objects: a target object and some other object. The *is_identical* method is invoked on a target object and compares the identity of the target object to the *other_object*.

Return Value

Returns TRUE if the object and the *other_object* are identical; otherwise, the operation returns FALSE.

Example

```
/*      Let "ostest" be a subclass of ServiceBase.
        The ostest class must be registered with the server. */

#include <somd.h>
#include <somos.h>
#include "ostest.h"

Environment ev;
ostest temp, pers, pers2;
ObjectIdentifier id;
boolean Result;

SOM_InitEnvironment ( &ev );
SOMD_Init ( &ev );

temp = _somdCreate ( &ev, "ostest", FALSE );
        // temp is a transient (DSOM) reference.

pers = _init_for_object_creation( temp, &ev );
pers2 = _init_for_object_creation( temp, &ev );

Result = _is_identical(pers, pers2);
        // Result is TRUE.
```

Exceptions

CORBA 1.1 standard exceptions.

- If the *other_object* is not a subclass of **somOS::ServiceBase**, then the BAD_PARAM CORBA standard exception is raised.

Original Class

CosObjectIdentity::IdentifiableObject

reinit Method

Reinitializes an object’s metastate.

IDL Syntax

void reinit(in metastate_t *metadata*);

Description

The **reinit** method initializes the metastate of the object, based on *metadata*. For the **somOS::ServiceBase** class, this method reinitializes the identity metastate of the instance.

Intended Usage

The **reinit** method should be overridden. This method enables the object’s metastate to be reinitialized based on *metadata*. The **somOS::Server** calls this method every time an object is reactivated. If you are a client programmer, do not call this method directly.

Parameters

metadata
A sequence holding the metadata of services.

Example

The following example is for class programmers:

```
/* In a subclass called ostest of somOS::ServiceBase ... */

#include <somd.h>
#include "ostest.ih"

SOM_Scope void  SOMLINK reinit(ostest somSelf,  Environment *ev,
                                   somOS_ServiceBase_metastate_t*
                                   meta_data)
{

        ostestData *somThis = ostestGetData(somSelf)
        ostestMethodDebug("ostest","reinit");

        ostest_parent_somOS_ServiceBase_reinit(somSelf, ev, meta_data);

        // Insert code here to perform reinitialization using metadata.
        // For example, ...

        somThis->MyVariable =
             *((long *)(meta_data->_buffer[somOS_somOSNaming].
                  service_metastate._value));
        ...

}
```

Note: If you are a client programmer, do not call this method directly.

Original Class

somOS::ServiceBase

Related Information

capture Method

somDestruct Method

This method is overridden to remove the entry for this object, if any, in the reference data table of the server.

IDL Syntax

void somDestruct(int octet *dofree*, inout somDestructCtrl *ctrl*);

Description

The overridden **somDestruct** method attempts to remove the entry for this object from the reference data table of the server (if it exists) prior to actually destroying the in-memory object. The attempt to remove an entry is considered to be successful if the server does not exist or if no entry exists in the reference data table.

Intended Usage

The server calls this method to destroy the in-memory object. If this method is called on a proxy, it destroys both the proxy and its target object. Call the **init_for_object_destruction** method before calling this method.

Parameters

receiver
A pointer to an object.

dofree
A boolean that indicates whether the caller wants the object storage freed after uninitialization of the current class has been completed. Passing 1 (true) indicates that the object storage should be freed.

ctrl
A pointer to a **somDestructCtrl** data structure. SOMobjects uses this data structure to control the uninitialization of the ancestor classes, thereby ensuring that no ancestor class receives multiple uninitialization calls. If a user invokes **somDestruct** on an object directly, a NULL (that is, zero) ctrl pointer can be passed. This instructs the receiving code to obtain a somDestructCtrl data structure from the class of the object.

Example

```
/*      Let "ostest" be a subclass of somOS::ServiceBase.
        The ostest class must be registered with the server. */

#include <somd.h>
#include <somos.h>
#include "ostest.ih"

Environment ev;
ostest temp, pers;

SOM_InitEnvironment ( &ev );
SOMD_Init ( &ev );

temp = _somdCreate ( &ev, "ostest", FALSE );
        // temp is a transient (DSOM) reference.

pers = _init_for_object_creation( temp, &ev );
        // Subsequently use pers and discard temp,
        // if the method creates a persistent reference.
...

_somDestruct(pers, 1, NULL);
        // Destroys both the proxy and target objects.
```

Original Class

somOS::ServiceBase

Related Information

uninit_for_object_destruction Method

uninit_for_object_destruction Method

Enables the object to perform uninitialization on itself prior to being destroyed.

IDL Syntax

void uninit_for_object_destruction();

Description

The **uninit_for_object_destruction** method performs uninitialization prior to destroying an object.

Intended Usage

The **uninit_for_object_destruction** method is an abstract method. Override it if some uninitialization must be performed prior to object destruction. For example, you may need to register the object in any frameworks or containers in which the object has been registered, or you may need to free previously allocated memory, or destroy the persistent reference (if appropriate) for the object.

Example

The following example is for class programmers:

```
/*       In the ostest subclass of somOS::ServiceBase ... */

#include <somd.h>
#include "ostest.ih"

SOM_Scope void  SOMLINK uninit_for_object_destruction(ostest
                                            somSelf, Environment *ev)
{

        ostestData *somThis = ostestGetData(somSelf);
        ostestMethodDebug("ostest","init_for_object_destruction");

        // Insert code here to perform destruction. For example, ...
        SOMFree(somThis->MyVariable);
        ...

        ostest_parent_somOS_ServiceBase_uninit_for_object_destruction
                                                        (somSelf, ev);

        return ( somSelf );

};
```

The following example is for client programmers:

```
/*       Let "ostest" be a subclass of ServiceBase.
         The ostest class must be registered with the server. */

#include <somd.h>
#include <somos.h>
#include "ostest.ih"

Environment     ev;
ostest temp, pers;

SOM_InitEnvironment ( &ev );
SOMD_Init ( &ev );

temp = _somdCreate ( &ev, "ostest", FALSE );
        // temp is a transient (DSOM) reference.
```

```
pers = _init_for_object_creation( temp, &ev );
        // Subsequently use pers and discard temp,
        // if the method creates a persistent reference.
...

_uninit_for_object_destruction( pers, &ev );
        // If persistent reference was created for pers then it is not
        // automatically destroyed by this method call.

_somDestruct(pers, 1, NULL);
        // Destroys both the proxy and target objects.
```

Original Class

somOS::ServiceBase

Related Information

init_for_object_creation Method

uninit_for_object_move Method

Enables the object to perform uninitialization on itself prior to being destroyed as a part of the **move** operation.

IDL Syntax

void uninit_for_object_move();

Description

The **init_for_object_move** method performs initialization when an object is moved.

Intended Usage

The **init_for_object_move** method is an abstract method that must be overridden if special initilization is required when an object is moved. SOMobjects Version 3.0 does not implement this method. However, you may want to supply this method when the object move feature is added in the future.

Original Class

somOS::ServiceBase

uninit_for_object_passivation Method

Enables the object to perform uninitialization on itself prior to being passivated.

IDL Syntax

void uninit_for_object_passivation();

Description

The **uninit_for_object_passivation** method performs uninitialization when an object is passivated.

Intended Usage

The **uninit_for_object_passivation** method is called just before passivating an object. The **somOS::Server** calls this method on the object when it is passivated using **passivate_object** or **passivate_all_objects** on the server. Override this method if you must perform additional uninitialization as compared to the default implementation. For example, you can free any memory that was allocated at the time of object creation or reactivation. You must treat this method as an uninitializer or destructor. Therefore, if you override this method, be sure to call the parent's uninitializer.

Example

The following example is for class programmers:

```
/* In the ostest subclass of somOS::ServiceBase ... */

#include <somd.h>
#include "ostest.ih"

SOM_Scope void SOMLINK uninit_for_object_passivation(ostest
                                            somSelf, Environment *ev)
{

        ostestData *somThis = ostestGetData(somSelf);
        ostestMethodDebug("ostest","init_for_object_passivation");

        ostest_parent_somOS_ServiceBase_uninit_for_object_passivation
                                                  (somSelf, ev);

        // Insert code here to perform passivation. For example, ...
        SOMFree(somThis->MyVariable);
        ...
        return ( somSelf );

}
```

Note: If you are a client programmer, do not call this method directly.

Original Class

somOS::ServiceBase

_get_constant_random_id Method

Returns the value of the **constant_random_id** attribute.

IDL Syntax

readonly attribute ObjectIdentifier constant_random_id;

Description

Gets the **constant_random_id** attribute value.

The **constant_random_id** attribute value is initialized when a **somOS::ServiceBase** instance is created.

Intended Usage

The returned value usually is used for a first-order identity comparison. If two **constant_random_id** values have the same value, that does not ensure that the two objects are identical. This is an indication that they *might* be identical. Use the **is_identical** method to establish absolute identity.

This method typically is not overridden.

Return Value

ObjectIdentifier
constant_random_id attribute value.

Example

```
/*      Let "ostest" be a subclass of somOS::ServiceBase.
        The ostest class must be registered with the server. */

#include <somd.h>
#include <somos.h>
#include "ostest.h"

Environment ev;
ostest temp, pers;
CosObjectIdentity_ObjectIdentifier id;

SOM_InitEnvironment ( &ev );
SOMD_Init ( &ev );

temp = _somdCreate ( &ev, "ostest", FALSE );
        // temp is a transient (DSOM) reference.

pers = _init_for_object_creation( temp, &ev );
        // Subsequently use pers and discard temp,
        // if the method creates a persistent reference.

id = __get_constant_random_id(pers, &ev);
```

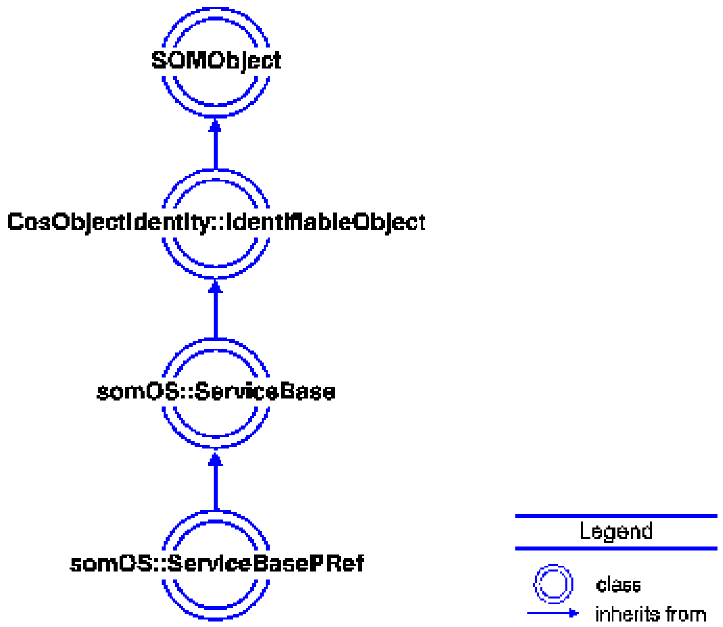
Exceptions

CORBA 1.1 standard exceptions.

Original Class

CosObjectIdentity::IdentifiableObject

somOS::ServiceBasePRef Class



somOS::ServiceBasePRef class is derived from **somOS::ServiceBase**. Objects derived from this class automatically invoke **make_persistent_ref** during **init_for_object_creation**. This specialization also automatically invokes **delete_ref** during **uninit_for_object_destruction**.

This class automatically registers the object with the server-object and requests the creation of a persistent reference for it when the object is created. This class also automatically destroys the persistent object reference for the object when the object is destroyed.

Intended Usage

If you want to automatically create and destroy object references for instances of your class, mix-in **somOS::ServiceBasePRef**.

File Stem

somos

Base Class

somOS::ServiceBase

Metaclass

SOMClass

Ancestor Classes

somOS::ServiceBase
SOMObject
CosObjectIdentity::IdentifableObject

New Methods

None.

Overridding Methods

init_for_object_creation
uninit_for_object_destruction

init_for_object_creation Method

Enables the object to perform initialization on itself after it is created the first time.

IDL Syntax

SOMObject init_for_object_creation();

Description

The default implementation of the **init_for_object_creation** method registers the object with the server by calling **make_persistent_ref**.

Intended Usage

The **init_for_object_creation** method performs routine initialization needed for the objects derived from **somOS::ServiceBasePRef** class. It must be overridden to perform proper initialization at the time of object creation. For example, you can register the object in any frameworks or containers to which the object should belong. You also can allocate any memory the object requires.

Return Value

Returns the object reference of the newly created object.

Example

The following example is for class programmers:

```
/* In the ostest subclass of somOS::ServiceBasePRef ... */
#include <somd.h>
#include "ostest.ih"

SOM_Scope SOMObject  SOMLINK init_for_object_creation(ostest somSelf,
                                                    Environment *ev)
{

        ostestData *somThis = ostestGetData(somSelf);
        ostestMethodDebug("ostest","init_for_object_creation");
        ostest_parent_somOS_ServiceBasePRef_init_for_object_creation
                                                         (somSelf, ev);
        // Insert code here to perform initialization. For example,
        ...
        somThis->MyVariable = (long *) SOMMalloc ( sizeof(long) );
        *(somThis->MyVariable) = VALUE_AT_THE_CREATE_TIME;
        ...
        return ( somSelf);

}
```

The following example is for client programmers:

```
#include <somd.h>
#include <somos.h>
#include "ostest.h"

Environment ev;
ostest temp, pers;

SOM_InitEnvironment ( &ev );
SOMD_Init ( &ev );

temp = _somdCreate ( &ev, "ostest", FALSE );
        // temp is a transient (DSOM) reference.
pers = _init_for_object_creation( temp, &ev );
        // Subsequently use pers and discard temp
```

```
                    // because pers is a persistent reference.
```

Original Class

somOS::ServiceBase

Related Information

uninit_for_object_destruction Method

uninit_for_object_destruction Method

Enables the object to perform uninitialization on itself prior to being destroyed.

IDL Syntax

void uninit_for_object_destruction();

Description

The default implementation of the **uninit_for_object_destruction** method removes the persistent reference of an object in the server by calling **delete_ref** on the server.

Intended Usage

The **uninit_for_object_destruction** method performs routine uninitialization needed for the objects derived from the **somOS::ServiceBasePRef** class. It must be overridden if additional uninitialization must be performed before object destruction. For example, you may need to register the object in any Frameworks or containers in which the object has been registered. Or you may need to free previously allocated memory.

Example

The following example is for class programmers:

```
/* In the ostest subclass of somOS::ServiceBasePRef ... */

#include <somd.h>
#include "ostest.ih"

SOM_Scope void  SOMLINK uninit_for_object_destruction(ostest
                                         somSelf, Environment *ev)
{

        ostestData *somThis = ostestGetData(somSelf);
        ostestMethodDebug("ostest","init_for_object_destruction");

        // Insert code here to perform destruction. For example, ...
        SOMFree(somThis->MyVariable);
        ...
       ostest_parent_somOS_ServiceBasePRef_uninit_for_object_destruction
                                                     (somSelf, ev);

        return ( somSelf );

}
```

The following example is for client programmers:

```
/*      Let "ostest" be a subclass of ServiceBasePRef.
        The ostest class must be registered with the server. */

#include <somd.h>
#include <somos.h>
#include "ostest.ih"

Environment ev;
ostest temp, pers;

SOM_InitEnvironment ( &ev );
SOMD_Init ( &ev );

temp = _somdCreate ( &ev, "ostest", FALSE );
        // temp is a transient (DSOM) reference.
```

```
pers = _init_for_object_creation( temp, &ev );
        // Subsequently use pers and discard temp
        // because pers is a persistent reference.
...
_uninit_for_object_destruction( pers, &ev);
        // Persistent reference is automatically destroyed.

_somDestruct(pers, 1, NULL);
        // Destroys both the proxy and target objects.
```

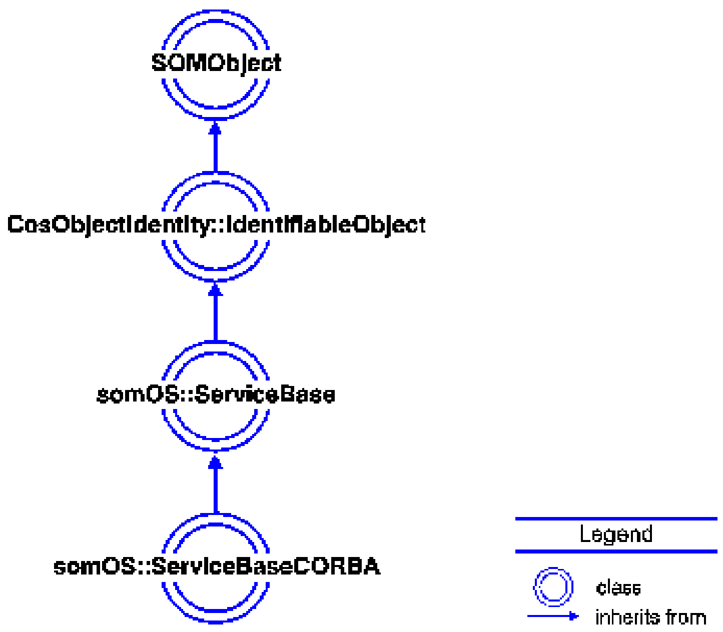
Original Class

somOS::ServiceBase

Related Information

init_for_object_creation Method

somOS::ServiceBaseCORBA Class



somOS::ServiceBaseCORBA class is a specialization of **somOS::ServiceBase** to support objects that have a persistent reference but do not have persistent state; namely, objects that are CORBA-compliant. When a method is invoked on an instance that has been passivated and reactivated, an **INV_OBJREF** standard exception is raised.

Intended Usage

If your class does not maintain the state for its instances persistently and you must maintain strict compliance with CORBA, then you must mix-in your class with the **somOS::-ServiceBaseCORBA** class. Having done so, when a method is invoked on an instance of your class that has been passivated and reactivated, an **INV_OBJREF** standard exception is raised.

File Stem

somos

Base Class

somOS::ServiceBase

Metaclass

SOMClass

Ancestor Classes

ServiceBase
SOMObject
CosObjectIdentity::IdentifiableObject

New Methods

None.

Overridding Methods

init_for_object_reactivation

init_for_object_reactivation Method

Raises the **INV_OBJREF** standard exception.

IDL Syntax

SOMObject init_for_object_reactivation();

Description

The **init_for_object_reactivation** method raises the **INV_OBJREF** standard exception to support CORBA-compliance for instances of the **somOS::ServiceBaseCORBA** class.

Intended Usage

The server calls this method when an object is reactivated; for example, when a method is called on an already passivated object. Because an exception is raised, the server fails to reactivate the object, and the invoked method is not executed.

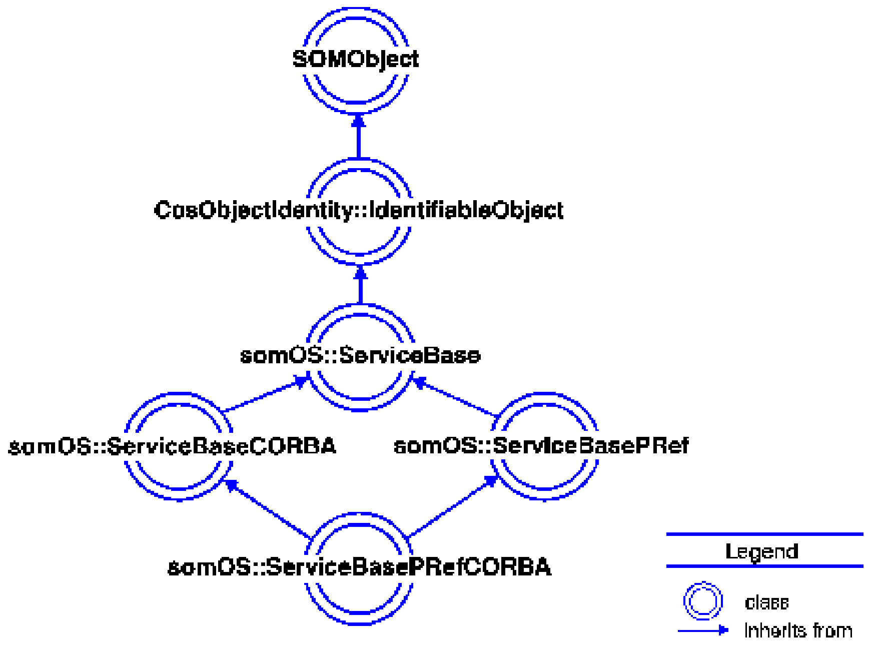
Return Value

Returns an exception.

Original Class

ServiceBaseCORBA

somOS::ServiceBasePRefCORBA Class



somOS::ServiceBasePRefCORBA is a specialization of **somOS::ServiceBase**. This class is similar to **ServiceBaseCORBA**, but persistent references are created automatically. The **somOS::ServiceBasePRefCORBA** class inherits from **somOS::ServiceBasePRef** and **ServiceBaseCORBA**. For object instances of this class, calling **init_for_object_reactivation** raises the **INV_OBJREF** standard exception.

Intended Usage

Use for objects having a persistent reference but no persistent state; namely, CORBA-compliant objects and persistent references that are automatically created and destroyed.

File Stem

somos

Base Class

somOS::ServiceBasePRef
ServiceBaseCORBA

Metaclass

SOMClass

Ancestor Classes

ServiceBasePRef
ServiceBaseCORBA
ServiceBase
SOMObject
CosObjectIdentity::IdentifiableObject

New Methods

None.

Overridding Methods

init_for_object_reactivation

init_for_object_reactivation Method

Raises the **INV_OBJREF** standard exception.

IDL Syntax

SOMObject init_for_object_reactivation();

Description

The **init_for_object_reactivation** method raises the **INV_OBJREF** standard exception to support CORBA-compliance for instances of **ServiceBasePrefCORBA** class.

Intended Usage

The server calls this method when an object is reactivated; for example, when a method is called on an already passivated object. Because an exception is raised, the server fails to reactivate the object, and the invoked method is not executed.

Return Value

Returns the object pointer of the specified object.

Original Class

somOS::ServiceBasePRefCORBA

Appendix A. BNF for Naming Constraint Language

The Naming Service allows searches based on properties attached to a name object binding. Service providers register their service and use *properties* to describe the service offered. Potential clients can then use a constraint expression to describe the requirements that service providers must satisfy. Constraints are expressed in a constraint language. Using the constraint language, you can specify arbitrarily complex expressions that involve property names and potential values.

The constraint language described below is an excerpt from Appendix B of the *Common Object Services Specification Volume 1* (OMG Document Number 94-1-1). It has been slightly modified to support future enhancements.

```
ConstraintExpr  : Expr
                  ;
Expr              :    Expr "or" Expr
                  |    Expr "and" Expr
                  |    Expr "xor" Expr
                  |    '(' Expr ')'
                  |    NumExpr Op NumExpr
                  |    StrExpr Op StrExpr
                  |    NumExpr Op StrExpr
                  ;
NumExpr           :    NumExpr "+" NumTerm
                  |    NumExpr "-" NumTerm
                  |    NumTerm
                  ;
NumTerm           :    NumFactor
                  |    NumTerm "*" NumFactor
                  |    NumTerm "/" NumFactor
                  ;
NumFactor         :    Num
                  |    Identifier
                  |    '(' NumExpr ')'
                  |    '-' NumFactor
                  ;
StrExpr           :    StrTerm
                  |    StrExpr "+" StrTerm
                  ;
StrTerm           :    String
                  |    '(' StrExpr ')'
                  ;
                  ;
Op                :    "==" | "<=" | ">=" | "!=" | "<" | ">"
                  ;
Identifier        :    Word
                  ;
Word              :    Letter { AlphaNum }+
                  ;
AlphaNum          :    Letter
                  |    Digit
                  |    "_"
                  ;
String            :    "'" { Char }* "'"
                  ;
Num               :    { Digit}+
                  |    { Digit}+ "." { Digit}*
                  ;
Char              :    Letter
                  |    Digit
                  |    Other
                  ;
```

```
Letter             :    a | b | c | d | e | f | g | h | i
                   |    j | k | l | m | n | o | p | q | r
                   |    s | t | u | v | w | x | y | z | A
                   |    B | C | D | E | F | G | H | I | J
                   |    K | L | M | N | O | P | Q | R | S
                   |    T | U | V | W | X | Z
                   ;
Digit              :    0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9
                   ;
Other              :    <Sp> | ~ | ! | @ | # | $ | % | ^ | &
                   |    * | ( | ) | - | _ | = | + | [ | {
                   |    ] | } | ; | : | " | \ | | | , | <
                   |    . | > | / | ?
                   ;
Sp                 :    " "
                   ;
```

The following precedence relations hold in the absence of parentheses, in the order of lowest to highest:

- *or* and *xor*
- *and*
- *not*
- + and -
- * and /
- Otherwise, left-to-right precedence

The following are some example constraints:

```
(1) name == 'ashoo'
(2) name == 'ashoo' and pet == 'flakes'
(3) Fee <= 5 or LowFreq >= 20
(4) DeviceType == 'Car' and Cost < 30000 and color == 'white'
     and Year > 1990
```

Index

Printed in U.S.A.

.