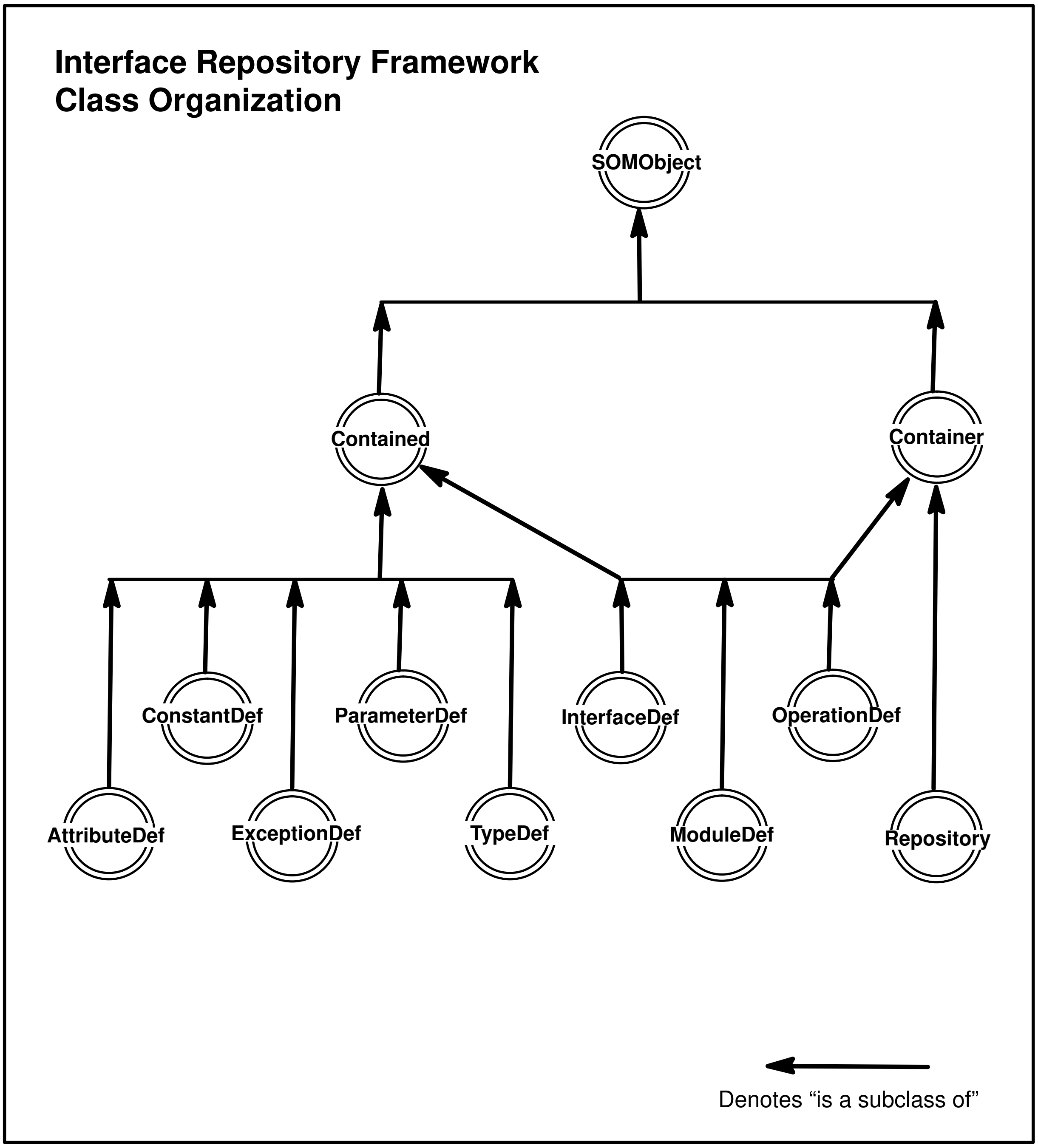
Interface Repository Framework Reference



Interface Repository Framework Class Organization

AttributeDef Class

Description

The **AttributeDef** class provides the interface for **attribute** definitions in the Interface Repository.

File Stem

attribdf

Base

Contained

Metaclass

SOMClass

Ancestor Classes

Contained, SOMObject

Types

```
enum AttributeMode {NORMAL, READONLY};

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

The **describe** method, inherited from **Contained,** returns an **AttributeDescription** structure in the “value” member of the **Description** structure (defined in the **Contained** class).

Attributes

Listed below is each available attribute, with its corresponding type in parentheses, followed by a description of its purpose:

type (TypeCode)

The **TypeCode** that represents the type of the **attribute**.
The **TypeCode** returned by the “_get_” form of the **type** attribute is contained in the receiving **AttributeDef** object, which retains ownership. Thus, the returned **TypeCode** should not be freed. To obtain a separate copy, use the **TypeCode_copy** operation. The “_set_” form of the attribute makes a private copy of the **TypeCode** you supply, to keep in the receiving object. You retain ownership of the passed **TypeCode**.

mode (AttributeMode)

The **AttributeMode** of the **attribute** (NORMAL or READONLY).

New Methods

None.

Overriding Methods

somInit
somUninit
somDumpSelf
somDumpSelfInt
describe

ConstantDef Class

Description

The **ConstantDef** class provides the interface for **constant** definitions in the Interface Repository.

File Stem

constdef

Base

Contained

Metaclass

SOMClass

Ancestor Classes

Contained, SOMObject

Types

```
struct ConstantDescription  {
     Identifier      name;
     RepositoryId  id;
     RepositoryId  defined_in;
     TypeCode       type;
     any                value;
};
```

The **describe** method, inherited from **Contained,** returns a **ConstantDescription** structure in the “value” member of the **Description** structure (defined in the **Contained** class).

Attributes

Listed below is each available attribute, with its corresponding type in parentheses, followed by a description of its purpose:

type (TypeCode)

The **TypeCode** that represents the type of **constant**.
The **TypeCode** returned by the “_get_” form of the **type** attribute is contained in the receiving **ConstantDef** object, which retains ownership. Thus, the returned **TypeCode** should not be freed. To obtain a separate copy, use the **TypeCode_copy** operation. The “_set_” form of the attribute makes a private copy of the **TypeCode** you supply, to keep in the receiving object. You retain ownership of the passed **TypeCode**.

value (any)

The value of the **constant**.

New Methods

None.

Overriding Methods

somInit
somUninit
somDumpSelf
somDumpSelfInt
describe

Contained Class

Description

The **Contained** class is the most generic form of interface for objects in SOM's CORBA-compliant Interface Repository (IR). All objects contained in the IR inherit this interface.

File Stem

containd

Base

SOMObject

Metaclass

SOMClass

Ancestor Classes

SOMObject

Types

```
typedef string RepositoryId;
struct Description {
    Identifier name;
    any    value;
};
```

Attributes

All attributes of the **Contained** class provide access to information kept within the receiving object. The "_get_" form of the attribute returns a memory reference that is only valid as long as the receiving object has not been freed (using **_somFree**). The "_set_" form of the attribute makes a (deep) copy of your data and places it in the receiving object. You retain ownership of all memory references passed using the "_set_" attributes.

Listed below is each available attribute, with its corresponding type in parentheses, followed by a description of its purpose:

name (Identifier)
: A simple name that identifies the **Contained** object within its containment hierarchy. The name may not be unique outside of the containment hierarchy; thus it may require qualification by **ModuleDef** name and/or **InterfaceDef** name.

id (RepositoryId)
: The value of the "id" field of the **Contained** object.
This is a string that uniquely identifies any object in the IR; thus it needs no qualification. Note that **RepositoryId**s have no relationship to the SOM type **somId.**

defined_in (RepositoryId)
: The value of the "defined_in" field of the **Contained** object.
This ID uniquely identifies the container where the **Contained** object is defined. Objects without global scope that do not appear within any other object are, by default, placed in the **Repository** object.

somModifiers (sequence<somModifier>)
: The **somModifiers** attribute is a sequence containing all modifiers associated with the object in the "implementation" section of the SOM IDL file where the receiving object is

defined. Note: This attribute is a SOM-unique extension of the Interface Repository; it is not stipulated by the CORBA specification.

New Methods

within
describe

Overriding Methods

somFree
somInit
somUninit
somDumpSelf
somDumpSelfInt

describe Method

Purpose

Returns a structure containing information defined in the IDL specification that corresponds to a specified **Contained** object in the Interface Repository.

IDL Syntax

Description describe ( );

Description

The **describe** method returns a structure containing information defined in the IDL specification of a **Contained** object. The specified object represents a component of an IDL interface (class) definition maintained within the Interface Repository.

When finished using the information in the returned **Description** structure, the client code must release the storage allocated for it. To free the associated storage, use a call similar to this:

```
if (desc.value._value)
    SOMFree (desc.value._value);
```

Caution: The **describe** method returns pointers to elements within objects (for example, **name**). Thus, the **somFree** method should *not* be used to release any of these objects while the **describe** information is still needed.

Parameters

receiver	A pointer to the **Contained** object in the Interface Repository for which a **Description** is needed.
ev	A pointer to the **Environment** structure for the caller.

Return Value

The **describe** method returns a structure of type **Description** containing information defined in the IDL specification of the receiving object.

The “name” field of the **Description** is the name of the type of description. The “name” values are from the following set:
{“ModuleDescription”, “InterfaceDescription”, “AttributeDescription”, “OperationDescription”, “ParameterDescription”, “TypeDescription”, “ConstantDescription”, “ExceptionDescription”}

The “value” field is a structure of type **any** whose “_value” field is a pointer to a structure of the type named by the “name” field of the **Description.** This structure provides all of the information contained in the IDL specification of the *receiver*. For example, if the **describe** method is invoked on an object of type **AttributeDef,** the “name” field of the returned **Description** will contain the identifier “AttributeDescription” and the “value” field will contain an **any** structure whose “_value” field is a pointer to an **AttributeDescription** structure.

Example

Here is a code fragment written in C that uses the **describe** method:

```
#include <containd.h>
#include <attribdf.h>
#include <somtc.h>

. . .
AttributeDef attr; /* An AttributeDef object (also a Contained) */
Description desc; /* .value field will be an AttributeDescription */
AttributeDescription *ad;
Environment *ev;

. . .
```

```
desc = Contained_describe (attr, ev);
ad = (AttributeDescription *) desc.value._value;
printf ("Attribute name: %s, defined in: %s\n",
         ad->name, ad->defined_in);
printf ("Attribute type: ");
TypeCode_print (ad->type, ev);
printf ("Attribute mode: %s\n", ad->mode == AttributeDef_READONLY ?
        "READONLY" : "NORMAL");
SOMFree (desc.value._value); /* Finished with describe output */
SOMObject_somFree (attr);    /* Finished with AttributeDef object */
```

Original Class

Contained

Related Information

Methods: within

within Method

Purpose

Returns a list of objects (in the Interface Repository) that contain a specified **Contained** object.

IDL Syntax

sequence<Container> within ( );

Description

The **within** method returns a sequence of objects within the Interface Repository that contain the specified **Contained** object. If the receiving object is an **InterfaceDef** or **ModuleDef**, it can only be contained by the object that defines it. Other objects can be contained by objects that define or inherit them.

If the object is global in scope, the sequence returned by **within** will have its **_length** field set to zero.

When finished using the sequence returned by this method, the client code is responsible for releasing each of the **Container**s in the sequence and freeing the sequence buffer. In C, this can be accomplished as follows:

```
if (seq._length) {
    long i;
    for (i=0; i<seq._length; i++)
        _somFree (seq._buffer[i]); /* Release each Container obj */
    SOMFree (seq._buffer);        /* Release the sequence buffer */
}
```

Parameters

receiver	A pointer to a **Contained** object for which containing objects are needed**.**
ev	A pointer to the **Environment** structure for the caller.

Return Value

The **within** method returns a sequence of **Container** objects that contain the specified **Contained** object.

Example

Here is a code fragment written in C that uses the **within** method:

```
#include <containd.h>
#include <containr.h>

. . .
Contained anObj;
Environment *ev;
sequence(Container) sc;
long i;
. . .
sc = Contained_within (anObj, ev);
printf ("%s is contained in (or inherited by):\n",
     Contained__get_name (anObj, ev));
for (i=0; i<sc._length; i++) {
     printf ("\t%s\n",
          Contained__get_name ((Contained) sc._buffer[i], ev));
     SOMObject_somFree (sc._buffer[i]);
}
if (sc._length)
     SOMFree (sc._buffer);
```

Original Class

Contained

Related Information

Methods: describe

Container Class

Description

The **Container** class is a generic interface that is common to all of the SOM CORBA-compliant Interface Repository (IR) objects that can hold or contain other objects. A **Container** object can be one of three types: **ModuleDef, InterfaceDef,** or **OperationDef.**

File Stem

containr

Base

SOMObject

Metaclass

SOMClass

Ancestor Classes

SOMObject

Types

```
typedef string InterfaceName;
// Valid values for InterfaceName are limited to the following set:
//   {"AttributeDef", "ConstantDef", "ExceptionDef", "InterfaceDef",
//     "ModuleDef", "ParameterDef", "OperationDef", "TypeDef", "all"}

struct ContainerDescription {
    Contained *contained_object;
    Identifier name;
    any  value;
};
```

New Methods

contents
lookup_name
describe_contents

Overriding Methods

somInit
somUninit
somDumpSelf
somDumpSelfInt

contents Method

Purpose

Returns a sequence indicating the objects contained within a specified **Container** object of the Interface Repository.

IDL Syntax

sequence<Contained> contents (
in InterfaceName *limit_type***,**
in boolean *exclude_inherited***);**

Description

The **contents** method returns a list of objects contained by the specified **Container** object. Each object represents a component of an IDL interface (class) definition maintained within the Interface Repository.

The **contents** method is used to navigate through the hierarchy of objects within the Interface Repository: Starting with the **Repository** object, this method can list all of the objects in the Repository, then all of the objects within the **ModuleDef** objects, then all within the **InterfaceDef** objects, and so on.

If the *"limit_type"* is set to "all", objects of all interface types are returned; otherwise, only objects of the requested interface type are returned. Valid values for **InterfaceName** are limited to the following set:

{"AttributeDef", "ConstantDef", "ExceptionDef", InterfaceDef",
"ModuleDef", "ParameterDef", "OperationDef", "TypeDef", "all"}

If *"exclude_inherited"* is set to TRUE, any inherited objects will *not* be returned.

When finished using the sequence returned by this method, the client code is responsible for releasing each of the objects in the sequence and freeing the sequence buffer. In C, this can be accomplished as follows:

```
if (seq._length) {
     long i;
     for (i=0; i<seq._length; i++)
        SOMObject_somFree (seq._buffer[i]); /* Release each object */
     SOMFree (seq._buffer);                 /* Release the buffer  */
}
```

Parameters

receiver A pointer to a **Container** object whose contained objects are needed**.**

ev A pointer to the **Environment** structure for the caller.

limit_type The name of one interface type (see the valid list above) or "all", to specify what type of objects the **contents** method should search for.

exclude_inherited
A **boolean** value: TRUE to exclude any inherited objects, or FALSE to include all objects.

Return Value

The **contents** method returns a sequence of pointers to objects contained within the specified **Container** object.

Example

Here is a code fragment written in C that uses the **contents** method:

```
#include <containr.h>

...

Container anObj;
Environment *ev;
sequence(Contained) sc;
long i;

...

sc = Container_contents (anObj, ev, "all", TRUE);
printf ("%s contains the following objects:\n",
    SOMObject_somIsA (anObj, _Contained) ?
        Contained__get_name ((Contained) anObj, ev) :
        "The Interface Repository");
for (i=0; i<sc._length; i++) {
    printf ("\t%s\n",
        Contained__get_name (sc._buffer[i], ev));
    SOMObject_somFree (sc._buffer[i]);
}
if (sc._length)
    SOMFree (sc._buffer);
else
    printf ("\t[none]\n");
```

Original Class

Container

Related Information

Methods: lookup_name, describe_contents

describe_contents Method

Purpose

Returns a sequence of descriptions of the objects contained within a specified **Container** object of the Interface Repository.

IDL Syntax

sequence<ContainerDescription> describe_contents (
in InterfaceName *limit_type***,**
in boolean *exclude_inherited***,**
in long *max_returned_objs***);**

Description

The **describe_contents** method combines the operations of the **contents** method and the **describe** method. That is, for each object returned by the **contents** operation, the description of the object is returned by invoking its **describe** operation. Each object represents a component of an IDL interface (class) definition maintained within the Interface Repository.

If the *"limit_type"* is set to "all", objects of all interface types are returned; otherwise, only objects of the requested interface type are returned. Valid values for **InterfaceName** are limited to the following set:

{"AttributeDef", "ConstantDef", "ExceptionDef", "InterfaceDef", "ModuleDef", "ParameterDef", "OperationDef", "TypeDef", "all"}

If *"exclude_inherited"* is set to TRUE, any inherited objects will *not* be returned.

The *"max_returned_objs"* argument is used to limit the number of objects that can be returned. If *"max_returned_objs"* is set to –1, the results for all contained objects will be returned.

When finished using the sequence returned by this method, the client code is responsible for freeing the "value._value" field in each description, releasing each of the objects in the sequence, and freeing the sequence buffer. In C, this can be accomplished as follows:

```
if (seq._length) {
     long i;
     for (i=0; i<seq._length; i++) {
          if (seq._buffer[i].value._value)
                                       /* Release each description */
               SOMFree (seq._buffer[i].value._value);
          SOMObject_somFree (seq._buffer[i].contained_object);
                                          /* Release each object */
     }
     SOMFree (seq._buffer);               /* Release the buffer  */
}
```

Parameters

receiver
A pointer to a **Container** object whose contained object descriptions are needed**.**

ev
A pointer to the **Environment** structure for the caller.

limit_type
The name of one interface type (see the valid list above) or "all", to specify what type of objects the **describe_contents** method should return.

exclude_inherited
A **boolean** value: TRUE to exclude any inherited objects, or FALSE to include all objects.

max_returned_objs
A **long** integer indicating the maximum number of objects to be returned by the method, or –1 to indicate no limit is set.

Return Value

The **describe_contents** method returns a sequence of **ContainerDescription** structures, one for each object contained within the specified **Container** object. Each **ContainerDescription** structure has a “contained_object” field, which points to the contained object, as well as “name” and “value” fields, which are the result of the **describe** method.

Example

Here is a code fragment written in C that uses the **describe_contents** method:

```
#include <containr.h>

...

Container anObj;
Environment *ev;
sequence(ContainerDescription) sc;
long i;

...

sc = Container_describe_contents (anObj, ev, ”all”, FALSE, –1L);
printf (”%s defines or inherits the following objects:\n”,
    SOMObject_somIsA (anObj, _Contained) ?
        Contained__get_name ((Contained) anObj, ev) :
        ”The Interface Repository”);
for (i=0; i<sc._length; i++) {
    printf (”\t%s\n”, sc._buffer[i].name);
    if (sc._buffer[i].value._value)
        SOMFree (sc._buffer[i].value._value);
    SOMObject_somFree (sc._buffer[i].contained_object);
}
if (sc._length)
    SOMFree (sc._buffer);
else
    printf (”\t[none]\n”);
```

Original Class

Container

Related Information

Methods: contents, describe, lookup_name

lookup_name Method

Purpose

Locates an object by name within a specified **Container** object of the Interface Repository, or within objects contained in the **Container** object.

IDL Syntax

sequence<Contained> lookup_name (
in Identifier *search_name***,**
in long *levels_to_search***,**
in InterfaceName *limit_type***,**
in boolean *exclude_inherited***);**

Description

The **lookup_name** method locates an object by name within a specified **Container** object, or within objects contained in the **Container** object. The *"search_name"* specifies the name of the object to be found. Each object represents a component of an IDL interface (class) definition maintained within the Interface Repository.

The *"levels_to_search"* argument controls whether the lookup is constrained to the specified **Container** object or whether objects contained within the **Container** object are also searched. The *"levels_to_search"* value should be –1 to search the **Container** and all contained objects; it should be 1 to search only the **Container** itself.

If *"limit_type"* is set to "all", the lookup locates an object of the specified name with any interface type; otherwise, the search locates the object only if it has the designated interface type. Valid values for **InterfaceName** are limited to the following set:

{"AttributeDef", "ConstantDef", "ExceptionDef", "InterfaceDef",
"ModuleDef", "ParameterDef", "OperationDef", "TypeDef", "all"}

If *"exclude_inherited"* is set to TRUE, any inherited objects will *not* be returned.

When finished using the sequence returned by this method, the client code is responsible for releasing each of the objects in the sequence and freeing the sequence buffer. In C, this can be accomplished as follows:

```
if (seq._length) {
     long i;
     for (i=0; i<seq._length; i++)
         SOMObject_somFree (seq._buffer[i]);
                                         /* Release each object */
     SOMFree (seq._buffer);              /* Release the buffer  */
     }
```

Parameters

receiver — A pointer to a **Container** object in which to locate the object.

ev — A pointer to the **Environment** structure for the caller.

search_name — The name of the object to be located.

levels_to_search — A long having the value 1 or –1.

limit_type — The name of one interface type (see the valid list above) or "all", to specify what type of object to search for.

exclude_inherited — A **boolean** value: TRUE to exclude an object when it is inherited, or FALSE to return the object from wherever it is found.

Return Value

The **lookup_name** method returns a sequence of pointers to objects of the given name contained within the specified **Container** object, or within objects contained in the **Container** object.

Example

Here is a code fragment written in C that uses the **lookup_name** method:

```
#include <containr.h>
#include <containd.h>
#include <repostry.h>

...

Container repo;
Environment *ev;
sequence(Contained) sc;
long i;
Identifier nameToFind;

...

repo = (Container) RepositoryNew ();
sc = Container_lookup_name (repo, ev, nameToFind, -1, "all", TRUE);
printf ("%d object%s found:\n",
         sc._length, sc._length == 1 ? "" : "s");
for (i=0; i<sc._length; i++) {
    printf ("\t%s\n",
        Contained__get_id (sc._buffer[i], ev));
    SOMObject_somFree (sc._buffer[i]);
}
if (sc._length)
    SOMFree (sc._buffer);
```

Original Class

Container

Related Information

Methods: contents, describe_contents

ExceptionDef Class

Description

The **ExceptionDef** class provides the interface for **exception** definitions in the Interface Repository.

File Stem

excptdef

Base

Contained

Metaclass

SOMClass

Ancestor Classes

Contained, SOMObject

Types

```
struct ExceptionDescription {
    Identifier      name;
    RepositoryId    id;
    RepositoryId    defined_in;
    TypeCode        type;
};
```

The **describe** method, inherited from **Contained,** returns an **ExceptionDescription** structure in the "value" member of the **Description** structure (defined in the **Contained** class).

Attributes

Listed below is each available attribute, with its corresponding type in parentheses, followed by a description of its purpose:

type (TypeCode)

The **TypeCode** that represents the type of the **exception**.
The **TypeCode** returned by the "_get_" form of the **type** attribute is contained in the receiving **ExceptionDef** object, which retains ownership. Thus the returned **TypeCode** should not be freed. To obtain a separate copy, use the **TypeCode_copy** operation. The "_set_" form of the attribute makes a private copy of the **TypeCode** you supply, to keep in the receiving object. You retain ownership of the passed **TypeCode.**

New Methods

None.

Overriding Methods

somInit
somUninit
somDumpSelf
somDumpSelfInt
describe

InterfaceDef Class

Description

The **InterfaceDef** class provides the interface for **interface** definitions in the Interface Repository.

File Stem

intfacdf

Base

Contained, Container

Metaclass

SOMClass

Ancestor Classes

Contained, Container, SOMObject

Types

```
struct FullInterfaceDescription {
    Identifier        name;
    RepositoryId      id;
    RepositoryId      defined_in;
    sequence<OperationDef::OperationDescription>  operation;
    sequence<AttributeDef::AttributeDescription>  attributes;
};

struct InterfaceDescription {
    Identifier        name;
    RepositoryId      id;
    RepositoryId      defined_in;
};
```

The **describe** method, inherited from **Contained,** returns an **InterfaceDescription** structure in the "value" member of the **Description** structure (defined in the **Contained** class).
The **describe_contents** method, inherited from **Container,** returns a sequence of these **Description** structures, each carrying a reference to an **InterfaceDescription** structure in its "value" member.

Implementation note: The two sequences "OperationDescription" and "AttributeDescription" are built dynamically within the **FullInterfaceDescription** structure, due to the **InterfaceDef** class's inheritance from the **Contained** class.

Attributes

All attributes of the **InterfaceDef** class provide access to information kept within the receiving **InterfaceDef** object. The "_get_" form of the attribute returns a memory reference that is only valid as long as the receiving object has not been freed (using **_somFree**). The "_set_" form of the attribute makes a (deep) copy of your data and places it in the receiving **InterfaceDef** object. You retain ownership of all memory references passed using the "_set_" attribute forms.

Listed below is each available attribute, with its corresponding type in parentheses, followed by a description of its purpose:

base_interfaces (sequence<RepositoryId>)
The sequence of **RepositoryId**s for all of the interfaces that the receiving interface inherits.

instanceData (TypeCode)
The **TypeCode** of a structure whose members are the internal instance variables, if any, described in the SOM **implementation** section of the interface. Note: This attribute is a SOM-unique extension of the Interface Repository; it is not stipulated by the CORBA specifications.

New Methods

describe_interface

Overriding Methods

somInit
somUninit
somDumpSelf
somDumpSelfInt
within
describe

describe_interface Method

Purpose

Returns (from the Interface Repository) a description of all the methods and attributes of an interface definition.

IDL Syntax

FullInterfaceDescription describe_interface ( );

Description

The **describe_interface** method returns a description of all the methods and attributes of an interface definition that are held in the Interface Repository.

When finished using the **FullInterfaceDescription** returned by this method, the client code is responsible for freeing the _buffer fields of the two sequences it contains. In C, this can be accomplished as follows:

```
if (fid.operation._length)
    SOMFree (fid.operation._buffer);        /* Release the buffer */
if (fid.attributes._length)
    SOMFree (fid.attributes._buffer);       /* Release the buffer */
```

Parameters

receiver A pointer to an object of class **InterfaceDef** representing the Interface Repository object where an interface definition is stored.

ev A pointer where the method can return exception information if an error is encountered.

Return Value

The **describe_interface** method returns a description of all the methods and attributes of an interface definition that are held in the Interface Repository.

Example

Here is a code fragment written in C that uses the **describe_interface** method:

```
#include <intfacdf.h>

...

InterfaceDef idef;
Environment *ev;
FullInterfaceDescription fid;
long i;

...

fid = InterfaceDef_describe_interface (idef, ev);
printf ("The %s interface has the following attributes:\n",
    Contained__get_name ((Contained) idef, ev));
if (!fid.attributes._length)
    printf ("\t[none]\n");
else {
    for (i=0; i<fid.attributes._length; i++)
        printf ("\t%s\n", fid.attributes._buffer[i].name);
    SOMFree (fid.attributes._buffer);
}
```

```
printf ("and the following methods:\n")
if (!fid.operation._length)
    printf ("\t[none]\n");
else {
    for (i=0; i<fid.operation._length; i++)
        printf ("\t%s\n", fid.operation._buffer[i].name);
    SOMFree (fid.operation._buffer);
}
```

Original Class

InterfaceDef

ModuleDef Class

Description

The **ModuleDef** class provides the interface for **module** definitions in the Interface Repository.

File Stem

moduledf

Base

Contained, Container

Metaclass

SOMClass

Ancestor Classes

Contained, Container, SOMObject

Types

```
struct ModuleDescription  {
     Identifier        name;
     RepositoryId  id;
     RepositoryId  defined_in;
};
```

The **describe** method, inherited from **Contained**, returns a **ModuleDescription** structure in the “value” member of the **Description** structure (defined in the **Contained** class).
The **describe_contents** method, inherited from **Container,** returns a sequence of these **Description** structures, each carrying a reference to a **ModuleDescription** structure in its “value” member.

New Methods

None.

Overriding Methods

somInit
somUninit
somDumpSelf
somDumpSelfInt
describe
within

OperationDef Class

Description

The **OperationDef** class provides the interface for operation (method) definitions in the Interface Repository.

File Stem

operatdf

Base

Contained, Container

Metaclass

SOMClass

Ancestor Classes

Contained, Container, SOMObject

Types

```
typedef Identifier ContextIdentifier;
enum OperationMode {NORMAL, ONEWAY};

struct OperationDescription {
    Identifier          name;
    RepositoryId        id;
    RepositoryId        defined_in;
    TypeCode            result;
    OperationMode       mode;
    sequence<ContextIdentifier> contexts;
    sequence<ParameterDef::ParameterDescription> parameter;
    sequence<ExceptionDef::ExceptionDescription> exceptions;
};
```

The **describe** method, inherited from **Contained,** returns an **OperationDescription** structure in the “value” member of the **Description** structure (defined in the **Contained** class).
The **describe_contents** method, inherited from **Container,** returns a sequence of these **Description** structures, each carrying a reference to an **OperationDescription** structure in its “value” member.

Attributes

Listed below is each available attribute, with its corresponding type in parentheses, followed by a description of its purpose:

result (TypeCode)
The **TypeCode** that represents the type of the operation (method).
The **TypeCode** returned by the “_get_” form of the **type** attribute is contained in the receiving **OperationDef** object, which retains ownership. Thus the returned **TypeCode** should not be freed. To obtain a separate copy, use the **TypeCode_copy** operation.
The “_set_” form of the attribute makes a private copy of the **TypeCode** you supply, to keep in the receiving object. You retain ownership of the passed **TypeCode.**

mode (OperationMode)
The **OperationMode** of the operation (method), either NORMAL or ONEWAY.

contexts (sequence<ContextIdentifier>)

The list of **ContextIdentifier**s associated with the operation (method).
The "_get_" form of the attribute returns a sequence whose buffer is owned by the receiving **OperationDef** object. You should not free it. The "_set_" form of the attribute makes a (deep) copy of the passed sequence; you retain ownership of the original storage.

New Methods

None.

Overriding Methods

somInit
somUninit
somDumpSelf
somDumpSelfInt
describe

ParameterDef Class

Description

The **ParameterDef** class provides the interface for **parameter** definitions in the Interface Repository.

File Stem

paramdef

Base

Contained

Metaclass

SOMClass

Ancestor Classes

Contained, SOMObject

Types

```
enum  ParameterMode  {IN, OUT, INOUT};

struct  ParameterDescription  {
     Identifier          name;
     RepositoryId        id;
     RepositoryId        defined_in;
     TypeCode            type;
     ParameterMode mode;
};
```

The **describe** method, inherited from **Contained,** returns a **ParameterDescription** structure in the "value" member of the **Description** structure (defined in the **Contained** class).

Attributes

Listed below is each available attribute, with its corresponding type in parentheses, followed by a description of its purpose:

type (TypeCode)

The **TypeCode** that represents the type of the **parameter**.
The **TypeCode** returned by the "_get_" form of the **type** attribute is contained in the receiving **ParameterDef** object, which retains ownership. Hence, the returned **TypeCode** should not be freed. To obtain a separate copy, use the **TypeCode_copy** operation. The "_set_" form of the attribute makes a private copy of the **TypeCode** you supply, to keep in the receiving object. You retain ownership of the passed **TypeCode.**

mode (ParameterMode)

The **ParameterMode** of the **parameter** (IN, OUT, or INOUT).

New Methods

None.

Overriding Methods

somInit
somUninit
somDumpSelf
somDumpSelfInt
describe

Repository Class

Description

The **Repository** class provides global access to SOM's CORBA-compliant Interface Repository (IR), which is discussed in Chapter 7, "The Interface Repository Framework," of the *SOM Toolkit User's Guide.*

File Stem

repostry

Base

Container

Metaclass

SOMClass

Ancestor Classes

Container, SOMObject

Types

```
struct RepositoryDescription {
    Identifier name;
    RepositoryId id;
    RepositoryId defined_in;
};
```

The inherited **describe_contents** method returns an instance of the **RepositoryDescription** structure in the "value" member of the **Description** structure defined in the **Container** interface.

New Methods

lookup_id
lookup_modifier
release_cache

Overriding Methods

describe_contents
somInit
somUninit
somFree
somDumpSelf
somDumpSelfInt

lookup_id Method

Purpose

Returns the object having a specified **RepositoryId**.

IDL Syntax

Contained lookup_id (
in RepositoryId *search_id***);**

Description

The **lookup_id** method returns the object having a **RepositoryId** given by the specified *search_id* argument. The returned object represents a component of an IDL interface (class) definition maintained within the Interface Repository.

When finished using the object returned by this method, the client code is responsible for releasing it, using the **somFree** method.

Parameters

receiver	A pointer to an object of class **Repository** representing SOM's Interface Repository**.**
ev	A pointer where the method can return exception information if an error is encountered.
search_id	An ID value of type **RepositoryId** that uniquely identifies the desired object in the Interface Repository.

Return Value

The **lookup_id** method returns the **Contained** object that has the specified **RepositoryId**.

Example

Here is a code fragment written in C that uses the **lookup_id** method:

```
#include <containd.h>
#include <repostry.h>

...

Repository repo;
Environment *ev;
Contained c;
RepositoryId objectToFind;

...

repo = RepositoryNew ();
c = Repository_lookup_id (repo, ev, objectToFind);
if (c) {
    printf ("lookup_id found object of type: %s, named: %s\n",
        SOMObject_somGetClassName (c), Contained__get_name (c, ev));
    SOMObject_somFree (c);
}
```

Original Class

Repository

Related Information

Methods: lookup_modifier, lookup_name, contents, within

lookup_modifier Method

Purpose

Returns the value of a given SOM **modifier** for a specified object [that is, for an object that is a component of an IDL interface (class) definition maintained within the Interface Repository].

IDL Syntax

string lookup_modifier (
in RepositoryId *id***,**
in string *modifier***);**

Description

The **lookup_modifier** method returns the string value of the given SOM **modifier** for an object with the specified **RepositoryId** within the Interface Repository. For a discussion of SOM modifiers, see the topic "Modifier statements" in Chapter 4, "SOM IDL and the SOM Compiler" in the *SOM Toolkit User's Guide.*

If the object with the given **RepositoryId** does not exist or does not possess the modifier, then NULL (or zero) is returned. If the object exists but the specified modifier does not have a value, a zero-length string value is returned.

Note: The **lookup_modifier** method is *not* stipulated by the CORBA specifications; it is a SOM-unique extension to the Interface Repository.

Parameters

receiver — A pointer to an object of class **Repository** representing SOM's Interface Repository.

ev — A pointer where the method can return exception information if an error is encountered.

id — The **RepositoryId** of the object whose modifier value is needed.

modifier — The name of a specific (SOM or user-specified) modifier whose string value is needed.

Return Value

The **lookup_modifier** method returns the string value of the given SOM **modifier** for an object with the specified **RepositoryId**, if it exists. If an existing modifier has no value, a zero-length string value is returned. If the object cannot be found, then NULL (or zero) is returned.

When the string value is no longer needed, client code must free the space for the string (using **SOMFree**).

Example

Here is a code fragment written in C that uses the **lookup_modifier** method:

```
#include <repostry.h>

...

Repository repo;
Environment *ev;
RepositoryId objectId;
string filestem;i

...

repo = RepositoryNew ();
filestem = Repository_lookup_modifier (repo, ev, objectId,
                                                  ”filestem”);
if (filestem) {
    printf
        (”The %s object’s filestem modifier has the value \”%s\”\n”,
            objectId, filestem);
    SOMFree (filestem);
} else
    printf (”No filestem modifier could be found for %s\n”,
            objectId);
```

Original Class

Repository

Related Information

Methods: lookup_id, lookup_name

release_cache Method

Purpose

Permits the Repository object to release the memory occupied by Interface Repository objects that have been implicitly referenced.

Syntax

void release_cache ( );

Description

This method allows the Repository object to release the memory occupied by implicitly referenced Interface Repository objects. Some methods (such as **describe_contents** and **lookup_name**) may cause some objects to be instantiated that are not directly accessible through object references that have been returned to the user. These objects are kept in an internal Interface Repository cache until the **release_cache** method is used to free them. The internal cache continuously replenishes itself over time as the need arises.

Parameters

receiver	A pointer to an object of class **Repository** representing SOM's Interface Repository.
ev	A pointer where the method can return exception information if an error is encountered.

Return Value

None.

Example

```
#include <repostry.h>

...

Repository repo;
Environment *ev;
sequence(ContainerDescription) scd;

...

scd = Container_describe_contents (
      (Container) repo, ev, "TypeDef", TRUE, -1);
Repository_release_cache (repo, ev);
```

Original Class

Repository

Related Information

See the section entitled "A word about memory management" in Chapter 7 of the *SOM Toolkit User's Guide.*

TypeDef Class

Description

The **TypeDef** class provides the interface for **typedef** definitions in the Interface Repository.

File Stem

typedef

Base

Contained

Metaclass

SOMClass

Ancestor Classes

Contained, SOMObject

Types

```
struct TypeDescription {
    Identifier    name;
    RepositoryId  id;
    RepositoryId  defined_in;
    TypeCode      type;
};
```

The **describe** method, inherited from **Contained,** returns a **TypeDescription** structure in the “value” member of the **Description** structure (defined in the **Contained** class).

Attributes

Listed below is each available attribute, with its corresponding type in parentheses, followed by a description of its purpose:

type (TypeCode)

The **TypeCode** that represents the type of the **typedef**.
The **TypeCode** returned by the “_get_” form of the **type** attribute is contained in the receiving **TypeDef** object, which retains ownership. Hence, the returned **TypeCode** should not be freed. To obtain a separate copy, use the **TypeCode_copy** operation.
The “_set_” form of the attribute makes a private copy of the **TypeCode** you supply, to keep in the receiving object. You retain ownership of the passed **TypeCode.**

New Methods

None.

Overriding Methods

somInit
somUninit
somDumpSelf
somDumpSelfInt
describe

TypeCode_alignment Function

Purpose

Supplies the alignment value for a given **TypeCode**.

IDL Syntax

short TypeCode_alignment ( );

Description

This function returns the alignment information associated with the given **TypeCode.** The alignment value is a short integer that should evenly divide any memory address where an instance of the type described by the **TypeCode** will occur.

Parameters

tc	The **TypeCode** whose alignment information is desired.
ev	A pointer to an Environment structure.

Return Value

A short integer containing the alignment value.

Related Information

Functions: TypeCodeNew, TypeCode_equal, TypeCode_kind, TypeCode_param_count, TypeCode_parameter, TypeCode_setAlignment, TypeCode_size, TypeCode_free, TypeCode_print

TypeCode_copy Function

Purpose

Creates a new copy of a given **TypeCode**.

IDL Syntax

TypeCode TypeCode_copy ( );

Description

The **TypeCode_copy** function creates a new copy of a given **TypeCode**. **TypeCode**s are complex data structures whose actual representation is hidden, and may contain internal references to **string**s and other **TypeCode**s. The copy created by this function is guaranteed not to refer to any previously existing **TypeCode**s or **string**s, and hence can be used long after the original **TypeCode** is freed or released (**TypeCode**s are typically contained in Interface Repository objects whose memory resources are released by the **_somFree** method).

All of the memory used to construct the **TypeCode** copy is allocated dynamically and should be subsequently freed *only* by using the **TypeCode_free** function.

This function is a SOM-unique extension to the CORBA standard.

Parameters

tc The **TypeCode** to be copied.

ev A pointer to an **Environment** structure. The CORBA standard mandates the use of this structure as a standard way to return exception information when an error condition is detected.

Return value

A new **TypeCode** with no internal references to any previously existing **TypeCode**s or **string**s. If a copy cannot be created successfully, the value NULL is returned. No exceptions are raised by this function.

Related Information

Functions: TypeCodeNew, TypeCode_alignment, TypeCode_equal, TypeCode_kind, TypeCode_param_count, TypeCode_parameter, TypeCode_size, TypeCode_free, TypeCode_print, TypeCode_setAlignment

TypeCode_equal Function

Purpose

Compares two **TypeCode**s for equality.

IDL Syntax

boolean TypeCode_equal (
TypeCode *tc2***);**

Description

The **TypeCode_equal** function can be used to determine if two distinct **TypeCode**s describe the same underlying abstract data type.

Parameters

tc — One of the **TypeCode**s to be compared.

ev — A pointer to an **Environment** structure. The CORBA standard mandates the use of this structure as a standard way to return exception information when an error condition is detected.

tc2 — The other **TypeCode** to be compared.

Return value

Returns TRUE (1) if the **TypeCode**s *tc* and *tc2* describe the same data type, with the same alignment. Otherwise, FALSE (0) is returned. No exceptions are raised by this function.

Related Information

Functions: TypeCodeNew, TypeCode_alignment, TypeCode_kind, TypeCode_param_count, TypeCode_parameter, TypeCode_copy, TypeCode_free, TypeCode_print, TypeCode_setAlignment, TypeCode_size

TypeCode_free Function

Purpose

Destroys a given **TypeCode** by freeing all of the memory used to represent it.

IDL Syntax

void TypeCode_free ( );

Description

The **TypeCode_free** function destroys a given **TypeCode** by freeing all of the memory used to represent it. **TypeCode**s obtained from the **TypeCode_copy** or **TypeCodeNew** functions should be freed using **TypeCode_free**. **TypeCode**s contained in Interface Repository objects should never be freed. Their memory is released when a **_somFree** method releases the Interface Repository object.

The **TypeCode_free** operation has no effect on **TypeCode** constants. **TypeCode** constants are static **TypeCode**s declared in the header file “somtcnst.h” or generated in files emitted by the SOM Compiler. Since **TypeCode** constants may be used interchangeably with dynamically created **TypeCode**s, it is *not* considered an error to attempt to free a **TypeCode** constant with the **TypeCode_free** function.

This function is a SOM-unique extension to the CORBA standard.

Parameters

tc — The **TypeCode** to be freed.

ev — A pointer to an **Environment** structure. The CORBA standard mandates the use of this structure as a standard way to return exception information when an error condition is detected.

Return value

None. No exceptions are raised by this function.

Related Information

Functions: TypeCodeNew, TypeCode_alignment, TypeCode_equal, TypeCode_kind, TypeCode_param_count, TypeCode_parameter, TypeCode_size, TypeCode_copy, TypeCode_print, TypeCode_setAlignment

TypeCode_kind Function

Purpose

Categorizes the abstract data type described by a **TypeCode**.

IDL Syntax

TCKind TypeCode_kind ( );

```
enum  TCKind {
        tk_null,  tk_void,
        tk_short,  tk_long,  tk_ushort,  tk_ulong,
        tk_float,  tk_double,  tk_boolean,  tk_char,
        tk_octet,  tk_any,  tk_TypeCode,  tk_Principal,
        tk_objref,  tk_struct,  tk_union,  tk_enum,  tk_string,
        tk_sequence,  tk_array,  tk_pointer,  tk_self, tk_foreign
      };
```

Description

The **TypeCode_kind** function can be used to classify a **TypeCode** into one of the categories listed in the **TCKind** enumeration. Based on the "kind" classification, a **TypeCode** may contain 0 or more additional parameters to fully describe the underlying data type.

Table 1 (see following page) indicates the number and function of these additional parameters. **TCKind** entries not listed in the table are basic data types and do not have any additional parameters. The designation "N" refers to the number of members in a **struct** or **union**, or the number of enumerators in an **enum**.

Parameters

tc The **TypeCode** whose **TCKind** categorization is requested.

ev A pointer to an **Environment** structure. The CORBA standard mandates the use of this structure as a standard way to return exception information when an error condition is detected.

Return value

Returns one of the enumerators listed in the **TCKind** enumeration shown above. No exceptions are raised by this function.

Related Information

Functions: TypeCodeNew, TypeCode_alignment, TypeCode_equal, TypeCode_param_count, TypeCode_parameter, TypeCode_copy, TypeCode_free, TypeCode_print, TypeCode_setAlignment, TypeCode_size

TCKind	Parameters	Type	Function
tk_objref	1	**string**	The ID of the corresponding **InterfaceDef** in the Interface Repository
tk_struct	2N+1	**string**	The name of the **struct**.
		——— next 2 repeat for each member ———	
		string	The name of the **struct** member.
		TypeCode	The type of the **struct** member.
tk_union	3N+2	**string**	The name of the **union**.
		TypeCode	The type of the discriminator.
		——— next 3 repeat for each member ———	
		long	The label value
		string	The name of the member.
		TypeCode	The type of the member.
tk_enum	N+1	**string**	The name of the **enum**.
		—— next repeats for each enumerator ——	
		string	The name of the enumerator.
tk_string	1	**long**	The maximum string length or 0.
tk_sequence	2	**TypeCode** **long**	The type of element in the sequence. The maximum number of elements or 0.
tk_array	2	**TypeCode** **long**	The type of element in the **array**. The maximum number of elements.
tk_pointer †	1	**TypeCode**	The type of the referenced datum.
tk_self †	1	**string**	The name of the referenced enclosing **struct** or **union**.
tk_foreign †	3	**string** **string** **long**	The name of the foreign type. The implementation context. The size of an instance.

Table 1. **TypeCode** information per **TCKind** category.

†The **TCKind** values **tk_pointer**, **tk_self**, and **tk_foreign** are SOM-unique extensions to the CORBA standard. They are provided to permit **TypeCode**s to describe types that cannot be expressed in standard IDL.

The **tk_pointer TypeCode** contains only one parameter — a **TypeCode** which describes the data type that the pointer references. The **tk_self TypeCode** is used to describe a "self-referential" structure or union without introducing unbounded recursion in the **TypeCode**. For example, the following C struct:

```
struct node {
   long count;
   struct node *next;
   };
```

could be described with a **TypeCode** created as follows:

```
TypeCode tcForNode;

tcForNode = TypeCodeNew (tk_struct, "node",
     "count", TypeCodeNew (tk_long),
     "next", TypeCodeNew (tk_pointer,
     TypeCodeNew (tk_self, "node")));
```

The **tk_foreign TypeCode** provides a more general escape mechanism, allowing **TypeCode**s to be created that partially describe non-IDL types. Since these foreign **TypeCode**s carry only a partial description of a type, the "implementation context" parameter can be used by a non-IDL execution environment to recognize other types that are known or understood in that environment. See the section entitled "Using the tk_foreign TypeCode" in Chapter 7 of the *SOM Toolkit User's Guide* for more information about using foreign **TypeCode**s in SOM IDL files.

Note that the use of self-referential structures, pointers, or foreign types is beyond the scope of the CORBA standard, and may result in a loss of portability or distributability in client code.

TypeCodeNew Function

Purpose

Creates a new **TypeCode** instance.

Syntax

TypeCode TypeCodeNew (TCKind *tag*, ...**);**

[The actual parameters indicated by "..." are variable in number and type, depending on the value of the *tag* parameter.] There are *no* implicit parameters to this function.

TypeCodeNew (tk_objref, string *interfaceId***);**
TypeCodeNew (tk_string, long *maxLength***);**
TypeCodeNew (tk_sequence, TypeCode *seqTC*, **long** *maxLength***);**
TypeCodeNew (tk_array, TypeCode *arrayTC*, **long** *length***);**
TypeCodeNew (tk_pointer, TypeCode *ptrTC***);**
TypeCodeNew (tk_self, string *structOrUnionName***);**
TypeCodeNew (tk_foreign, string *typename*, **string** *impCtx*, **long** *instSize***);**

TypeCodeNew (tk_struct, string *name*,
  string *mbrName*, **TypeCode** *mbrTC*, [...,]
  [*mbrName* and *mbrTC* repeat as needed]
  NULL);

TypeCodeNew (tk_union, string *name*, **TypeCode** *swTC*,
  long *flag*, **long** *labelValue*, **string** *mbrName*, **TypeCode** *mbrTC*, [...,]
  [*flag, labelValue, mbrName* and *mbrTC* repeat as needed]
  NULL);

TypeCodeNew (tk_enum, string *name*,
  string *enumId*, [...,]
  [*enumId*s repeat as needed]
  NULL);

TypeCodeNew (TCKind *allOtherTagValues***);**

Description

The **TypeCodeNew** function creates a new instance of a **TypeCode** from the supplied parameters. **TypeCode**s are complex data structures whose actual representation is hidden. The number and types of arguments required by **TypeCodeNew** varies depending on the value of the first argument. All of the valid invocation sequences are shown in the "Syntax" section above. There are *no* implicit parameters to this function.

All **TypeCode**s created by **TypeCodeNew** should be destroyed (when no longer needed) using the **TypeCode_free** function.

This function is a SOM-unique extension to the CORBA standard.

Parameters

tag — The type or category of **TypeCode** to create.

interfaceId — A string containing the fully-qualified interface name that is the subject of an object reference type.

name — A string that gives the name of a **struct, union,** or **enum**.

mbrName — A string that gives the name of a **struct** or **union** member element.

enumId — A string that gives the name of an **enum** enumerator.

structOrUnionName
A string that gives the name of a **struct** or **union** that has been previously named in the current **TypeCode** and is the subject of a self-referential pointer

type. See the footnote on **tk_self** in the table given in the **TypeCode_kind** function description for an example of what this means and how it is applied.

maxLength The maximum permitted length of a **string** or a **sequence**. The value 0 (zero) means that the **string** or **sequence** is considered unbounded.

length The maximum number of elements that can be stored in an array. All IDL arrays are bounded, hence a value of zero denotes an array of zero elements.

flag One of the following constant values used to distinguish a labeled case in an IDL discriminated **union** switch statement from the default case:

TCREGULAR_CASE – The value 1
TCDEFAULT_CASE – The value 2

labelValue The actual value associated with a regular labeled case in an IDL discriminated **union** switch statement. If preceded by the argument TCDEFAULT_CASE, the value zero should be used.

mbrTC A **TypeCode** that represents the data type of a **struct** or **union** member.

swTC A **TypeCode** that represents the data type of the discriminator in an IDL **union** statement.

seqTC A **TypeCode** that describes the data type of the elements in a **sequence**.

arrayTC A **TypeCode** that describes the data type of the elements of an **array**.

ptrTC A **TypeCode** that describes the data type referenced by a pointer.

typename A string that provides the name of a foreign type.

impCtx A string that identifies an implementation context where a foreign type is understood.

instSize A long that holds the size of a foreign type instance. If the size is variable or is not known, the value zero should be used.

allOtherTagValues
One of the values:
tk_null, tk_void, tk_short, tk_long,
tk_ushort, tk_ulong, tk_float, tk_double,
tk_boolean, tk_char, tk_octet,
tk_any, tk_TypeCode, or **tk_Principal**
All of these tags represent basic IDL data types that do not require any other descriptive parameters.

Return value

A new **TypeCode** instance, or NULL if the new instance could not be created.

Related Information

Functions: TypeCode_alignment, TypeCode_copy, TypeCode_equal, TypeCode_free, TypeCode_kind, TypeCode_param_count, TypeCode_parameter, TypeCode_print, TypeCode_size, TypeCode_setAlignment

TypeCode_param_count Function

Purpose

Obtains the number of parameters available in a given **TypeCode**.

IDL Syntax

long TypeCode_param_count ( );

Description

The **TypeCode_param_count** function can be used to obtain the actual number of parameters contained in a specified **TypeCode**. Each **TypeCode** contains sufficient parameters to fully describe its underlying abstract data type. Refer to the table given in the description of the **TypeCode_kind** function.

Parameters

tc	The **TypeCode** whose parameter count is desired.
ev	A pointer to an **Environment** structure. The CORBA standard mandates the use of this structure as a standard way to return exception information when an error condition is detected.

Return value

Returns the actual number of parameters associated with the given **TypeCode**, in accordance with the table shown in the **TypeCode_kind** description. No exceptions are raised by this function.

Related Information

Functions: TypeCodeNew, TypeCode_alignment, TypeCode_equal, TypeCode_kind, TypeCode_parameter, TypeCode_copy, TypeCode_free, TypeCode_print, TypeCode_size, TypeCode_setAlignment

TypeCode_parameter Function

Purpose

Obtains a specified parameter from a given **TypeCode**.

IDL Syntax

any TypeCode_parameter (
long *index***);**

Description

The **TypeCode_parameter** function can be used to obtain any of the parameters contained in a given **TypeCode**. Refer to the table shown in the description of the **TypeCode_kind** function for a list of the number and type of parameters associated with each category of **TypeCode**.

Note: This function should *not* be used for any of the IDL basic data types (that is, any types in the **TCKind** enumeration that are *not* listed in Table 1 under the **TypeCode_kind** function).

Parameters

tc	The **TypeCode** whose parameter is desired.
ev	A pointer to an **Environment** structure. The CORBA standard mandates the use of this structure as a standard way to return exception information when an error condition is detected.
index	The number of the desired parameter. Parameters are numbered from 0 to N–1, where N is the value returned by the **Typecode_param_count** function.

Return value

Returns the requested parameter in the form of an **any**. This function raises the Bounds exception if the value of the index exceeds the number of parameters available in the given **TypeCode**. Because the values exist within the specified **TypeCode**, you should not free the results returned from this function. A Bounds exception is also raised if this function is called on any of the IDL basic data types (see Note above).

An **any** is a basic IDL data type that is represented as the following structure in C or C++:

```
typedef struct any {
    TypeCode _type;
    void *   _value;
} any;
```

Since all **TypeCode** parameters have one of only three types (**string**, **TypeCode**, or **long**), the **_type** member will always be set to **TC_string**, **TC_TypeCode**, or **TC_long,** as appropriate. The **_value** member always points to the actual parameter datum. For example, the following code can be used to extract the name of a structure from a **TypeCode** of kind **tk_struct** in C:

```
#include <repostry.h> /* Interface Repository class */
#include <typedef.h>  /* Interface Repository TypeDef class */
#include <somtcnst.h> /* TypeCode constants */
TypeCode x;
Environment *ev = somGetGlobalEnvironment ();
TypeDef aTypeDefObj;
sequence(Contained) sc;
any parm;
string name;
Repository repo;

...
```

```
/* 1st, obtain a TypeCode from an Interface Repository object,
 * or use a TypeCode constant.
 */
repo = RepositoryNew ();
sc = _lookup_name (repo, ev,
    "AttributeDescription", -1, "TypeDef", TRUE);
if (sc._length) {
    aTypeDefObj = sc._buffer[0];
    x = __get_type (aTypeDefObj, ev);
    }
else
    x = TC_AttributeDescription;
if (TypeCode_kind (x, ev) == tk_struct) {
    parm = TypeCode_parameter (x, ev, 0); /* Get structure name */
    if (TypeCode_kind (parm._type, ev) != tk_string) {
        printf ("Error, unexpected TypeCode: ");
        TypeCode_print (parm._type, ev);
    } else {
        name = *((string *)parm._value);
        printf ("The struct name is %s\n", name);
    }
} else {
    printf ("TypeCode is not a tk_struct: ");
    TypeCode_print (x, ev);
}
```

Related Information

Functions: TypeCodeNew, TypeCode_alignment, TypeCode_equal, TypeCode_kind, TypeCode_param_count, TypeCode_copy, TypeCode_free, TypeCode_print, TypeCode_size, TypeCode_setAlignment

TypeCode_print Function

Purpose

Writes all of the information contained in a given **TypeCode** to “stdout”.

IDL Syntax

void TypeCode_print ( );

Description

The **TypeCode_print** function can be used during program debugging to inspect the contents of a **TypeCode**. It prints (in a human-readable format) all of the information contained in the **TypeCode**. The format of the information shown by **TypeCode_print** is the same form that could be used by a C programmer to code the corresponding **TypeCodeNew** function call to create the **TypeCode**.

This function is a SOM-unique extension to the CORBA standard.

Parameters

tc — The **TypeCode** to be examined.

ev — A pointer to an **Environment** structure. The CORBA standard mandates the use of this structure as a standard way to return exception information when an error condition is detected.

Return value

None. No exceptions are raised by this function.

Related Information

Functions: TypeCodeNew, TypeCode_alignment, TypeCode_equal, TypeCode_kind, TypeCode_param_count, TypeCode_parameter, TypeCode_copy, TypeCode_free, TypeCode_size, TypeCode_setAlignment

TypeCode_setAlignment Function

Purpose

Overrides the default alignment value associated with a given **TypeCode.**

IDL Syntax

void TypeCode_setAlignment (short *alignment***);**

Description

The **TypeCode_setAlignment** function overrides the default alignment value associated with a given **TypeCode.**

Parameters

tc	The **TypeCode** to receive the new alignment value.
ev	A pointer to an Environment structure.
alignment	A short integer that specifies the alignment value.

Return Value

None.

Related Information

Functions: TypeCodeNew, TypeCode_alignment, TypeCode_equal, TypeCode_kind, TypeCode_param_count, TypeCode_parameter, TypeCode_size, TypeCode_free, TypeCode_print

TypeCode_size Function

Purpose

Provides the minimum size of an instance of the abstract data type described by a given **TypeCode**.

IDL Syntax

long TypeCode_size ( );

Description

The **TypeCode_size** function is used to obtain the minimum size of an instance of the abstract data type described by a given **TypeCode**.

This function is a SOM-unique extension to the CORBA standard.

Parameters

tc The **TypeCode** whose instance size is desired.

ev A pointer to an **Environment** structure. The CORBA standard mandates the use of this structure as a standard way to return exception information when an error condition is detected.

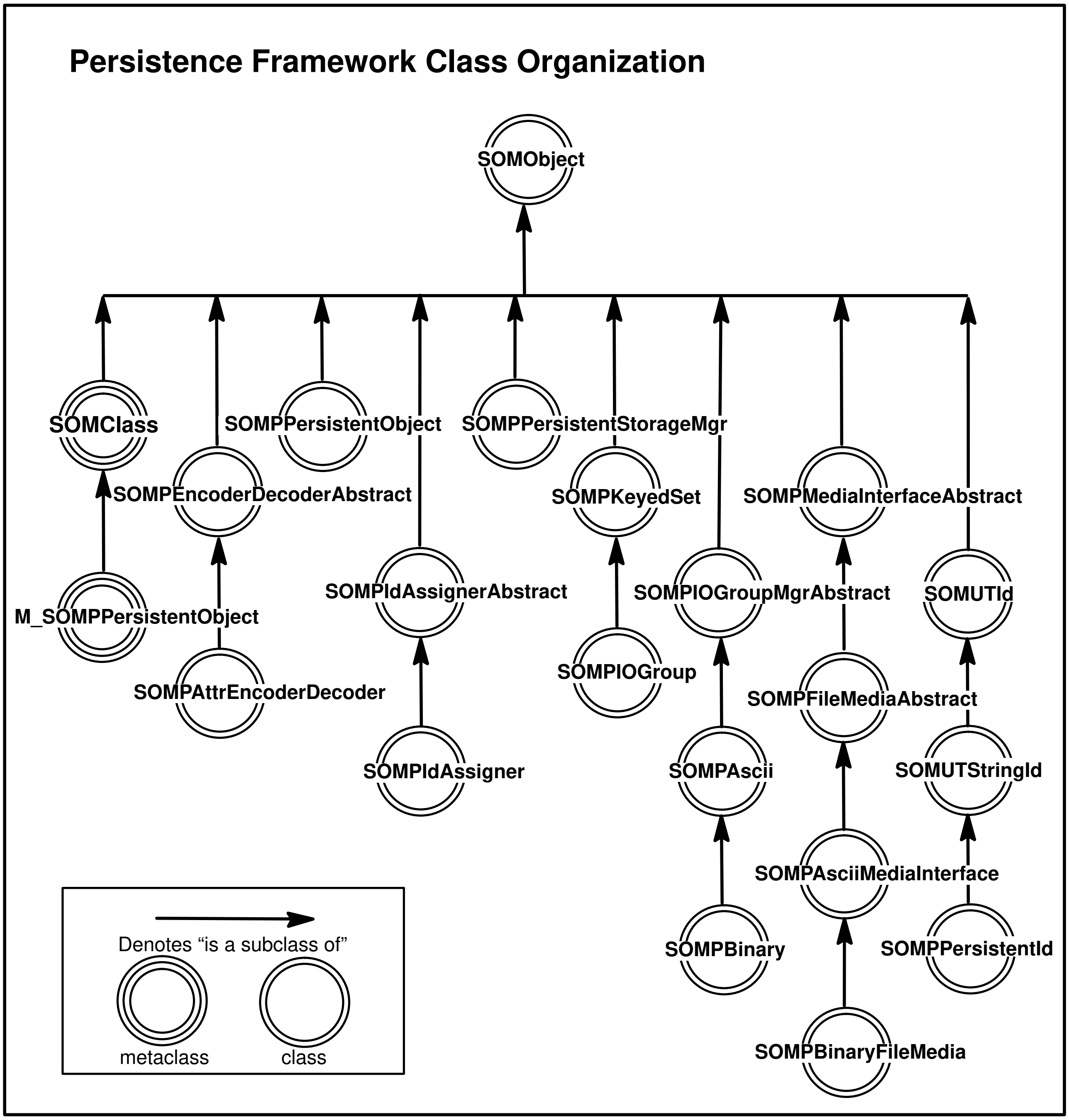
Return value

The amount of memory needed to hold an instance of the data type described by a given **TypeCode**. No exceptions are raised by this function.

Related Information

Functions: TypeCodeNew, TypeCode_alignment, TypeCode_equal, TypeCode_kind, TypeCode_param_count, TypeCode_parameter, TypeCode_copy, TypeCode_free, TypeCode_print, TypeCode_setAlignment

Persistence Framework Reference



M_SOMPPersistentObject Class

Description

The **M_SOMPPersistentObject** class is the metaclass of **SOMPPersistentObject**. This class keeps track of the class-level default Encoder/Decoder.

File Stem

po

Base

SOMClass

Metaclass

SOMClass

Ancestor Classes

SOMClass, **SOMObject**

New Methods

sompSetClassLevelEncoderDecoderName
sompGetClassLevelEncoderDecoderName

Overriding Methods

somInit

sompGetClassLevelEncoderDecoderName Method

Purpose

Gets the class-level default Encoder/Decoder class name.

IDL Syntax

string sompGetClassLevelEncoderDecoderName ( );

Description

The **sompGetClassLevelEncoderDecoderName** method gets the name of the class-level default Encoder/Decoder class. This method is typically used by implementors of I/O Group Managers.

Parameters

receiver	A pointer to an **M_SOMPPersistentObject** class object.
ev	A pointer to the **Environment** structure for the caller.

Return Value

The **sompGetClassLevelEncoderDecoderName** method returns the name of the class-level default Encoder/Decoder class.

Example

```
SOMPPersistentObject po;
string classEDName;
Environment *ev;
/* ... */
classEDName = _sompGetClassLevelEncoderDecoderName(_somGetClass(po),
                                                    ev);
```

Original Class

M_SOMPPersistentObject

Related Information

Methods: sompSetClassLevelEncoderDecoderName, sompGetEncoderDecoderName, sompSetEncoderDecoderName

sompSetClassLevelEncoderDecoderName Method

Purpose

Sets the class-level default Encoder/Decoder class name.

IDL Syntax

void sompSetClassLevelEncoderDecoderName (
in string *myName***);**

Description

The **sompSetClassLevelEncoderDecoderName** method sets the name of the class-level default Encoder/Decoder class. This method is typically used by client programs.

Parameters

receiver	A pointer to an **M_SOMPPersistentObject** class object.
ev	A pointer to the **Environment** structure for the caller.
myName	A **string** representing the name of the Encoder/Decoder class.

Return Value

None.

Example

```
SOMPPersistentObject po;
Environment *myEnv;
string edClassName = ”myEncoderDecoder”;
/* ... */
_sompSetClassLevelEncoderDecoderName(
    _somGetClass(po), myEnv, edClassName);
```

Original Class

M_SOMPPersistentObject

Related Information

Methods: sompGetClassLevelEncoderDecoderName, sompGetEncoderDecoderName, sompSetEncoderDecoderName

SOMPAscii Class

Description

The **SOMPAscii** class is an I/O Group Manager that stores groups of objects to the file system using the media interface **SOMPAsciiMediaInterface.** This I/O Group Manager stores non-object data (for example, the number of objects in the group) in ASCII format.

File Stem

fsagm

Base

SOMPIOGroupMgrAbstract

Metaclass

SOMClass

Ancestor Classes

SOMPIOGroupMgrAbstract, SOMObject

New Methods

Overriding Methods

sompNewMediaInterface
sompGetMediaInterface
sompFreeMediaInterface
sompInstantiateMediaInterface
sompWriteGroup
sompReadGroup
sompReadObjectData
sompDeleteObjectFromGroup
sompGroupExists
sompObjectInGroup
sompMediaFormatOk
somInit
somUninit

SOMPAsciiMediaInterface Class

Description

The **SOMPAsciiMediaInterface** class implements an object store based on a file system. The numeric data it writes is in ASCII format.

File Stem

fmi

Base

SOMPFileMediaAbstract

Metaclass

SOMClass

Ancestor Classes

SOMPFileMediaAbstract, SOMPMediaInterfaceAbstract, SOMObject

New Methods

sompInitSpecific
sompStat
sompQueryBlockSize
sompGetMediaName

Overriding Methods

sompInitReadWrite, sompInitReadOnly
sompOpen, sompClose
sompSeekPosition, sompSeekPositionRel
sompGetOffset
sompReadBytes, sompWriteBytes
sompReadOctet, sompWriteOctet
sompReadShort, sompWriteShort
sompReadUnsignedShort, sompWriteUnsignedShort
sompReadLong, sompWriteLong
sompReadUnsignedLong, sompWriteUnsignedLong
sompReadDouble, sompWriteDouble
sompReadFloat, sompWriteFloat
sompReadCharacter, sompWriteCharacter
sompReadSomobject, sompWriteSomobject
sompReadString, sompWriteString
sompReadStringToBuffer
sompReadLine, sompWriteLine
sompReadTypeCode, sompWriteTypeCode
somInit, somUninit

sompGetMediaName Method

Purpose

Gets the name of the file for the current media interface object.

IDL Syntax

string sompGetMediaName (
in string *toBuffer***);**

Description

The **sompGetMediaName** method puts the file name for the current media interface object into *toBuffer* and returns *toBuffer*. The buffer must be allocated (and freed) by the client. This method is typically used by implementors of I/O Group Managers.

Parameters

receiver	A pointer to an object of class **SOMPAsciiMediaInterface.**
ev	A pointer to the **Environment** structure for the calling method.
toBuffer	The name of the file for the associated media interface object.

Return Value

The **sompGetMediaName** method returns the address of the buffer where the file name was placed.

Example

```
Environment *ev;
SOMAsciiMediaInterface mymi;
char buffer[SOMPMAXIDSIZE];
/* ... */
somPrintf(”File name is: %s\n”,
   _sompGetMediaName(mymi, ev, (string) buffer));
```

Original Class

SOMPAsciiMediaInterface

Related Information

Methods: sompInitSpecific, sompQueryBlockSize, sompStat

sompInitSpecific Method

Purpose

Initializes the permissions and creation flags of ASCII file media.

IDL Syntax

void sompInitSpecific (
inout mediaInfoType *mediaInfo***);**

Description

The **sompInitSpecific** method initializes file media with specific characteristics. Use this method if neither **sompInitReadWrite** nor **sompInitReadOnly** is appropriate. This method is typically used by client programs.

Parameters

receiver A pointer to an object of class **SOMPAsciiMediaInterface**.

ev A pointer to the **Environment** structure for the caller.

mediaInfo A pointer to information that a particular (concrete) ASCII medium requires before opening a file. For a media interface to an AIX, OS/2, or Windows file system, the *mediaInfo* parameter should be a pointer to a **mediaInfoType** structure such as the following (which also needs a related "mediaFlagsType" structure as shown):

```
typedef struct mediaInfo_Type  {
  string Name;                      /* file name                  */
  mediaFlagsType Flags;             /* flags (see mediaFlagsType) */
  unsigned short unix_file_permissions; /* permissions (unused    */
} mediaInfoType;                        /*              on OS/2)  */

typedef struct mediaFlags_Type {
  octet  open_for_read;             /* 1=read only                */
  octet  open_for_readwrite;        /* 1=read/write               */
  octet  create_if_nonexistent;     /* 1=create if file nonexist. */
  octet  share;                     /* 1=allow multiple readers   */
} mediaFlagsType;
```

The following items provide additional information for setting the "Flags" bits and the "unix_file_permissions":

- Set either **open_for_read** or **open_for_readwrite**. Setting both implies read/write.
- Set **create_if_nonexistent** to create a file if it does not already exist. This flag is ignored if **open_for_read** is set.
- Set **share** to allow others to read the file while you have it open.
- Set **unix_file_permissions** as for the **chmod** command. This flag is ignored if the file is not created. The number is an octal number, for example, 0664. (This argument is not used for OS/2 or Windows.)

Return Value

None.

Example

```
mediaInfoType rwInfo;
char *mediaInfo = "/u/roger/dog.dat";

Environment *ev;
rwInfo.Flags.open_for_readwrite    = TRUE;
rwInfo.Flags.open_for_read         = FALSE;
rwInfo.Flags.create_if_nonexistent = TRUE;
rwInfo.Flags.share                 = TRUE;
rwInfo.unix_file_permissions       = 0664;

rwInfo.Name = (string) SOMMalloc(strlen(mediaInfo)+1);
strcpy(rwInfo.Name, mediaInfo);
_sompInitSpecific(somSelf, ev, &rwInfo);
SOMFree(rwInfo.Name);
```

Original Class

SOMPAsciiMediaInterface

Related Information

Methods: sompGetMediaName, sompQueryBlockSize, sompStat

sompQueryBlockSize Method

Purpose

Returns an optimal block size for I/O operations.

IDL Syntax

unsigned long sompQueryBlockSize ( );

Description

The **sompQueryBlockSize** method returns an optimal block size for I/O operations. This method is called prior to allocating a buffer to be used for reading or writing. This method is typically used by implementors of I/O Group Managers.

Parameters

receiver	A pointer to an object of class **SOMPAsciiMediaInterface.**
ev	A pointer to the **Environment** structure for the caller.

Return Value

The **sompQueryBlockSize** method returns the optimal block size needed for the file system.

Example

```
long blockLength;
SOMPAsciiMediaInterface srcFmi;
Environment *ev;
/* ... */
blockLength = _sompQueryBlockSize(srcFmi, ev);
```

Original Class

SOMPAsciiMediaInterface

Related Information

Methods: sompGetMediaName, sompInitSpecific, sompStat

sompStat Method

Purpose

Sets **sompstat** information for the file associated with the current object.

IDL Syntax

void sompStat (
inout sompstat *fileStats***);**

Description

The **sompStat** method fills the **sompstat** structure for the file associated with the current object. The structure is defined in the sompstad.idl file. This method is typically used by implementors of I/O Group Managers.

Parameters

receiver	A pointer to an object of class **SOMPAsciiMediaInterface.**
ev	A pointer to the **Environment** structure for the caller.
fileStats	A pointer to a **sompstat** structure.

Return Value

None. However, the **sompStat** method may raise a **sompException** exception.

Example

```
#include <stdio.h>
#include <sys/stat.h>
Environment *myenv;
struct stat tmpFileStats;
SOMPMediaInterfaceAbstract tmpFmi;
/* ... */
_sompStat(tmpFmi, myenv, (sompstat *)&(tmpFileStats));
```

Original Class

SOMPAsciiMediaInterface

Related Information

Methods: sompGetMediaName, sompInitSpecific, sompQueryBlockSize

SOMPAttrEncoderDecoder Class

Description

An Encoder/Decoder is an object that is paired with a persistent object and handles the reading/writing of that persistent object's data. The encoder/decoder is instantiated by an I/O Group Manager when the persistent object is saved or restored. The **SOMPAttrEncoderDecoder** class is an Encoder/Decoder that can save/restore the IDL attributes of a given persistent object. This is the default encoder/decoder for a persistent object unless specified otherwise.

For example, suppose an IDL interface definition includes the following attributes:

```
attribute string   name;
attribute string   food;
attribute long     weight;
attribute short    height;
```

The attributes"name" and "weight" are made persistent by specifying the following SOM modifiers in the "implementation" section of the interface definition:

```
name: persistent;
weight: persistent;
```

File Stem

defedidl

Base

SOMPEncoderDecoderAbstract

Metaclass

SOMClass

Ancestor Classes

SOMPEncoderDecoderAbstract, SOMObject

New Methods

None.

Overriding Methods

somInit
sompEDRead
sompEDWrite

SOMPBinary Class

Description

The **SOMPBinary** class is an I/O Group Manager that writes groups of objects in a format similar to the **SOMPAscii** class, except that this class stores numeric data in binary.

File Stem

fsgm

Base

SOMPAscii

Metaclass

SOMClass

Ancestor Classes

SOMPAscii, SOMPIOGroupMgrAbstract, SOMObject

New Methods

None.

Overriding Methods

sompInstantiateMediaInterface

SOMPBinaryFileMedia Class

Description

The **SOMPBinaryFileMedia** class provides an interface to the file system that writes numbers in binary.

File Stem

fmib

Base

SOMPAsciiMediaInterface

Metaclass

SOMClass

Ancestor Classes

SOMPAsciiMediaInterface, SOMPFileMediaAbstract, SOMPMediaInterfaceAbstract, SOMObject

New Methods

None

Overriding Methods

sompReadOctet, sompWriteOctet
sompReadShort, sompWriteShort
sompReadUnsignedShort, sompWriteUnsignedShort
sompReadLong, sompWriteLong
sompReadUnsignedLong, sompWriteUnsignedLong
sompReadDouble, sompWriteDouble
sompReadFloat, sompWriteFloat
sompReadString, sompWriteString
sompReadStringToBuffer

SOMPEncoderDecoderAbstract Class

Description

The **SOMPEncoderDecoderAbstract** class is the abstract definition of an Encoder/Decoder.

An Encoder/Decoder is an object that handles the reading and writing of the instance data of a persistent object. A class derived from the **SOMPEncoderDecoderAbstract** class must be created and paired with a persistent object to save or restore that persistent object.

The derived Encoder/Decoder name is bound to a persistent object using the method **sompSetEncoderDecoderName** or **sompSetClassLevelEncoderDecoderName.**

A persistent object's Encoder/Decoder is instantiated by an I/O Group Manager at the moment that object is saved or restored.

File Stem

eda

Base

SOMObject

Metaclass

SOMClass

Ancestor Classes

SOMObject

New Methods

sompEDRead
sompEDWrite

sompEDRead Method

Purpose

Reads the stored instance data of a persistent object, and updates the persistent object to reflect the data.

IDL Syntax

void sompEDRead (
in SOMPMediaInterfaceAbstract *thisMedia*,
in SOMPPersistentObject *thisObject*);

Description

The **sompEDRead** method reads the stored persistent data of the given persistent object *thisObject*, updating the persistent object to reflect the data. This method is the mirror image of the **sompEDWrite** method, and it expects the data to be in the format of **sompEDWrite.**

The **sompEDRead** method uses the media interface that is passed in as the argument *thisMedia.* The method assumes that the media interface has been initialized and opened. Also, it assumes that the media interface will subsequently be closed by the caller.

The **sompEDRead** method may raise a **sompException** exception.

This method is typically overridden by implementors of Encoder/Decoders.

Parameters

receiver	A pointer to an object of class **SOMPEncoderDecoderAbstract.**
ev	A pointer to the **Environment** structure for the caller.
thisMedia	A pointer to an object of class **SOMPMediaInterfaceAbstract**.
thisObject	A pointer to an object of class **SOMPPersistentObject**.

Return Value

None.

Example

```
SOMPEncoderDecoderAbstract ed;
Environment *ev;
ed = _sompGetEncoderDecoder(thisPo, ev);

/* ... */
if (ed) {
  _sompEDRead(ed, ev,
  _sompGetMediaInterface(somSelf, ev), thisPo);
}
```

Original Class

SOMPEncoderDecoderAbstract

Related Information

Methods: sompEDWrite

sompEDWrite Method

Purpose

Writes the instance data of a persistent object to storage media in a format that is compatible with the **sompEDRead** method.

IDL Syntax

void sompEDWrite (
in SOMPMediaInterfaceAbstract *thisMedia***,**
in SOMPPersistentObject *thisObject***);**

Description

The **sompEDWrite** method writes all the instance data of a specified persistent object, *thisObject*, to storage media, using a format that is compatible with the **sompEDRead** method.

The **sompEDWrite** method uses the media interface that is passed in as the argument *thisMedia.* The method assumes that the media interface has been initialized and opened. It also assumes that the media interface will be closed by the caller.

This method may raise a **sompException** exception.

This method is typically overridden by implementors of Encoder/Decoders.

Parameters

receiver	A pointer to an object of class **SOMPEncoderDecoderAbstract**.
ev	A pointer to the **Environment** structure for the caller.
thisMedia	A pointer to an object of class **SOMPMediaInterfaceAbstract**.
thisObject	A pointer to an object of class **SOMPPersistentObject**.

Return Value

None.

Example

```
SOMPEncoderDecoderAbstract ed;
Environment *ev;
/* ... */
ed = _sompGetEncoderDecoder(thisPo, ev);
if (ed) {
   _sompEDWrite(ed, ev,
   _sompGetMediaInterface(somSelf, ev), thisPo);
   /* ... */
```

Original Class

SOMPEncoderDecoderAbstract

Related Information

Methods: sompEDRead

SOMPFileMediaAbstract Class

Description

The **SOMPFileMediaAbstract** class represents the abstract class definition for an interface to file media.

File Stem

fma

Base

SOMPMediaInterfaceAbstract

Metaclass

SOMClass

Ancestor Classes

SOMPMediaInterfaceAbstract, SOMObject

New Methods

Group: General

sompInitReadOnly
sompInitReadWrite

Group: Positioning

sompSeekPosition
sompSeekPositionRel
sompGetOffset

Group: I/O

sompReadBytes, sompWriteBytes
sompReadShort, sompWriteShort
sompReadUnsignedShort, sompWriteUnsignedShort
sompReadLong, sompWriteLong
sompReadUnsignedLong, sompWriteUnsignedLong
sompReadFloat, sompWriteFloat
sompReadDouble, sompWriteDouble
sompReadOctet, sompWriteOctet
sompReadCharacter, sompWriteCharacter
sompReadSomobject, sompWriteSomobject
sompReadLine, sompWriteLine
sompReadString, sompWriteString
sompReadStringToBuffer

Overriding Methods

sompOpen
sompClose

sompGetOffset Method

Purpose

Returns the current offset within the Media Interface file.

IDL Syntax

long sompGetOffset ( );

Description

The **sompGetOffset** method gets the current offset within a Media Interface file. This method is typically used by implementors of I/O Group Managers.

Parameters

receiver	A pointer to an object of class **SOMPFileMediaAbstract.**
ev	A pointer to the **Environment** structure for the caller.

Return Value

The **sompGetOffset** method returns a long integer representing the current offset.

Example

```
Environment *ev;
SOMPFileMediaAbstract fmi;
/* ... */
nextObjectPos = _sompGetOffset(fmi, ev);
```

Original Class

SOMPFileMediaAbstract

Related Information

Methods: sompSeekPosition, **sompSeekPositionRel**

sompInitReadOnly Method

Purpose

Initializes the Media Interface file for read-only access.

IDL Syntax

void sompInitReadOnly (
in string *mediaInfo***);**

Description

The **sompInitReadOnly** method initializes the Media Interface file for read-only access.

The *mediaInfo* input parameter is the file name. The **sompInitReadOnly** method prepares a read-only file for opening. The file must exist or **sompOpen** will fail. This method allows multiple readers of the file.

The **sompInitReadOnly** method may raise a **sompException** exception.

This method is typically used by implementors of I/O Group Managers.

Parameters

receiver	A pointer to an object of class **SOMPFileMediaAbstract.**
ev	A pointer to the **Environment** structure for the caller.
mediaInfo	The file name.

Return Value

None.

Example

```
string IOInfo = ”dog.dat”;
Environment *ev;
SOMPAsciiMediaInterface mia;
/* ... */
if (mia == NULL) {
   /* ... */
   _sompInitReadOnly(mia, ev, IOInfo);
}
```

Original Class

SOMPFileMediaAbstract

Related Information

Methods: sompInitReadWrite, **sompInitSpecific**, **sompOpen**, **sompClose**

sompInitReadWrite Method

Purpose

Initializes the Media Interface file for read/write access.

IDL Syntax

void sompInitReadWrite (
in string *mediaInfo***);**

Description

The **sompInitReadWrite** method initialize the file media for read/write access.

The *mediaInfo* input parameter is the file name. The **sompInitReadWrite** method prepares a read/write file for opening. When **sompOpen** is subsequently called, the file is created if it does not already exist. Multiple readers and one writer of the file are allowed. Under AIX, permissions are set to allow read/write access to both owner and group, and read access to others.

The **sompInitReadWrite** method may raise a **sompException** exception.

This method is typically used by implementors of I/O Group Managers.

Parameters

receiver	A pointer to an object of class **SOMPFileMediaAbstract**.
ev	A pointer to the **Environment** structure for the caller.
mediaInfo	The name of the file to be prepared for opening.

Return Value

None.

Example

```
string IOInfo = ”dog.dat”;
Environment *ev;
SOMPAsciiMediaInterface mia;
/* ... */
if (mia == NULL) {
   /* ... */
   _sompInitReadWrite(mia, ev, IOInfo);
}
```

Original Class

SOMPFileMediaAbstract

Related Information

Methods: sompInitReadOnly, **sompOpen**, **sompClose**

sompReadBytes Method

Purpose

Reads a byte stream of the specified length starting from the current position in a Media Interface file into a preallocated buffer.

IDL Syntax

void sompReadBytes (
in string *bytestream,*
in long *length***);**

Description

The **sompReadBytes** method reads the specified *length* of bytes from the file and places them in memory beginning at *bytestream*. This method may raise a **sompException** exception. This method is typically used by implementors of Encoders/Decoders and I/O Group Managers.

Parameters

receiver	A pointer to a **SOMPFileMediaAbstract** object.
ev	A pointer to the **Environment** structure for the caller.
bytestream	A pointer to a buffer where the byte stream will be placed.
length	A long integer representing the number of bytes to read.

Return Value

None.

Example

```
string c;
Environment *ev;
SOMPFileMediaAbstract media;
/* ... */
_sompReadBytes(media, ev, c, 1);
```

Original Class

SOMPFileMediaAbstract

Related Information

Methods: sompWriteBytes

sompReadCharacter Method

Purpose

Reads character data from a Media Interface file.

IDL Syntax

void sompReadCharacter (
inout char ***c*);**

Description

The **sompReadCharacter** method reads data of type **char**. This method may raise a **sompException** exception. This method is typically used by implementors of Encoders/Decoders and I/O Group Managers.

Parameters

receiver	A pointer to an object of class **SOMPFileMediaAbstract.**
ev	A pointer to the **Environment** structure for the caller.
c	A pointer to a memory location where the character data will be placed.

Return Value

None.

Example

```
char chr;
Environment *ev;
SOMPFileMediaAbstract media;
/* ... */
_sompReadCharacter(media, ev, &chr);
```

Original Class

SOMPFileMediaAbstract

Related Information

Methods: sompWriteCharacter

sompReadDouble Method

Purpose

Reads double-precision floating-point data from a Media Interface file.

IDL Syntax

void sompReadDouble (
inout double *f*);

Description

The **sompReadDouble** method reads data of type **double.** This method may raise a **sompException** exception. This method is typically used by implementors of Encoders/Decoders and I/O Group Managers.

Parameters

receiver	A pointer to an object of class **SOMPFileMediaAbstract**.
ev	A pointer to the **Environment** structure for the caller.
f	A pointer to storage where the **double** value will be placed.

Return Value

None.

Example

```
double dbl;
Environment *ev;
SOMPFileMediaAbstract media;
/* ... */
_sompReadDouble(media, ev, &dbl);
```

Original Class

SOMPFileMediaAbstract

Related Information

Methods: sompWriteDouble

sompReadFloat Method

Purpose

Reads floating-point data from a Media Interface file.

IDL Syntax

void sompReadFloat (
inout float *f*);

Description

The **sompReadFloat** method reads data of type **float.** This method may raise a **sompException** exception. This method is typically used by implementors of Encoders/Decoders and I/O Group Managers.

Parameters

receiver	A pointer to an object of class **SOMPFileMediaAbstract**.
ev	A pointer to the **Environment** structure for the caller.
f	A pointer to storage where the **float** value will be placed.

Return Value

None.

Example

```
SOMPFileMediaAbstract media;
Environment *ev;
float f;
/* ... */
_sompReadFloat(media, ev, &f);
```

Original Class

SOMPFileMediaAbstract

Related Information

Methods: sompWriteFloat

sompRead<*IntegralType*> Methods

Purpose

Reads integers from a Media Interface file.

IDL Syntax

void sompReadShort (inout short *i1***);**

void sompReadUnsignedShort (inout unsigned short *i1***);**

void sompReadLong (inout long *i2***);**

void sompReadUnsignedLong (inout unsigned long *i2***);**

Description

The **sompRead**<*IntegralType*> methods read an integer of the type indicated above. These methods may raise a **sompException** exception. This method is typically used by implementors of Encoders/Decoders and I/O Group Managers.

Parameters

receiver	A pointer to an object of class **SOMPFileMediaAbstract.**
ev	A pointer to the **Environment** structure for the caller.
i	A pointer to the memory location where the integer will be stored.

Return Value

None.

Example

```
SOMPFileMediaAbstract media;
Environment *ev;
short i1;
/* ... */
_sompReadShort(media, ev, &i1);
```

Original Class

SOMPFileMediaAbstract

Related Information

Methods: sompWrite<IntegralType>

sompReadLine Method

Purpose

Reads a line from the Media Interface file into a preallocated buffer.

IDL Syntax

void sompReadLine (
in string *buffer,*
in long *bufsize***);**

Description

The **sompReadLine** method reads a string from the Media Interface file up to and including the first newline character. The string is stored in *buffer*. This method is suitable for reading strings written using **sompWriteLine.** The *bufsize* parameter specifies the size of *buffer.* If the string read is longer than *bufsize*, then it is truncated to fit. A null character is stored at the end of the string. The newline character, if read, is included in the string.

This method may raise a **sompException** exception.

This method is typically used by implementors of Encoders/Decoders and I/O Group Managers.

Parameters

receiver	A pointer to an object of class **SOMPFileMediaAbstract.**
ev	A pointer to the **Environment** structure for the caller.
buffer	A pointer to the memory location where the line will be placed.
bufsize	The size of *buffer.*

Return Value

None.

Example

```
char ioBuff[SOMPMAXIDSIZE];
Environment *myEnv;
SOMPAsciiMediaInterface *myFmi;
/* ... */
_sompReadLine (myFmi, myEnv, ioBuff, SOMPMAXIDSIZE);
```

Original Class

SOMPFileMediaAbstract

Related Information

Methods: sompWriteLine

sompReadOctet Method

Purpose

Reads an eight-bit quantity from the Media Interface file.

IDL Syntax

void sompReadOctet (
inout octet *f*);

Description

The **sompReadOctet** method reads data of type **octet** (an eight-bit quantity)**.** This method may raise a **sompException** exception. This method is typically used by implementors of Encoders/Decoders and I/O Group Managers.

Parameters

receiver	A pointer to an object of class **SOMPFileMediaAbstract**.
ev	A pointer to the **Environment** structure for the caller.
f	A pointer to storage where the **octet** value will be placed.

Return Value

None.

Example

```
octet f;
Environment *ev;
SOMPFileMediaAbstract media;
/* ... */
_sompReadOctet(media, ev, &f);
```

Original Class

SOMPFileMediaAbstract

Related Information

Methods: sompWriteOctet

sompReadSomobject Method

Purpose

Reads a SOM Object from the Media Interface file or, if the object is non-persistent, instantiates the proper class.

IDL Syntax

void sompReadSomobject (
inout SOMObject *so***);**

Description

The **sompReadSomobject** method reads a **SOMObject** from the Media Interface file. If the object is persistent, the object will be read; if the object is non-persistent, the proper class is instantiated, but no data is read in.

This method may raise a **sompException** exception. This method is typically used by implementors of Encoders/Decoders and I/O Group Managers.

Parameters

receiver	A pointer to an object of class **SOMPFileMediaAbstract.**
ev	A pointer to the **Environment** structure for the caller.
so	A pointer to the memory area where the **SOMObject** was placed.

Return Value

None.

Example

```
SOMObject ro;
Environment *ev;
SOMPFileMediaAbstract media;
/* ... */
_sompReadSomobject(media, ev, &ro);
```

Original Class

SOMPFileMediaAbstract

Related Information

Methods: sompWriteSomobject

sompReadString Method

Purpose

Allocates space for a string and reads the string from the Media Interface file.

IDL Syntax

void sompReadString (
inout string *rstring***);**

Description

The **sompReadString** method reads a string from the Media Interface file. This method may raise a **sompException** exception. This method is typically used by implementors of Encoders/Decoders and I/O Group Managers.

Parameters

receiver	A pointer to an object of class **SOMPFileMediaAbstract.**
ev	A pointer to the **Environment** structure for the caller.
rstring	A pointer to the memory location where a pointer to the string will be placed.

Return Value

None.

Example

```
string rstring;
Environment *ev;
SOMPFileMediaAbstract media;
/* ... */
_sompReadString(media, ev, &rstring);
```

Original Class

SOMPFileMediaAbstract

Related Information

Methods: sompWriteString

sompReadStringToBuffer Method

Purpose

Reads a string into a preallocated buffer.

IDL Syntax

void sompReadStringToBuffer (
in string *buffer*,
in long *bufsize***);**

Description

The **sompReadStringToBuffer** method reads a string into a preallocated *buffer*. In *buffer,* the string will be null-terminated. The *bufsize* parameter specifies the size of the buffer. If the string to be read is larger than *bufsize*, then it is truncated to fit the specified *bufsize.*

This method may raise a **sompException** exception. This method is typically used by implementors of Encoders/Decoders and I/O Group Managers.

Parameters

receiver	A pointer to an object of class **SOMPFileMediaAbstract** representing the file containing the string.
ev	A pointer to the **Environment** structure for the caller.
buffer	A pointer to an area in memory where the string will be stored.
bufsize	A long integer representing the size of the buffer.

Return Value

None.

Example

```
#define BUFFER_SIZE 100
char buffer[BUFFER_SIZE];
Environment *ev;
SOMPFileMediaAbstract fmi;
/* ... */
_sompReadStringToBuffer(fmi, ev, buffer, BUFFER_SIZE);
```

Original Class

SOMPFileMediaAbstract

Related Information

Methods: sompWriteString

sompReadTypeCode Method

Purpose

Reads an IDL type code from a Media Interface file.

IDL Syntax

void sompReadTypeCode (
inout TypeCode *tc*);

Description

The **sompReadTypeCode** method reads an IDL type code from a Media Interface file.

This method may raise a **sompException** exception. This method is typically used by implementors of I/O Group Managers.

Parameters

receiver A pointer to an object of class **SOMPFileMediaAbstract.**

ev A pointer to the **Environment** structure for the caller.

tc A pointer to a memory area where the **TypeCode** value will be placed.

Return Value

None.

Example

```
SOMPFileMediaAbstract media;
TypeCode thisTypeCode;
Environment *ev;
/* ... */
_sompReadTypeCode(media, ev, &thisTypeCode);
```

Original Class

SOMPFileMediaAbstract

Related Information

Methods: sompWriteTypeCode

sompSeekPosition, sompSeekPositionRel Methods

Purpose

Positions the Media Interface file to a specified offset.

IDL Syntax

void sompSeekPosition (
in long *offset***);**

void sompSeekPositionRel (
in long *offset***);**

Description

The **sompSeekPosition** method positions the storage medium to a specified offset, so that any further reading or writing will begin from that point. Reading and writing implicitly repositions the file, so use of this method is not usually necessary.

The **sompSeekPositionRel** method positions the media to a specified offset relative to the current position in the file. This method is like **sompSeekPosition**, except that the offset is not measured from the beginning of the file.

These methods may raise a **sompException** exception. This method is typically used by implementors of Encoders/Decoders and I/O Group Managers.

Parameters

receiver	A pointer to an object of class **SOMPFileMediaAbstract.**
ev	A pointer to the **Environment** structure for the caller.
offset	A **long** integer representing the offset at which to position the storage medium.

Return Value

None.

Example

```
SOMPFileMediaAbstract fmi;
Environment *ev;
long offset;
/* ... */
_sompSeekPosition(fmi, ev, offset);
```

Original Class

SOMPFileMediaAbstract

Related Information

Methods: sompGetOffset

sompWriteBytes Method

Purpose

Writes a byte stream of a specified length to the Media Interface file.

IDL Syntax

void sompWriteBytes (
in string *bytestream*
in long *length***);**

Description

The **sompWriteBytes** method writes a byte stream of the specified length to the Media Interface file. This method may raise a **sompException** exception. This method is typically used by implementors of Encoders/Decoders and I/O Group Managers.

Parameters

receiver	A pointer to an object of class **SOMPFileMediaAbstract**.
ev	A pointer to the **Environment** structure for the caller.
bytestream	A pointer to the buffer containing the bytes to be written.
length	A long integer representing the number of bytes to write.

Return Value

None.

Example

```
#define BUFFERSIZE 100
SOMPFileMediaAbstract fmi;
Environment *ev;
char buffer[BUFFERSIZE];
/* ... */
_sompWriteBytes(fmi, ev, buffer, BUFFERSIZE);
```

Original Class

SOMPFileMediaAbstract

Related Information

Methods: sompReadBytes

sompWriteCharacter Method

Purpose

Writes a character to the Media Interface file.

IDL Syntax

void sompWriteCharacter (
in char *c*);

Description

The **sompWriteCharacter** method writes data of type **char** to the Media Interface file. This method may raise a **sompException** exception. This method is typically used by implementors of Encoders/Decoders and I/O Group Managers.

Parameters

receiver	A pointer to a **SOMPFileMediaAbstract** object.
ev	A pointer to the **Environment** structure for the caller.
c	A character (**char**) to be written.

Return Value

None.

Example

```
SOMPFileMediaAbstract media;
Environment *ev
char c;
/* ... */
_sompWriteCharacter(media, ev, c);
```

Original Class

SOMPFileMediaAbstract

Related Information

Methods: sompReadCharacter

sompWriteDouble Method

Purpose

Writes double-precision floating-point data to the Media Interface file.

IDL Syntax

void sompWriteDouble (
in double *f*);

Description

The **sompWriteDouble** method writes data of type **double** to the Media Interface File. This method may raise a **sompException** exception. This method is typically used by implementors of Encoders/Decoders and I/O Group Managers.

Parameters

receiver	A pointer to an object of class **SOMPFileMediaAbstract.**
ev	A pointer to the **Environment** structure for the caller.
f	The **double** value to be written.

Return Value

None.

Example

```
SOMPFileMediaAbstract media;
Environment *ev;
double d;
/* ... */
_sompWriteDouble(media, ev, d);
```

Original Class

SOMPFileMediaAbstract

Related Information

Methods: sompReadDouble

sompWriteFloat Method

Purpose

Writes floating-point data to the Media Interface file.

IDL Syntax

void sompWriteFloat (
in float *f*);

Description

The **sompWriteFloat** method writes data of type **float** to the Media Interface File. This method may raise a **sompException** exception. This method is typically used by implementors of Encoders/Decoders.

Parameters

receiver	A pointer to an object of class **SOMPFileMediaAbstract.**
ev	A pointer to the **Environment** structure for the caller.
f	The **float** value to be written.

Return Value

None.

Example

```
SOMPFileMediaAbstract media;
Environment *ev;
float f;
/* ... */
_sompWriteFloat(media, ev, f);
```

Original Class

SOMPFileMediaAbstract

Related Information

Methods: sompReadFloat

sompWrite<*IntegralType*> Methods

Purpose

Writes an integer to the Media Interface file.

IDL Syntax

void sompWriteShort (in short *i1***);**

void sompWriteUnsignedShort (in unsigned short *i1***);**

void sompWriteLong (in long *i2***);**

void sompWriteUnsignedLong (in unsigned long *i2***);**

Description

The **sompWrite**<*Integraltype*> methods write an integer of the type shown above. These methods may raise a **sompException** exception. This method is typically used by implementors of Encoders/Decoders.

Parameters

receiver	A pointer to an object of class **SOMPFileMediaAbstract**.
ev	A pointer to the **Environment** structure for the caller.
i	The integral value to be written.

Return Value

None.

Example

```
SOMPFileMediaAbstract media;
Environment *ev;
short s;
/* ... */
_sompWriteShort(media, ev, s);
```

Original Class

SOMPFileMediaAbstract

Related Information

Methods: sompRead<*IntegralType*>

sompWriteLine Method

Purpose

Writes a line to the Media Interface file.

IDL Syntax

void sompWriteLine (
in string *buffer***);**

Description

The **sompWriteLine** method writes the line stored in *buffer* to a file. The terminating null character is not written.

This method does not append a newline character to the given string before writing. If the user of this method intends to restore the line using the **sompReadLine** method, then the user must put the newline character in *buffer* before invoking this method.

This method may raise a **sompException** exception. This method is typically used by implementors of Encoders/Decoders.

Parameters

receiver	A pointer to an object of class **SOMPFileMediaAbstract.**
ev	A pointer to the **Environment** structure for the caller.
buffer	The line to be written.

Return Value

None.

Example

```
char ioBuff[SOMPMAXIDSIZE];
Environment *myEnv;
SOMPAsciiMediaInterface *myFmi;

_sompWriteLine (myFmi, myEnv, ioBuff);
```

Original Class

SOMPFileMediaAbstract

Related Information

Methods: sompReadLine

sompWriteOctet Method

Purpose

Writes an eight-bit quantity to the Media Interface file.

IDL Syntax

void sompWriteOctet (
in octet *f*);

Description

The **sompWriteOctet** method writes data of type **octet** (an eight-bit quantity) to the Media Interface File. This method may raise a **sompException** exception. This method is typically used by implementors of Encoders/Decoders.

Parameters

receiver	A pointer to an object of class **SOMPFileMediaAbstract.**
ev	A pointer to the **Environment** structure for the caller.
f	The **octet** value to be written.

Return Value

None.

Example

```
SOMPfileMediaAbstract media;
Environment *ev;
octet o;
/* ... */
_sompWriteOctet(media, ev, o);
```

Original Class

SOMPFileMediaAbstract

Related Information

Methods: sompReadOctet

sompWriteSomobject Method

Purpose

Writes a SOM Object to the Media Interface file or, if the object is non-persistent, stores only its class name.

IDL Syntax

void sompWriteSomobject (
in SOMObject *so***,**
in SOMObject *parentObject***);**

Description

If **SOMObject** is persistent, **sompWriteSomobject** writes out the ID of the object and adds the object to the list of objects that will eventually be written as a result of this call. If *parentObject* is not NULL, then this object's ID will be written relative to the *parentObject*. Relative IDs have the advantage of not requiring updating when the file containing the object is moved. If **SOMObject** is not persistent, then **sompWriteSomobject** writes out only the class name of the object.

This method may raise a **sompException** exception. This method is typically used by implementors of Encoders/Decoders.

Parameters

receiver	A pointer to an object of class **SOMPFileMediaAbstract.**
ev	A pointer to the **Environment** structure for the caller.
so	A pointer to the **SOMObject** to be written.
parentObject	A pointer to the container object of this **SOMObject**.

Return Value

None.

Example

```
SOMPFileMediaAbstract media;
Enviornment *ev;
SOMObject somo;
/* ... */
_sompWriteSomobject(media, ev, somo, NULL);
```

Original Class

SOMPFileMediaAbstract

Related Information

Methods: sompReadSomobject

sompWriteString Method

Purpose

Writes a string to the Media Interface file.

IDL Syntax

void sompWriteString (
in string *string***);**

Description

The **sompWriteString** method writes a string to the Media Interface file. This method may raise a **sompException** exception. The string is written out in two parts. The first part is the number of bytes. The second part is the characters in the string. The null terminator is not written. This method is typically used by implementors of Encoders/Decoders.

Parameters

receiver	A pointer to an object of class **SOMPFileMediaAbstract.**
ev	A pointer to the **Environment** structure for the caller.
string	The string to be written.

Return Value

None.

Example

```
SOMPFileMediaStract media;
Environment *ev;
string buffer = ”This is a test”;
/* ... */
_sompWriteString(media, ev, buffer);
```

Original Class

SOMPFileMediaAbstract

Related Information

Methods: sompReadString

sompWriteTypeCode Method

Purpose

Writes a restorable version of an IDL type code to the Media Interface file.

IDL Syntax

void sompWriteTypeCode (
in TypeCode *tc***);**

Description

The **sompWriteTypeCode** method writes a restorable version of an IDL type code to the Media Interface file.

This method may raise a **sompException** exception. This method is typically used by implementors of Encoders/Decoders.

Parameters

receiver	A pointer to an object of class **SOMPFileMediaAbstract**.
ev	A pointer to the **Environment** structure for the caller.
tc	The type code to be written.

Return Value

None.

Example

```
SOMPFileMediaStract media;
Environment *ev;
TypeCode tc;
/* ... */
_sompWriteTypeCode(media, ev, tc);
```

Original Class

SOMPFileMediaAbstract

Related Information

Methods: sompReadTypeCode

SOMPIdAssigner Class

Description

The **SOMPIdAssigner** class is the concrete definition (the base class) of an ID Assigner. When a persistent object is initialized with the **sompInitNextAvail** method, an ID Assigner is used to give the persistent object an ID.

This class implements a method, **sompGetSystemAssignedId,** for obtaining a persistent object ID, which is composed of a path name and file name. The path name portion of the ID is determined based on the string assigned to the environment variable SOMP_PERSIST. If SOMP_PERSIST is undefined or NULL, a dot (".") is used to indicate the current directory.

The file name portion of the ID is the string **"p"** concatenated with a seven digit hex number. This number is read from a file named somplast.id, which is stored at the path determined by the value of the environment variable SOMP_PERSIST. If file somplast.id does not exist, the number 0000000 is used; then, the somplast.id file is created and the number 1 is written into it.

File Stem

poid

Base

SOMPIdAssignerAbstract

Metaclass

SOMClass

Ancestor Classes

SOMPIdAssignerAbstract, **SOMObject**

New Methods

None.

Overriding Methods

sompGetSystemAssignedId

SOMPIdAssignerAbstract Class

Description

The **SOMPIdAssignerAbstract** class provides the abstract definition of an ID Assigner. An ID Assigner is used when a persistent object is initialized with **sompInitNextAvail**, to give the persistent object an ID. **SOMPIdAssignerAbstract** defines a method, **sompGetSystemAssignedId,** for obtaining a persistent object ID.

File Stem

poida

Base

SOMObject

Metaclass

SOMClass

Ancestor Classes

SOMObject

New Methods

sompGetSystemAssignedId

sompGetSystemAssignedId Method

Purpose

Assigns a persistent ID to a persistent object.

IDL Syntax

void sompGetSystemAssignedId (
inout SOMPPersistentId *id***);**

Description

The **sompGetSystemAssignedId** method assigns a persistent ID to a persistent object. This method also creates the "**somplast.id**" file if it has not previously been created (see more information at the **SOMPIdAssigner** class). The file is created in the current directory, if the environmental variable **SOMP_PERSIST** has not been set, or in the directory contained in **SOMP_PERSIST**. It is the responsibility of the persistent object to free the memory occupied by the ID.

The **sompGetSystemAssignedId** method may raise a **sompException** exception. This method is typically used by client programs and may be overridden by implementors of ID Assigners.

Parameters

receiver	A pointer to an object of class **SOMPIdAssignerAbstract.**
ev	A pointer to the **Environment** structure for the caller.
id	A pointer to storage where the system ID will be placed.

Return Value

None.

Example

```
SOMPIdAssignerAbstract thisAssigner;
Environment *ev;
SOMPPersistentId id;
/* ... */
_sompGetSystemAssignedId(thisAssigner, ev,  &id);
```

Original Class

SOMPIdAssignerAbstract

SOMPIOGroup Class

Description

An I/O Group is a collection of objects that are grouped together so they can be written or read to/from a storage medium at the same time. For example, the group of objects could be stored in the same file. Objects are added to, located, or removed from the I/O Group using a unique (long) integer key as the ID for an object.

File Stem

iogrp

Base

SOMPKeyedSet

Metaclass

SOMClass

Ancestor Classes

SOMPKeyedSet, SOMObject

New Methods

sompCount
sompFirst
sompNewIterator
sompNextObjectInGroup
sompFreeIterator
sompAddToGroup
sompRemoveFromGroup
sompFindByKey

Overriding Methods

somInit
somUninit
somDumpSelfInt

sompAddToGroup Method

Purpose

Adds a specified object to a given I/O Group of objects that are written/read as a unit.

IDL Syntax

void sompAddToGroup (
in SOMObject *newObject***,**
in SOMPIOGroupKey *key***);**

Description

The **sompAddToGroup** method adds the specified *newObject* to the designated I/O Group. The *key* assigned to the *newObject* must be unique among all of the objects in this group. This method is typically used by implementors of I/O Group Managers.

Parameters

receiver	A pointer to an object of class **SOMPIOGroup** representing the I/O Group to which the *newObject* will be added.
ev	A pointer to the **Environment** structure for the calling method.
newObject	The name of the object to be added.
key	A unique **long** that will be used in other methods to identify the *newObject.*

Return Value

None.

Example

```
#include <somp.h>          /* persistence framework */
#include <iogrp.h>         /* SOMPIOGroup class */
#include <direntry.h>      /* an object class to collect */

SOMPTError getErrorCode(Environment *ev);

int main()
{
   int ec = 0;
   Environment *myEnv;
   SOMPTError rc;

   dirEntry name1, name2, anyobj;
   SOMPIOGroup myGroup;
   SOMPIteratorHandle currItem;

   myEnv = somGetGlobalEnvironment();

/* Create the IO Group.
   -------------------- */
   myGroup = SOMPIOGroupNew();

/* Create the directory entries.
   ----------------------------- */
   name1 = dirEntryNew();
   _mkEntry  (name1, "Roger", "555-5085");

   name2 = dirEntryNew();
   _mkEntry  (name2, "Hari", "555-5079");

/* Add entries to the group - in other than creation order.
   -------------------------------------------------------- */
   _sompAddToGroup (myGroup, myEnv, name2, 2);
   _sompAddToGroup (myGroup, myEnv, name1, 1);
```

Original Class

SOMPIOGroup

Related Information

Methods: sompCount, sompFindByKey, sompFirst, sompFreeIterator, sompNewIterator, sompNextObjectInGroup, sompRemoveFromGroup

sompCount Method

Purpose

Determines the number of objects composing an I/O Group that is written/read as a unit.

IDL Syntax

long sompCount ( );

Description

The **sompCount** method determines the number of objects in the designated I/O Group. This method is typically used by implementors of I/O Group Managers.

Parameters

receiver	A pointer to an object of class **SOMPIOGroup** representing the I/O Group to be counted.
ev	A pointer to the **Environment** structure for the calling method.

Return Value

The **sompCount** method returns an integer indicating the number of objects in the I/O Group.

Example

```
#include <somp.h>          /* persistence framework */
#include <iogrp.h>         /* SOMPIOGroup class */
#include <direntry.h>      /* an object class to collect */

SOMPTError getErrorCode(Environment *ev);

int main()
{
   int ec = 0;
   Environment *myEnv;
   SOMPTError rc;

   dirEntry name1, name2, anyobj;
   SOMPIOGroup myGroup;
   SOMPIteratorHandle currItem;

   myEnv = somGetGlobalEnvironment();

/* Create the IO Group.
   -------------------- */
   myGroup = SOMPIOGroupNew();

/* Create the directory entries.
   ----------------------------- */
   name1 = dirEntryNew();
   _mkEntry  (name1, "Roger", "555-5085");

   name2 = dirEntryNew();
   _mkEntry  (name2, "Hari", "555-5079");

/* Add entries to the group - in other than creation order.
   -------------------------------------------------------- */
   _sompAddToGroup (myGroup, myEnv, name2, 2);
   _sompAddToGroup (myGroup, myEnv, name1, 1);

/* Make sure both got in.
   ---------------------- */
   printf ("Group contains %d entries.\n",
     _sompCount(myGroup, myEnv));
```

Original Class

SOMPIOGroup

Related Information

Methods: sompAddToGroup, sompFindByKey, sompFirst, sompFreeIterator, sompNewIterator, sompNextObjectInGroup, sompRemoveFromGroup

sompFindByKey Method

Purpose

Gets the object that corresponds to a specified key.

IDL Syntax

SOMObject sompFindByKey (
in SOMPIOGroupKey *key***);**

Description

The **sompFindByKey** method uses the key that identifies a specific object within a given I/O Group in order to find and return the object. This method is typically used by implementors of I/O Group Managers.

Parameters

receiver	A pointer to an object of class **SOMPIOGroup** representing the I/O Group
ev	A pointer to the **Environment** structure for the calling method.
key	A unique **long** that identifies the object to be returned.

Return Value

The **sompFindByKey** method returns the object corresponding to the specified key.

Example

```
#include <somp.h>          /* persistence framework */
#include <iogrp.h>         /* SOMPIOGroup class */
#include <direntry.h>      /* an object class to collect */

SOMPTError getErrorCode(Environment *ev);

int main()
{
   int ec = 0;
   Environment *myEnv;
   SOMPTError rc;

   dirEntry name1, name2, anyobj;
   SOMPIOGroup myGroup;
   SOMPIteratorHandle currItem;

   myEnv = somGetGlobalEnvironment();

/* Create the IO Group.
   -------------------- */
   myGroup = SOMPIOGroupNew();

/* Create the directory entries.
   ----------------------------- */
   name1 = dirEntryNew();
   _mkEntry  (name1, "Roger", "555-5085");

   name2 = dirEntryNew();
   _mkEntry  (name2, "Hari", "555-5079");

/* Add entries to the group - in other than creation order.
   -------------------------------------------------------- */
   _sompAddToGroup (myGroup, myEnv, name2, 2);
   _sompAddToGroup (myGroup, myEnv, name1, 1);

/* Find an entry by key.
   --------------------- */
   printf ("Entry 2:\n");

   anyobj = _sompFindByKey (myGroup, myEnv, 2);
   _lsEntry(anyobj);
```

Original Class

SOMPIOGroup

Related Information

Methods: sompAddToGroup, sompCount, sompFirst, sompFreeIterator, sompNewIterator, sompNextObjectInGroup, sompRemoveFromGroup

sompFirst Method

Purpose

Gets the first available object in an I/O Group.

IDL Syntax

SOMObject sompFirst ( );

Description

The **sompFirst** method returns the first available object in the specified I/O Group, beginning at the current location in the group. The object returned is not based on any ordering within the group. This method is typically used by implementors of I/O Group Managers.

Parameters

receiver	A pointer to an object of class **SOMPIOGroup** representing the I/O Group.
ev	A pointer to the **Environment** structure for the calling method.

Return Value

The **sompFirst** method returns the first available object.

Example

```
#include <somp.h>          /* persistence framework */
#include <iogrp.h>         /* SOMPIOGroup class */
#include <direntry.h>      /* an object class to collect */

SOMPTError getErrorCode(Environment *ev);

int main()
{
   int ec = 0;
   Environment *myEnv;
   SOMPTError rc;

   dirEntry name1, name2, anyobj;
   SOMPIOGroup myGroup;
   SOMPIteratorHandle currItem;

   myEnv = somGetGlobalEnvironment();

/* Create the IO Group.
   -------------------- */
   myGroup = SOMPIOGroupNew();

/* Create the directory entries.
   ----------------------------- */
   name1 = dirEntryNew();
   _mkEntry  (name1, "Roger", "555-5085");

   name2 = dirEntryNew();
   _mkEntry  (name2, "Hari", "555-5079");

/* Add entries to the group - in other than creation order.
   -------------------------------------------------------- */
   _sompAddToGroup (myGroup, myEnv, name2, 2);
   _sompAddToGroup (myGroup, myEnv, name1, 1);

/* Find the first (physical) entry in the group.
   --------------------------------------------- */
   printf ("First member of Group:\n");
   anyobj = _sompFirst (myGroup, myEnv);
   _lsEntry(anyobj);
   printf("\n");
```

Original Class

SOMPIOGroup

Related Information

Methods: sompAddToGroup, sompCount, sompFindByKey, sompFreeIterator, sompNewIterator, sompNextObjectInGroup, sompRemoveFromGroup

sompFreeIterator Method

Purpose

Frees the resources used by a given iterator for a designated I/O Group.

IDL Syntax

void sompFreeIterator (
in SOMPIteratorHandle *iteratorHandle***);**

Description

This method is typically used by implementors of I/O Group Managers.

Parameters

receiver A pointer to an object of class **SOMPIOGroup** representing the I/O Group corresponding to the iterator.

ev A pointer to the **Environment** structure for the calling method.

iteratorHandle The same handle previously returned by the **sompNewIterator** method and subsequently used by the **sompNextObjectInGroup** method.

Return Value

None.

Example

```
SOMPIOGroup thisGroup;
Environment *ev;
SOMPIteratorHandle hit;
/* ... */
_sompFreeIterator(thisGroup, ev, hit);
```

Original Class

SOMPIOGroup

Related Information

Methods: sompAddToGroup, sompCount, sompFindByKey, sompFirst, sompNewIterator, sompNextObjectInGroup, sompRemoveFromGroup

sompNewIterator Method

Purpose

Gets a handle for use in iterating through an I/O Group of objects that are written/read as a unit.

IDL Syntax

SOMPIteratorHandle sompNewIterator ( );

Description

The **sompNewIterator** method returns a handle that can be used with the **sompNextObjectInGroup** method of this class, in order to iterate through all the objects contained in the designated I/O Group. This method is typically used by implementors of I/O Group Managers.

Parameters

receiver	A pointer to an object of class **SOMPIOGroup** representing the I/O Group.
ev	A pointer to the **Environment** structure for the calling method.

Return Value

The **sompNewIterator** method returns a handle used for iterating through the I/O Group.

Example

```
SOMPIteratorHandle hit;
SOMPIOGroup thisGroup;
Environment *ev;
/* ... */
hit = _sompNewIterator(thisGroup, ev);
```

Original Class

SOMPIOGroup

Related Information

Methods: sompAddToGroup, sompCount, sompFindByKey, sompFirst, sompFreeIterator, sompNextObjectInGroup, sompRemoveFromGroup

sompNextObjectInGroup Method

Purpose

Gets the next object in an I/O Group of objects that are written/read as a unit.

IDL Syntax

SOMObject sompNextObjectInGroup (
in SOMPIteratorHandle *iteratorHandle***);**

Description

The **sompNextObjectInGroup** method returns the next object in the designated I/O Group. If no more objects are left, the method returns NULL. This method is typically used by implementors of I/O Group Managers.

Parameters

receiver	A pointer to an object of class **SOMPIOGroup** representing the I/O Group.
ev	A pointer to the **Environment** structure for the calling method.
iteratorHandle	The same handle previously returned by the **sompNewIterator** method.

Return Value

The **sompNextObjectInGroup** method returns the next object in the I/O Group, or NULL if no more objects exist.

Example

```
SOMPIOGroup myGroup;
SOMPIteratorHandle hit;
SOMObject obj;
Environment *ev = SOM_CreateLocalEnvironment();

hit = _sompNewIterator(myGroup, ev);
while (obj = _sompNextObjectInGroup(myGroup, ev, hit)) {
   ...
   <use obj>
   ...
   }
_sompFreeIterator(myGroup, ev, hit);
```

Original Class

SOMPIOGroup

Related Information

Methods: sompAddToGroup, sompCount, sompFindByKey, sompFirst, sompFreeIterator, sompNewIterator, sompRemoveFromGroup

sompRemoveFromGroup Method

Purpose

Removes and returns a designed object that was part of an I/O Group.

IDL Syntax

SOMObject sompRemoveFromGroup (
in SOMPIOGroupKey *key***);**

Description

The **sompRemoveFromGroup** method removes from the specified I/O Group the object that corresponds to the designated *key*. The object is also returned. This method is typically used by implementors of I/O Group Managers.

Parameters

receiver	A pointer to an object of class **SOMPIOGroup** representing the I/O Group.
ev	A pointer to the **Environment** structure for the calling method.
key	A **long** previously assigned to uniquely identify the object that is being removed.

Return Value

The **sompRemoveFromGroup** method returns the specified object after removing it from the I/O Group.

Example

```
#include <somp.h>          /* persistence framework */
#include <iogrp.h>         /* SOMPIOGroup class */
#include <direntry.h>      /* an object class to collect */

SOMPTError getErrorCode(Environment *ev);

int main()
{
   int ec = 0;
   Environment *myEnv;
   SOMPTError rc;

   dirEntry name1, name2, anyobj;
   SOMPIOGroup myGroup;
   SOMPIteratorHandle currItem;

   myEnv = somGetGlobalEnvironment();

/* Create the IO Group.
   -------------------- */
   myGroup = SOMPIOGroupNew();

/* Create the directory entries.
   ----------------------------- */
   name1 = dirEntryNew();
   _mkEntry  (name1, ”Roger”, ”555–5085”);

   name2 = dirEntryNew();
   _mkEntry  (name2, ”Hari”, ”555–5079”);

/* Add entries to the group – in other than creation order.
   -------------------------------------------------------- */
   _sompAddToGroup (myGroup, myEnv, name2, 2);
   _sompAddToGroup (myGroup, myEnv, name1, 1);

   anyobj = _sompRemoveFromGroup (myGroup, myEnv, 1);
```

Original Class

SOMPIOGroup

Related Information

Methods: sompAddToGroup, sompCount, sompFindByKey, sompFirst, sompFreeIterator, sompNewIterator, sompNextObjectInGroup

SOMPIOGroupMgrAbstract Class

Description

The **SOMPIOGroupMgrAbstract** class is the abstract definition of an I/O Group Manager. The I/O Group Manager is used by a Persistent Storage Manager to store or restore persistent objects to or from the file system.

File Stem

iogma

Base

SOMObject

Metaclass

SOMClass

Ancestor Classes

SOMObject

New Methods

sompWriteGroup
sompReadGroup
sompGroupExists
sompNewMediaInterface
sompGetMediaInterface
sompFreeMediaInterface
sompReadObjectData
sompObjectInGroup
sompDeleteObjectFromGroup
sompMediaFormatOk
sompInstantiateMediaInterface

sompDeleteObjectFromGroup Method

Purpose

Deletes a given object from a specified group in persistent storage.

IDL Syntax

void sompDeleteObjectFromGroup (
in SOMPPersistentId *objectID***);**

Description

The **sompDeleteObjectFromGroup** method deletes the specified object from this group in persistent storage. This method may raise a **sompException** exception. This method is typically used by implementors of I/O Group Managers.

Parameters

receiver	A pointer to an object of class **SOMPIOGroupMgrAbstract.**
ev	A pointer to the **Environment** structure for the caller.
objectID	A pointer to the ID of the object to be deleted.

Return Value

None.

Example

```
SOMPIOGroupMgrAbstract thisGroupMgr;
Environment *ev;
SOMPPersistentId objectID;
/* ... */
_sompDeleteObjectFromGroup(thisGroupMgr, ev, ObjectID);
```

Original Class

SOMPIOGroupMgrAbstract

Related Information

Methods: sompWriteGroup, **sompReadGroup**, **sompGroupExists**, **sompNewMediaInterface**, **sompGetMediaInterface**, **sompFreeMediaInterface**, **sompReadObjectData**, **sompObjectInGroup**, **sompInstantiateMediaInterface, sompMediaFormatOk**

sompFreeMediaInterface Method

Purpose

Frees a specified I/O Group Manager's interface to storage media.

IDL Syntax

void sompFreeMediaInterface ( );

Description

The **sompFreeMediaInterface** method frees the given I/O Group Manager's media interface. This method is typically overridden by implementors of I/O Group Mangers.

Parameters

receiver A pointer to an object of class **SOMPIOGroupMgrAbstract.**

ev A pointer to the **Environment** structure for the caller.

Return Value

None.

Example

```
SOM_Scope void  SOMLINK somUninit(SOMPTemplate somSelf)
{
    Environment ev;
    SOMPTemplateData *somThis = SOMPTemplateGetData(somSelf);
    SOMPTemplateMethodDebug("SOMPTemplate","somUninit");

    _sompFreeMediaInterface(somSelf, &ev);
}
```

Original Class

SOMPIOGroupMgrAbstract

Related Information

Methods: sompWriteGroup, **sompReadGroup**, **sompGroupExists**, **sompNewMediaInterface**, **sompGetMediaInterface**, **sompReadObjectData**, **sompObjectInGroup**, **sompDeleteObjectFromGroup**, **sompInstantiateMediaInterface**

sompGetMediaInterface Method

Purpose

Obtains a specified I/O group's media interface.

IDL Syntax

SOMPMediaInterfaceAbstract sompGetMediaInterface ( );

Description

The **sompGetMediaInterface** method locates the I/O group's interface to storage media. This method is typically overridden by implementors of I/O Group Mangers.

Parameters

receiver	A pointer to an object of class **SOMPIOGroupMgrAbstract.**
ev	A pointer to the **Environment** structure for the caller.

Return Value

The **sompGetMediaInterface** method returns a pointer to a **SOMPMediaInterfaceAbstract** object. If **sompNewMediaInterface** has not been called first, this method returns NULL. This method also returns NULL after a call to **sompFreeMediaInterface**.

Example

```
SOMPEncoderDecoderAbstract ed;
Environment ev;
SOMObject thisPo;
/* ... */
if (ed)
  _sompEDWrite(ed, ev,
  _sompGetMediaInterface(somSelf, ev), thisPo);
```

Original Class

SOMPIOGroupMgrAbstract

Related Information

Methods: sompWriteGroup, **sompReadGroup**, **sompGroupExists**, **sompNewMediaInterface**, **sompFreeMediaInterface**, **sompReadObjectData**, **sompObjectInGroup**, **sompDeleteObjectFromGroup**, **sompInstantiateMediaInterface, sompMediaFormatOk**

sompGroupExists Method

Purpose

Determines whether an I/O group exists in persistent storage.

IDL Syntax

boolean sompGroupExists (
in string *IOInfo*);

Description

The **sompGroupExists** method checks whether a group exists in persistent storage. This method is typically overridden by implementors of I/O Group Mangers.

Parameters

receiver A pointer to an object of class **SOMPIOGroupMgrAbstract**.

ev A pointer to the **Environment** structure for the caller.

IOInfo Information that a particular (concrete) I/O Group Manager requires to determine whether the I/O group exists on a storage medium.

Return Value

The **sompGroupExists** method returns TRUE if the specified I/O Group exists in persistent storage, and FALSE otherwise.

Example

```
SOMPIOGroupMgrAbstract lclGroupMgr;
boolean exists = FALSE;
Environment ev;
/* ... */
exists = _sompGroupExists(lclGroupMgr, ev, IOGroupName);
```

Original Class

SOMPIOGroupMgrAbstract

Related Information

Methods: sompWriteGroup, **sompReadGroup**, **sompNewMediaInterface**, **sompGetMediaInterface**, **sompFreeMediaInterface**, **sompReadObjectData**, **sompObjectInGroup**, **sompDeleteObjectFromGroup**, **sompInstantiateMediaInterface, sompMediaFormatOk**

sompInstantiateMediaInterface Method

Purpose

Instantiates the proper file media interface used by this I/O Group Manager.

IDL Syntax

SOMPMediaInterfaceAbstract sompInstantiateMediaInterface ( );

Description

The **sompInstantiateMediaInterface** method should be overridden by subclasses to change the media interface that the specified I/O Group Manager uses to write out I/O Groups. The default implementation of this method does nothing. This method is typically overridden by implementors of I/O Group Mangers.

Parameters

receiver A pointer to an object of class **SOMPIOGroupMgrAbstract.**

ev A pointer to the **Environment** structure for the caller.

Return Value

The **sompInstantiateMediaInterface** method returns a pointer to an object of class **SOMPMediaInterfaceAbstract**.

Example

```
SOM_Scope void  SOMLINK sompNewMediaInterface(
  SOMPTemplate somSelf, Environment *ev,
  string IOInfo)
{
SOMPTemplateData *somThis = SOMPTemplateGetData(somSelf);

SOMPTemplateMethodDebug(”SOMPTemplate”,”sompNewMediaInterface”);

if (_mia == NULL) {
  _mia = _sompInstantiateMediaInterface(somSelf, ev);
  _sompInitReadWrite(_mia, ev, IOInfo);
    } /* endif */
}
```

Original Class

SOMPIOGroupMgrAbstract

Related Information

Methods: sompWriteGroup, **sompReadGroup**, **sompGroupExists**, **sompNewMediaInterface**, **sompFreeMediaInterface**, **sompReadObjectData**, **sompObjectInGroup**, **sompDeleteObjectFromGroup**, **sompGetMediaInterface, sompMediaFormatOk**

sompMediaFormatOk Method

Purpose

Checks whether a media format is supported by this I/O Group Manager.

IDL Syntax

boolean sompMediaFormatOk (
in string *mediaFormatName***);**

Description

The **sompMediaFormatOk** method determines whether *mediaFormatName* names a format supported by *receiver*. Every file contains the name of the class of the I/O Group Manager, and this checks the *mediaFormatName* to verify it is the same as the class of the receiver. This method is typically overridden by implementors of I/O Group Mangers.

Parameters

receiver A pointer to the I/O Group Manager.

ev A pointer to the **Environment** structure for the caller.

mediaFormatName
The name of the media format to be tested.

Return Value

The **sompMediaFormatOk** method returns TRUE if the specified media format is supported by the *receiver*; otherwise, it returns FALSE.

Example

```
char formatName[SOMPMAXIDSIZE];
Environment *ev;
SOMPMediaInterfaceAbstract fmi;
SOMPIOGroupMgrAbstract iogma;
/* ... */
_sompReadStringToBuffer(fmi, ev, formatName, BUFFER_SIZE);
if (!(_sompMediaFormatOk(iogma, ev, formatName))) {
         sompRaiseException(ev,
         SOMPERROR_FRAMEWORK_ERROR,
         SOMPERROR_MEDIA_FORMAT_ERROR);
         /* ... */
```

Original Class

SOMPIOGroupMgrAbstract

Related Information

Methods: sompWriteGroup, **sompReadGroup**, **sompGroupExists**, **sompNewMediaInterface**, **sompGetMediaInterface**, **sompFreeMediaInterface**, **sompReadObjectData**, **sompDeleteObjectFromGroup**, **sompInstantiateMediaInterface, sompObjectInGroup**

sompNewMediaInterface Method

Purpose

Initializes a new Media Interface for an I/O Group Manager and prepares it for I/O.

IDL Syntax

void sompNewMediaInterface (
in string *IOInfo***);**

Description

The **sompNewMediaInterface** method initializes a new Media Interface for this I/O Group Manager and prepares it for I/O to storage media. This method must be called prior to calling any other method of the **SOMPIOGroupMgrAbstract** class.

The **sompNewMediaInterface** method may raise a **sompException** exception. This method is typically overridden by implementors of I/O Group Mangers.

Parameters

receiver A pointer to an object of class **SOMPIOGroupMgrAbstract.**

ev A pointer to the **Environment** structure for the caller.

IOInfo A string containing information needed by a particular (concrete) I/O Group Manager to initialize a new Media Interface.

Return Value

None.

Example

```
SOMPAscii myIOGroupMgr;
char IOGroupName[SOMPMAXIDSIZE];
Environment ev;
/* ... */
_sompNewMediaInterface(myIOGroupMgr, ev, IOGroupName);
```

Original Class

SOMPIOGroupMgrAbstract

Related Information

Methods: sompWriteGroup, **sompReadGroup**, **sompGroupExists**, **sompGetMediaInterface**, **sompFreeMediaInterface**, **sompReadObjectData**, **sompObjectInGroup**, **sompDeleteObjectFromGroup**, **sompInstantiateMediaInterface, sompMediaFormatOk**

sompObjectInGroup Method

Purpose

Checks whether a given object exists in a given group in persistent storage.

IDL Syntax

boolean sompObjectInGroup (
in SOMPPersistentId *objectID***);**

Description

The **sompObjectInGroup** method checks whether the given object exists in this group. This method may raise a **sompException** exception. This method is typically overridden by implementors of I/O Group Mangers.

Parameters

receiver	A pointer to an object of class **SOMPIOGroupMgrAbstract.**
ev	A pointer to the **Environment** structure for the caller.
objectID	A pointer to the ID of the object to be checked.

Return Value

The **sompObjectInGroup** method returns TRUE if the object exists in the group. Otherwise, it returns FALSE.

Example

```
SOM_Scope boolean SOMLINK sompWriteGroup(
  SOMPTemplate somSelf, Environment *ev,
  SOMPPersistentObject storeObj)
{
SOMPIteratorHandle hit;
SOMObject thisPo;
SOMPIOGroup thisGroup;
SOMPEncoderDecoderAbstract ed;

SOMPTemplateData *somThis = SOMPTemplateGetData(somSelf);
SOMPTemplateMethodDebug(”SOMPTemplate”,”sompWriteGroup”);

thisGroup = _sompGetIOGroup(storeObj, ev);

hit = _sompNewIterator(thisGroup, ev);

thisPo = _sompNextObjectInGroup(thisGroup, ev, hit)
```

Original Class

SOMPIOGroupMgrAbstract

Related Information

Methods: sompWriteGroup, **sompReadGroup**, **sompGroupExists**, **sompNewMediaInterface**, **sompGetMediaInterface**, **sompFreeMediaInterface**, **sompReadObjectData**, **sompDeleteObjectFromGroup**, **sompInstantiateMediaInterface, sompMediaFormatOk**

sompReadGroup Method

Purpose

Instantiates and initializes the I/O Group of a given persistent object. Persistent data of each object is not read, and the objects are left in an unstable state.

IDL Syntax

SOMPIOGroup sompReadGroup (
in SOMPPersistentId *objectId***);**

Description

The **sompReadGroup** method instantiates and initializes the I/O Group to which the specified persistent object belongs. The instance data of the objects in the I/O Group is not read. The objects are left in an unstable state.

The **sompReadGroup** method may raise a **sompException** exception. This method is typically overridden by implementors of I/O Group Mangers.

Parameters

receiver	A pointer to an object of class **SOMPIOGroupMgrAbstract**.
ev	A pointer to the **Environment** structure for the caller.
objectId	A pointer to a persistent ID designating a persistent object. Persistent IDs take the form: <IOGroupMgrClassName>:<RepositoryName>:<RepositoryOffset>.

Return Value

The **sompReadGroup** method returns the I/O Group of the given persistent object. If an error is encountered, the group is NULL and a **sompException** exception is raised.

Example

```
SOMPAscii myIOGroupMgr;
Environment *ev;
SOMPPersistentId pid = SOMPPersistentIdNew();
/* ... */
myPO = _sompReadGroup(myIOGroupMgr, ev, pid);
```

Original Class

SOMPIOGroupMgrAbstract

Related Information

Methods: sompWriteGroup, **sompGroupExists**, **sompNewMediaInterface**, **sompGetMediaInterface**, **sompFreeMediaInterface**, **sompReadObjectData**, **sompObjectInGroup**, **sompDeleteObjectFromGroup**, **sompInstantiateMediaInterface** **sompMediaFormatOk**

sompReadObjectData Method

Purpose

Reads a persistent object from storage, effectively stabilizing an unstable object.

IDL Syntax

void sompReadObjectData (
in SOMPPersistentObject *thisPo***);**

Description

The **sompReadObjectData** method reads a persistent object from storage and effectively stabilizes the object. This method may raise a **sompException** exception. This method is typically overridden by implementors of I/O Group Mangers.

Parameters

receiver	A pointer to an object of class **SOMPIOGroupMgrAbstract.**
ev	A pointer to the **Environment** structure for the caller.
thisPo	A pointer to an object of class **SOMPPersistentObject.**

Return Value

None.

Example

```
SOMPIOGroupMgrAbstract lclGroupMgr;
SOMObject thisPo;
Environment ev;
/* ... */
_sompReadObjectData(lclGroupMgr, ev, thisPo);
```

Original Class

SOMPIOGroupMgrAbstract

Related Information

Methods: sompWriteGroup, **sompReadGroup**, **sompGroupExists**, **sompNewMediaInterface**, **sompGetMediaInterface**, **sompFreeMediaInterface**, **sompObjectInGroup**, **sompDeleteObjectFromGroup**, **sompInstantiateMediaInterface** **sompMediaFormatOk**

sompWriteGroup Method

Purpose

Stores all of the dirty objects of an I/O Group or an individual object, as determined by the I/O Group Mgr implementor.

IDL Syntax

boolean sompWriteGroup (
in SOMPPersistentObject *thisPo***);**

Description

The **sompWriteGroup** method can store the specified object and all others grouped with it, or can store only the given object. The effective procedure is the choice of the I/O Group Mgr implementor.

To store the object and all others grouped with it, the implementation should first invoke the method **_sompGetIOGroup(***thisPo*, ...**)** to get the group, and then iterate across the group, storing each object. If a group is stored in this manner, the **sompWriteGroup** method should return TRUE.

Conversely, if the I/O Group Mgr is written to store only the individual object, the **sompWriteGroup** method should return FALSE.

Note: When storing objects of the I/O Group, only objects marked as “dirty” need to be stored.

This method may raise a **sompException** exception. This method is typically overridden by implementors of I/O Group Mangers.

Parameters

receiver	A pointer to an object of class **SOMPIOGroupMgrAbstract**.
ev	A pointer to the **Environment** structure for the caller.
thisPo	A pointer to a persistent object.

Return Value

The **sompWriteGroup** method returns TRUE if an I/O group is stored, or FALSE if only the specified object is stored.

Example

```
SOMPAscii myIOGroupMgr;
SOMObject myObj;
Environment ev;
/* ... */
_sompWriteGroup(myIOGroupMgr, ev, myObj);
```

Original Class

SOMPIOGroupMgrAbstract

Related Information

Methods: sompReadGroup, **sompGroupExists**, **sompNewMediaInterface**, **sompGetMediaInterface**, **sompFreeMediaInterface**, **sompReadObjectData**, **sompObjectInGroup**, **sompDeleteObjectFromGroup**, **sompInstantiateMediaInterface, sompMediaFormatOk**

SOMPMediaInterfaceAbstract Class

Description

The **SOMPMediaInterfaceAbstract** class is the abstract class definition for an interface to storage media. The **SOMPFileMediaAbstract** class is derived from this class and adds methods for file-specific I/O.

File Stem

mia

Base

SOMObject

Metaclass

SOMClass

Ancestor Classes

SOMObject

New Methods

sompOpen
sompClose

Overriding Methods

None.

sompClose Method

Purpose

Closes the Media Interface.

IDL Syntax

void sompClose ( );

Description

The **sompClose** method closes the Media Interface. This method may raise a **sompException** exception. This method is typically used by implementors of I/O Group Managers.

Parameters

receiver A pointer to an object of class **SOMPMediaInterfaceAbstract.**

ev A pointer to the **Environment** structure for the caller.

Return Value

None.

Example

```
SOM_Scope void  SOMLINK sompFreeMediaInterface(
   SOMPTemplate somSelf, Environment *ev)
{
SOMPTemplateData *somThis = SOMPTemplateGetData(somSelf);
SOMPTemplateMethodDebug(”SOMPTemplate”,”sompFreeMediaInterface”);

if (_mia) {
   _sompClose(_mia, ev);
   _somFree(_mia);
   _mia = NULL;
} /* endif */
}
```

Original Class

SOMPMediaInterfaceAbstract

Related Information

Methods: sompOpen

sompOpen Method

Purpose

Opens the Media Interface.

IDL Syntax

void sompOpen ( );

Description

The **sompOpen** method opens a file. For the file system, the Media Interface must first be initialized with an invocation of the method **sompInitReadWrite, sompInitReadOnly,** or **sompInitSpecific.** The **sompOpen** method may raise a **sompException** exception. This method is typically used by implementors of I/O Group Managers.

Parameters

receiver	A pointer to an object of class **SOMPMediaInterfaceAbstract**.
ev	A pointer to the **Environment** structure for the caller.

Return Value

None.

Example

```
SOM_Scope void  SOMLINK sompNewMediaInterface(
  SOMPTemplate somSelf, Environment *ev, string IOInfo)
{
SOMPTemplateData *somThis = SOMPTemplateGetData(somSelf);
SOMPTemplateMethodDebug("SOMPTemplate","sompNewMediaInterface");

if (_mia == NULL) {
  _mia = _sompInstantiateMediaInterface(somSelf, ev);
  _sompInitReadWrite(_mia, ev, IOInfo);
  if (ev->_major == NO_EXCEPTION) {
     _sompOpen(_mia, ev);
  } /* endif */
} /* endif */
}
```

Original Class

SOMPMediaInterfaceAbstract

Related Information

Methods: sompClose

SOMPPersistentId Class

Description

The **SOMPPersistentId** class is the class for a persistent ID. A persistent ID is a string of the form:

<IOGroupMgrClassName> : <RepositoryName> : <RepositoryOffset>

For the file system, an example persistent ID would be as follows:

SOMPAscii:/u/otp/misc/animals:0

where "SOMPAscii" is the I/O Group Manager class name, "/u/otp/misc/animals" is the repository name (the path and file name), and "0" is the repository offset (roughly, the record number).

File Stem

pid

Base

SOMUTStringId

Metaclass

SOMClass

Ancestor Classes

SOMUTStringId, SOMObject

New Methods

sompEqualsIOGroupName
sompGetIOGroupMgrClassName
sompGetIOGroupMgrClassNameLen
sompGetIOGroupName
sompGetIOGroupNameLen
sompGetGroupOffset
sompSetIOGroupMgrClassName
sompSetIOGroupName
sompSetGroupOffset

Overriding Methods

somutSetIdString
somutGetIdString

sompEqualsIOGroupName Method

Purpose

Compares two persistent IDs to determine if they are the same.

IDL Syntax

boolean sompEqualsIOGroupName (
in SOMPPersistentId *id***);**

Description

The **sompEqualsIOGroupName** method compares the RepositoryName (path and filename) of one persistent ID against the RepositoryName portion of another specified persistent ID to determine whether they denote the same persistent object. This method is typically used by implementors of I/O Group Managers.

Parameters

receiver	A pointer to an object of class **SOMPPersistentId** representing the persistent ID of a persistent object.
ev	A pointer to the **Environment** structure for the calling method.
id	A pointer to another persistent ID.

Return Value

The **sompEqualsIOGroupName** method returns TRUE if the specified RepositoryNames match, or FALSE otherwise.

Example

```
SOMPPersistentId myId, nearId;
Environment *ev;
/* ... */
if (_sompEqualsIOGroupName (myId, ev, nearId)) {
   /* ... */
```

Original Class

SOMPPersistentId

sompGetGroupOffset Method

Purpose

Gets the offset portion (the RepositoryOffset) of a persistent ID that identifies the storage location of a persistent object.

IDL Syntax

long sompGetGroupOffset ( );

Description

The **sompGetGroupOffset** method gets the offset portion of the persistent ID that is used to reference a persistent object. This is the third portion of a persistent ID; it designates the RepositoryOffset (roughly equivalent to the record number).

A complete persistent ID takes the form:

<IOGroupMgrClassName>:<RepositoryName>:<RepositoryOffset>

This method is typically used by implementors of I/O Group Managers.

Parameters

receiver	A pointer to an object of class **SOMPPersistentId** representing the persistent ID of a persistent object.
ev	A pointer to the **Environment** structure for the calling method.

Return Value

The **sompGetGroupOffset** method returns a **long** that is the value of the offset.

Example

```
long givenOffset;
SOMPPersistentId id;
Environment *ev;
/* ... */
givenOffset = _sompGetGroupOffset(id, ev);
```

Original Class

SOMPPersistentId

Related Information

Methods: sompSetGroupOffset

sompGetIOGroupMgrClassName Method

Purpose

Gets the "I/O Group Manager class" portion of a persistent ID that identifies the storage location of a persistent object.

IDL Syntax

string sompGetIOGroupMgrClassName (
in string *toBuffer***);**

Description

The **sompGetIOGroupMgrClassName** method gets the "I/O Group Manager class" portion of the persistent ID that is used to reference a persistent object.

A complete persistent ID takes the form:

<IOGroupMgrClassName>:<RepositoryName>:<RepositoryOffset>

This method is typically used by client programs.

Parameters

receiver	A pointer to an object of class **SOMPPersistentId** representing the persistent ID of a persistent object.
ev	A pointer to the **Environment** structure for the calling method.
toBuffer	A string identifying the buffer where the I/O Group Manager class name will be placed.

Return Value

The **sompGetIOGroupMgrClassName** method returns the I/O Group Manager class name into the indicated buffer.

Example

```
string tptr;
SOMPPersistentId myId;
Environment *ev;
char idBuffer[SOMPMAXIDSIZE];
/* ... */
tptr = _sompGetIOGroupMgrClassName(myId, ev, (char*)idBuffer);
```

Original Class

SOMPPersistentId

Related Information

Methods: sompSetIOGroupMgrClassName, sompGetIOGroupMgrClassNameLen

sompGetIOGroupMgrClassNameLen Method

Purpose

Gets the length of the “I/O Group Manager class name” portion of a persistent ID that identifies the storage location of a persistent object.

IDL Syntax

short sompGetIOGroupMgrClassNameLen ( );

Description

The **sompGetIOGroupMgrClassNameLen** method gets the length of the “I/O Group Manager class name” portion of the persistent ID that is used to reference a persistent object.

A complete persistent ID takes the form:

<IOGroupMgrClassName>:<RepositoryName>:<RepositoryOffset>

This method is typically used by client programs.

Parameters

receiver A pointer to an object of class **SOMPPersistentId** representing the persistent ID of a persistent object.

ev A pointer to the **Environment** structure for the calling method.

Return Value

The **sompGetIOGroupMgrClassNameLen** method returns a **short** integer designating the length of the I/O Group Manager class name.

Example

```
short len;
string tptr;
SOMPPersistentId myId;
Environment *ev;
string idBuffer;
/* ... */
len = _sompGetIOGroupMgrClassNameLen(myId)
idBuffer = SOMMalloc(len+1);
tptr = _sompGetIOGroupMgrClassName(myId, ev, (char*)idBuffer);
```

Original Class

SOMPPersistentId

Related Information

Methods: sompGetIOGroupMgrClassName

sompGetIOGroupName Method

Purpose

Gets the “I/O Group name” portion (the RepositoryName) of a persistent ID that identifies the storage location of a persistent object.

IDL Syntax

string sompGetIOGroupName (
in string *toBuffer***);**

Description

The **sompGetIOGroupName** method gets the “I/O Group name” portion of the persistent ID that is used to reference a persistent object. This is the second portion of a persistent ID; it designates the RepositoryName (the path and filename portion of the ID).

A complete persistent ID takes the form:

<IOGroupMgrClassName>:<RepositoryName>:<RepositoryOffset>

This method is typically used by implementors of I/O Group Managers.

Parameters

receiver	A pointer to an object of class **SOMPPersistentId** representing the persistent ID of a persistent object.
ev	A pointer to the **Environment** structure for the calling method.
toBuffer	A string identifying the buffer where the RepositoryName (path/filename) will be placed.

Return Value

The **sompGetIOGroupName** method returns the RepositoryName into the indicated buffer.

Example

```
string fp;
Environment *ev;
char idBuff[SOMPMAXIDSIZE];
SOMPPersistentId pid;
/* ... */
fp  = _sompGetIOGroupName (pid, ev, idBuff);
```

Original Class

SOMPPersistentId

Related Information

Methods: sompSetIOGroupName, sompGetIOGroupNameLen

sompGetIOGroupNameLen Method

Purpose

Gets the length of the “I/O Group name” portion (the RepositoryName) of a persistent ID that identifies the storage location of a persistent object.

IDL Syntax

short sompGetIOGroupNameLen ( );

Description

The **sompGetIOGroupNameLen** method gets the length of the “I/O Group name” portion of the persistent ID that is used to reference a persistent object. This is the second portion of a persistent ID; it designates the RepositoryName (the path and filename portion of the ID).

A complete persistent ID takes the form:

<IOGroupMgrClassName>:<RepositoryName>:<RepositoryOffset>

This method is typically used by implementors of I/O Group Managers.

Parameters

receiver	A pointer to an object of class **SOMPPersistentId** representing the persistent ID of a persistent object.
ev	A pointer to the **Environment** structure for the calling method.

Return Value

The **sompGetIOGroupNameLen** method returns a **short** integer designating the length of the RepositoryName.

Example

```
string fp;
Environment *ev;
string idBuff;
SOMPPersistentId pid;
short len;
/* ... */
len = _sompGetIOGroupNameLen(pid);
idBuff = SOMMalloc(len +1);
fp  = _sompGetIOGroupName (pid, ev, idBuff);
```

Original Class

SOMPPersistentId

Related Information

Methods: sompGetIOGroupName

sompSetGroupOffset Method

Purpose

Sets the offset portion (the RepositoryOffset) of a persistent ID that will identify the storage location of a persistent object.

IDL Syntax

void sompSetGroupOffset (
in long *offset***);**

Description

The **sompSetGroupOffset** method sets the offset portion of the persistent ID that will be used to reference a persistent object. This is the third portion of a persistent ID; it designates the RepositoryOffset (roughly equivalent to the record number).

A complete persistent ID takes the form:
<IOGroupMgrClassName>:<RepositoryName>:<RepositoryOffset>

The **sompSetGroupOffset** method may raise a **sompException** exception. This method is typically used by client programs.

Parameters

receiver	A pointer to an object of class **SOMPPersistentId** representing the persistent ID of a persistent object.
ev	A pointer to the **Environment** structure for the calling method.
offset	A **long** integer defining the final portion of the persistent ID.

Return Value

None.

Example

```
PersistentDog dog = PersistentDogNew();
SOMPPersistentId pid = SOMPPersistentIdNew();
Environment *ev;
/* ... */
_sompSetIOGroupMgrClassName(pid, ev, ”SOMPBinary”);
_sompSetIOGroupName(pid, ev, ”/u/roger/dogs/dog1.dog”);
_sompSetGroupOffset(pid, ev, 0);
```

Original Class

SOMPPersistentId

Related Information

Methods: sompGetGroupOffset

sompSetIOGroupMgrClassName Method

Purpose

Sets the “I/O Group Manager class” portion of a persistent ID that will identify the storage location of a persistent object.

IDL Syntax

void sompSetIOGroupMgrClassName (
in string *newName***);**

Description

The **sompSetIOGroupMgrClassName** method sets the “I/O Group Manager class” portion of the persistent ID that will be used to reference a persistent object.

A complete persistent ID takes the form:

<IOGroupMgrClassName>:<RepositoryName>:<RepositoryOffset>

The **sompSetIOGroupMgrClassName** method may raise a **sompException** exception. This method is typically used by client programs.

Parameters

receiver — A pointer to an object of class **SOMPPersistentId** representing the persistent ID of a persistent object.

ev — A pointer to the **Environment** structure for the calling method.

newName — A string defining the initial portion of the persistent ID. The initial portion is the class name of the I/O Group Manager of the persistent object.

Return Value

None.

Example

```
PersistentDog dog = PersistentDogNew();
SOMPPersistentId pid = SOMPPersistentIdNew();
Environment *ev;
/* ... */
_sompSetIOGroupMgrClassName(pid, ev, ”SOMPBinary”);
_sompSetIOGroupName(pid, ev, ”/u/roger/dogs/dog1.dog”);
_sompSetGroupOffset(pid, ev, 0);
```

Original Class

SOMPPersistentId

Related Information

Methods: sompGetIOGroupMgrClassName

sompSetIOGroupName Method

Purpose

Sets the “I/O Group name” portion (the RepositoryName) of a persistent ID that will identify the storage location of a persistent object.

IDL Syntax

void sompSetIOGroupName (
in string *newName***);**

Description

The **sompSetIOGroupName** method sets the “I/O Group name” portion of the persistent ID that will be used to reference a persistent object. This is the second portion of a persistent ID; it designates the Repository Name (the path and filename portion of the ID).

A complete persistent ID takes the form:
<IOGroupMgrClassName>:<RepositoryName>:<RepositoryOffset>

The **sompSetIOGroupName** method may raise a **sompException** exception. This method is typically used by client programs.

Parameters

receiver	A pointer to an object of class **SOMPPersistentId** representing the persistent ID of a persistent object.
ev	A pointer to the **Environment** structure for the calling method.
newName	A string defining the second portion (the path and filename) of the persistent ID.

Return Value

None.

Example

```
PersistentDog dog = PersistentDogNew();
SOMPPersistentId pid = SOMPPersistentIdNew();
Environment *ev;
/* ... */
_sompSetIOGroupMgrClassName(pid, ev, ”SOMPBinary”);
_sompSetIOGroupName(pid, ev, ”/u/roger/dogs/dog1.dog”);
_sompSetGroupOffset(pid, ev, 0);
```

Original Class

SOMPPersistentId

Related Information

Methods: sompGetIOGroupName

SOMPPersistentObject Class

Description

This is the concrete base class for persistent objects. A persistent object has the following responsibilities:

- To manage object IDs.
- To manage persistent pointers (i.e., persistent child objects).
- To maintain persistent state (e.g., stable or unstable)
- To manage the Encoder/Decoder.

File Stem

po

Base

SOMObject

Metaclass

M_SOMPPersistentObject

Ancestor Classes

SOMObject

New Methods

sompInitNextAvail
sompInitNearObject
sompInitGivenId
sompInitIOGroup
sompGetIOGroup
sompGetPersistentId
sompGetPersistentIdString
sompGetRelativeIdString
sompGetEncoderDecoder
sompFreeEncoderDecoder
sompSetEncoderDecoderName
sompGetEncoderDecoderName
sompSetState
sompCheckState
sompClearState
sompEquals
sompActivated
sompPassivate
sompSetDirty
sompIsDirty
sompGetDirty
sompMarkForCompaction

Overriding Methods

somInit
somUninit
somDumpSelfInt

sompActivated Method

Purpose

Notifies a persistent object that its persistent data has just been read.

IDL Syntax

void sompActivated ( );

Description

The **sompActivated** method is called by the Persistence Framework after a persistent object has been restored. This enables a client to perform any needed initializations prior to using the persistent object.

If not overridden, this method sets the state of the object to SOMP_STATE_STABLE**.** This method is typically overridden by implementors of persistent classes.

Parameters

receiver	A pointer to an object of class **SOMPPersistentObject**.
ev	A pointer to the **Environment** structure for the caller.

Return Value

None.

Example

```
SOM_Scope void  SOMLINK sompActivated(
  SOMPPersistentObject somSelf,
  Environment *ev)
{
SOMPPersistentObjectData *somThis =
    SOMPPersistentObjectGetData(somSelf);
SOMPPersistentObjectMethodDebug(
   ”SOMPPersistentObject”,”sompActivated”);

_parent_sompActivated(somSelf);
_transientData = _persistentData;
}
```

Original Class

SOMPPersistentObject

Related Information

Methods: sompPassivate

sompCheckState Method

Purpose

Checks whether an object is in a specified state.

IDL Syntax

boolean sompCheckState (
in unsigned short *state***);**

Description

The **sompCheckState** method checks the receiver's *state*. Possible values for *state* include the following persistent object states:

SOMP_STATE_UNDEFINED
: Indicates that the object exists, but has not yet been initialized.

SOMP_STATE_UNSTABLE
: Indicates that the object was restored in the name space without reading the instance data.

SOMP_STATE_STABLE
: Indicates that the object exists, that instance data is complete, and that the object is ready to use.

SOMP_STATE_DIRTY
: Indicates that the object can be stored to persistent media.

SOMP_STATE_COMPACT
: Indicates that the entire I/O Group should be compacted the next time this object is stored.

This method is typically used by implementors of I/O Group Managers.

Parameters

receiver	A pointer to an object of class **SOMPPersistentObject.**
ev	A pointer to the **Environment** structure for the caller.
state	An unsigned short designating a persistent object state.

Return Value

The **sompCheckState** method returns TRUE if this object is currently in the specified state. Otherwise, it returns FALSE.

Example

```
PersistentDog dog1;
Environment *ev;
/* ... */
if (_sompCheckState(dog1, ev, SOMP_STATE_UNDEFINED))
   printf ("dog 1 SOMP_STATE_UNDEFINED is TRUE\n");
```

Original Class

SOMPPersistentObject

Related Information

Methods: sompSetState, sompClearState, sompSetDirty

sompClearState Method

Purpose

Clears the specified state.

IDL Syntax

void sompClearState (
in unsigned short *state***);**

Description

The **sompClearState** method clears the receiver's *state*. Possible values for *state* include the following persistent object states:

SOMP_STATE_UNDEFINED
: Indicates that the object exists, but has not yet been initialized.

SOMP_STATE_UNSTABLE
: Indicates the object was restored in the name space without reading the instance data.

SOMP_STATE_STABLE
: Indicates that the object exists, that instance data is complete, and that the object is ready to use.

SOMP_STATE_DIRTY
: Indicates that the in-memory object has been changed and should be stored to persistent media when **sompStoreObject** is called on the persistent storage manager.

SOMP_STATE_COMPACT
: Indicates that the entire I/O Group should be compacted the next time this object is stored.

This method is typically used by implementors of I/O Group Managers.

Parameters

receiver	A pointer to an object of class **SOMPPersistentObject.**
ev	A pointer to the **Environment** structure for the caller.
state	An unsigned short designating a persistent object state.

Return Value

None.

Example

```
SOMObject thisPo;
Environment *ev;
/* ... */
_sompClearState(thisPo, ev, SOMP_STATE_DIRTY);
```

Original Class

SOMPPersistentObject

Related Information

Methods: sompSetState, sompCheckState, sompSetDirty

sompEquals Method

Purpose

Checks whether an object's ID is equal to a specified ID.

IDL Syntax

boolean sompEquals (
in SOMPPersistentId *otherId***);**

Description

The **sompEquals** method determines whether this object's ID is equivalent to the given ID. This method is typically used by client programs.

Parameters

receiver	A pointer to an object of class **SOMPPersistentObject.**
ev	A pointer to the **Environment** structure for the caller.
otherId	A pointer to an object of class **SOMPPersistentId.**

Return Value

The **sompEquals** method returns TRUE if the ID's are equivalent. Otherwise, it returns FALSE.

Example

```
SOMObject cmpObj;
SOMPPersistentId;
Environment *ev;
/* ... */
if (_sompEquals(cmpObj, ev, lookfor))
   /* ... */
```

Original Class

SOMPPersistentObject

Related Information

Methods: sompGetPersistentIdString

sompFreeEncoderDecoder Method

Purpose

Frees a persistent object's Encoder/Decoder.

IDL Syntax

void sompFreeEncoderDecoder ( );

Description

The **sompFreeEncoderDecoder** method frees the *receiver* object's Encoder/Decoder.This method is typically used by implementors of I/O Group Managers.

Parameters

receiver	A pointer to an object of class **SOMPPersistentObject.**
ev	A pointer to the **Environment** structure for the caller.

Return Value

None.

Example

```
SOMPPersistentObject thisPo;
Environment *ev;
/* ... */
_sompFreeEncoderDecoder(thisPo, ev);
```

Original Class

SOMPPersistentObject

Related Information

Methods: sompGetEncoderDecoder, sompSetEncoderDecoderName, sompGetEncoderDecoderName, sompSetClassLevelEncoderDecoderName, sompGetClassLevelEncoderDecoderName

sompGetDirty Method

Purpose

Determines whether the state of an object is “dirty.”

IDL Syntax

boolean sompGetDirty ( );

Description

The **sompGetDirty** method checks a persistent object to determine whether its dirty bit has been set. This method is typically used by implementors of I/O Group Managers.

Parameters

receiver	A pointer to an object of class **SOMPPersistentObject.**
ev	A pointer to the **Environment** structure for the caller.

Return Value

The **sompGetDirty** method returns TRUE if the dirty bit is set, or FALSE otherwise.

Example

```
SOM_Scope boolean  SOMLINK sompIsDirty(
   PersistentDog somSelf, Environment *ev)
{
PersistentDogData *somThis = PersistentDogGetData(somSelf);
PersistentDogMethodDebug(”PersistentDog”,”sompIsDirty”);
return (_sompGetDirty(somSelf, ev));
}
```

Original Class

SOMPPersistentObject

Related Information

Methods: sompIsDirty, sompSetDirty

sompGetEncoderDecoder Method

Purpose

Returns an Encoder/Decoder.

IDL Syntax

SOMPEncoderDecoderAbstract sompGetEncoderDecoder ( );

Description

The **sompGetEncoderDecoder** method returns the appropriate encoder to use in storing/restoring the receiver. This method is typically used by implementors of I/O Group Managers.

Parameters

receiver	A pointer to an object of class **SOMPPersistentObject.**
ev	A pointer to the **Environment** structure for the caller.

Return Value

The **sompGetEncoderDecoder** method returns a pointer to this object's Encoder/Decoder. If the Encoder/Decoder cannot be instantiated, the method returns NULL.

Example

```
SOMPEncoderDecoderAbstract ed;
SOMObject thisPo;
Environment *ev;
/* ... */
ed = _sompGetEncoderDecoder(thisPo, ev);
```

Original Class

SOMPPersistentObject

Related Information

Methods: sompFreeEncoderDecoder, sompSetEncoderDecoderName, sompGetEncoderDecoderName, sompSetClassLevelEncoderDecoderName, sompGetClassLevelEncoderDecoderName

sompGetEncoderDecoderName Method

Purpose

Gets the class name of the appropriate Encoder/Decoder.

IDL Syntax

string sompGetEncoderDecoderName ( );

Description

The **sompGetEncoderDecoderName** method gets the class name of the Encoder/Decoder. This method is typically used by implementors of I/O Group Managers.

Parameters

receiver	A pointer to an object of class **SOMPPersistentObject**.
ev	A pointer to the **Environment** structure for the caller.

Return Value

The **sompGetEncoderDecoderName** method returns the class name of the Encoder/Decoder.

Example

```
string edName;
SOMPPersistentObject po;
Environment *ev;
/* ... */
edName = sompGetEncoderDecoderName(po, ev);
```

Original Class

SOMPPersistentObject

Related Information

Methods: sompSetEncoderDecoderName, sompGetClassLevelEncoderDecoderName, sompSetClassLevelEncoderDecoderName

sompGetIOGroup Method

Purpose

Gets a pointer to the **SOMPIOGroup** object for a given persistent object.

IDL Syntax

SOMPIOGroup sompGetIOGroup ( );

Description

The **sompGetIOGroup** method returns a pointer to the I/O Group object associated with the receiver. The returned group contains this object and any other objects in the same I/O Group. This method is typically used by implementors of I/O Group Managers.

Parameters

receiver	A pointer to an object of class **SOMPPersistentObject.**
ev	A pointer to the **Environment** structure for the caller.

Return Value

The **sompGetIOGroup** method returns a pointer to a **SOMPIOGroup** object.

Example

```
SOM_Scope boolean SOMLINK sompWriteGroup(SOMPTemplate somSelf,
Environment *ev,
SOMPPersistentObject storeObj)
{
SOMPIteratorHandle hit;
SOMObject thisPo;
SOMPIOGroup thisGroup;
SOMPEncoderDecoderAbstract ed;

SOMPTemplateData *somThis = SOMPTemplateGetData(somSelf);
SOMPTemplateMethodDebug(”SOMPTemplate”,”sompWriteGroup”);

thisGroup = _sompGetIOGroup(storeObj, ev);

hit = _sompNewIterator(thisGroup, ev);

while ((thisPo = _sompNextObjectInGroup
       (thisGroup, ev, hit))!=NULL)
        /* ... */
```

Original Class

SOMPPersistentObject

Related Information

Methods: sompInitIOGroup, sompInitNearObject

sompGetPersistentId Method

Purpose

Gets the ID for a persistent object, returning it as a persistent ID.

IDL Syntax

SOMPPersistentId sompGetPersistentId ( );

Description

The **sompGetPersistentId** method gets the ID of a persistent object, returning it as a persistent ID. This method is typically used by client programs.

Parameters

receiver	A pointer to an object of class **SOMPPersistentObject.**
ev	A pointer to the **Environment** structure for the caller.

Return Value

The **sompGetPersistentId** method returns the persistent ID for a specified persistent object.

Example

```
phoneDir mylist;
static char idBuff[SOMPMAXIDSIZE];
Environment *ev;
string idString;
/* ... */
idString = _somutGetIdString(
    _sompGetPersistentId(mylist, ev), ev, idBuff);
```

Original Class

SOMPPersistentObject

Related Information

Methods: sompGetPersistentIdString

sompGetPersistentIdString Method

Purpose

Gets the persistent ID for a persistent object, returning the ID as a string.

IDL Syntax

string sompGetPersistentIdString (
in string *outBuf***);**

Description

The **sompGetPersistentIdString** method gets the fully-qualified persistent ID for *receiver,* stores it in *outBuf,* and returns *outBuf.* The caller must allocate sufficient memory in *outBuf* and free the storage when appropriate. This method is typically used by client programs.

Parameters

receiver	A pointer to an object of class **SOMPPersistentObject** whose ID is needed.
ev	A pointer to the **Environment** structure for the caller.
outBuf	A pointer to the area in memory where the ID is to be stored.

Return Value

The **sompGetPersistentIdString** method returns *outBuf,* which now contains a string that is the persistent ID for the specified persistent object.

Example

```
phoneDir mylist;
static char idBuff[SOMPMAXIDSIZE];
Environment *ev;
string idString;
/* ... */
idString = _somutGetIdString(
    _sompGetPersistentId(mylist, ev), ev, idBuff);
```

Original Class

SOMPPersistentObject

Related Information

Methods: sompGetPersistentId

sompGetRelativeIdString Method

Purpose

Constructs a relative persistent ID for a persistent object, returning it as a string.

IDL Syntax

string sompGetRelativeIdString (
in SOMObject *ifNearThisObj,*
in string *relativeIdString***);**

Description

The **sompGetRelativeIdString** method constructs a relative persistent ID for the receiving persistent object. If the *receiver* is in the same I/O group as the persistent object specified by *ifNearThisObj*, the constructed ID has the form:

<IOGroupMgrClassName>:$RELATIVE:<offset>.

Otherwise, it has the fully qualified form:

<IOGroupMgrClassName>:<path/filename>:<offset>.

The caller must allocate sufficient memory in *relativeIdString* and free the storage when appropriate. This method is typically used by implementors of Encoders/Decoders.

Parameters

receiver A pointer to an object of class **SOMPPersistentObject** whose ID is needed.

ev A pointer to the **Environment** structure for the caller.

ifNearThisObj A pointer to another persistent object.

relativeIdString A character buffer in which the Id will be constructed.

Return Value

The **sompGetPersistentIdString** method returns *relativeIdString,* which now contains the relative persistent ID for the specified persistent object.

Example

```
SOMObject so;
Environment *ev;
SOMObject parentObject
string buffer;
/* ... */
buffer = SOMMalloc(SOMPMAXIDSIZE * sizeof(char));
outStr = _sompGetRelativeIdString (so, ev, parentObject, buffer);
```

Original Class

SOMPPersistentObject

Related Information

Methods: sompGetPersistentId, sompGetPersistentIdString

sompInitGivenId Method

Purpose

Copies information from an ID parameter into the object's persistent ID.

IDL Syntax

void sompInitGivenId (
in SOMPPersistentId *thisId***);**

Description

The **sompInitGivenId** method copies information from a specified ID parameter *thisId* into the object's persistent ID. This method may raise a **sompException** exception. This method is typically used by client programs.

Parameters

receiver	A pointer to an object of class **SOMPPersistentObject**, representing a persistent object to which the specified ID should be assigned.
ev	A pointer to the **Environment** structure for the caller.
thisId	A pointer to an object of class **SOMPPersistentId**, which represents the persistent ID to assign to the specified persistent object.

Return Value

None.

Example

```
phoneDir  mylist;
SOMPPersistentId pid;
mylist = phoneDirNew();
Environment *ev;

pid = SOMPPersistentIdNew();
_somutSetIdString(pid, ev, "SOMPAscii:./pdata/phoneDir:0");
checkError(ev);
_sompInitGivenId (mylist, ev, pid);
```

Original Class

SOMPPersistentObject

Related Information

Methods: sompInitNextAvail, sompInitNearObject

sompInitIOGroup Method

Purpose

Initializes the I/O group of a specified persistent object.

IDL Syntax

SOMPIOGroup sompInitIOGroup (
in SOMObject *nearPersistentObj***);**

Description

The **sompInitIOGroup** method initializes the I/O group for the receiver. It is used after the object's persistent ID was created with one of the methods **sompInitNextAvail, sompInitNearObject,** or **sompInitGivenId**. Each persistent object is a member of an I/O Group, which can consist of one or more persistent objects.

The *nearPersistentObj* argument determines the I/O Group to which this persistent *receiver* object will belong. If the given *nearPersistentObj* is NULL, a new I/O Group is instantiated. Otherwise, *nearPersistentObj* should be a persistent object with an existing I/O Group. The specified *receiver* object is then made a member of the *nearPersistentObj*'s I/O Group. This method is typically overridden by implementors of persistent classes.

Parameters

receiver	A pointer to an object of class **SOMPPersistentObject**.
ev	A pointer to the **Environment** structure for the caller.
nearPersistentObj	A pointer to an existing persistent object.

Return Value

The **sompInitIOGroup** method returns a pointer to an object of class **SOMPIOGroup.** The pointer represents the offset of the specified object within its new I/O Group.

Example

```
SOMPPersistentObject neighbor, self;
Environment *ev;
/* ... */
_sompInitIOGroup(somSelf, ev, neighbor);
```

Original Class

SOMPPersistentObject

Related Information

Methods: sompInitNextAvail, sompInitNearObject, sompInitGivenId, sompGetIOGroup

sompInitNearObject Method

Purpose

Gives a specified persistent object a persistent ID nearby the ID of another specified persistent object.

IDL Syntax

void sompInitNearObject (
in SOMPPersistentObject *nearObj***);**

Description

The **sompInitNearObject** method gives a specified persistent object a persistent ID nearby the ID of another specified persistent object. This method may raise a **sompException** exception. This method is typically used by client programs.

Parameters

receiver A pointer to an object of class **SOMPPersistentObject**, representing a persistent object to which an ID should be assigned.

ev A pointer to the **Environment** structure for the caller.

nearObj A pointer to an object of class **SOMPPersistentObject**, which represents the persistent object whose persistent ID is to be nearby the assigned ID of the receiver.

Return Value

None.

Example

```
dirEntry  name1, name2;
phoneDir  mylist;
SOMPPersistentId pid;
Environment *ev;

mylist = phoneDirNew();
pid = SOMPPersistentIdNew();
_somutSetIdString(pid, ev, "SOMPAscii:./pdata/phoneDir:0");
checkError(ev);
_sompInitGivenId (mylist, ev, pid);
checkError(ev);
_somFree (pid);

name1 = dirEntryNew();
_mkEntry  (name1, "Roger", "555-5085");
 _sompInitNearObject(name1, ev, mylist);
checkError(ev);
_addEntry (mylist, name1);
```

Original Class

SOMPPersistentObject

Related Information

Methods: sompInitNextAvail, sompInitGivenId

sompInitNextAvail Method

Purpose

Gives a specified persistent object the next available persistent ID, as determined by the IdAssigner.

IDL Syntax

void sompInitNextAvail (
in SOMPIdAssigner *thisAssigner***);**

Description

The **sompInitNextAvail** method gives a specified persistent object the next available persistent ID, as determined by the specified ID Assigner. This method may raise a **sompException** exception. This method is typically used by client programs.

Parameters

receiver	A pointer to an object of class **SOMPPersistentObject**, representing a persistent object to which an ID should be assigned.
ev	A pointer to the **Environment** structure for the caller.
thisAssigner	A pointer to an object of class **SOMPIdAssigner**, which represents the ID Assigner that will provide the next available persistent ID.

Return Value

None.

Example

```
PersistentDog dog = PersistentDogNew();
SOMPIdAssigner systemIdAssigner = SOMPIdAssignerNew();
Environment *ev;
/* ... */
_sompInitNextAvail(dog, ev, systemIdAssigner);
```

Original Class

SOMPPersistentObject

Related Information

Methods: sompInitNearObject, sompInitGivenId

sompIsDirty Method

Purpose

Tells whether an object should be considered dirty.

IDL Syntax

boolean sompIsDirty ( );

Description

The **sompIsDirty** method is used for optimization. By default, the method always returns TRUE. Optimized persistent objects can override this implementation to call **sompGetDirty**, and then be sure to invoke **sompSetDirty** whenever the object is updated. This method is typically overridden by implementors of persistent classes.

Parameters

receiver	A pointer to an object of class **SOMPPersistentObject.**
ev	A pointer to the **Environment** structure for the caller.

Return Value

The **sompIsDirty** method by default returns TRUE, unless it is overridden to call **sompGetDirty.**

Example

```
SOM_Scope boolean  SOMLINK sompIsDirty(PersistentDog somSelf,
  Environment *ev)
{
  PersistentDogData *somThis = PersistentDogGetData(somSelf);
  PersistentDogMethodDebug(”PersistentDog”,”sompIsDirty”);
  return (_sompGetDirty(somSelf, ev));
}
```

Original Class

SOMPPersistentObject

Related Information

Methods: sompGetDirty, sompSetDirty

sompMarkForCompaction Method

Purpose

Marks a persistent object's I/O Group for compaction the next time the object is saved.

IDL Syntax

void sompMarkForCompaction ( );

Description

The **sompMarkForCompaction** method marks a persistent object's entire I/O Group to be compacted the next time the object is saved. Compaction removes any unused space from the file containing the I/O Group. This method is typically used by client programs.

Parameters

receiver	A pointer to an object of class **SOMPPersistentObject** representing the persistent object whose I/O Group is to be marked for compaction.
ev	A pointer to the **Environment** structure for the caller.

Return Value

None.

Example

```
PersistentDog myDog;
Environment *ev;
/* ... */
_sompMarkForCompaction(myDog, ev);
```

Original Class

SOMPPersistentObject

Related Information

Methods: sompSetDirty

sompPassivate Method

Purpose

Notifies a persistent object that its persistent data is about to be stored.

IDL Syntax

void sompPassivate ( );

Description

The **sompPassivate** method is called by the Persistence Framework just prior to saving the persistent data of a persistent object. This enables a client to tidy up any data necessary in the persistent object (or its embedded objects) before the data is saved.

If not overridden, this method has no effect. This method is typically overridden by implementors of persistent classes.

Parameters

receiver	A pointer to the object to be passivated.
ev	A pointer to the **Environment** structure for the caller.

Return Value

None.

Example

```
SOM_Scope void  SOMLINK sompPassivate(
  SOMPPersistentObject somSelf,
  Environment *ev)
{
SOMPPersistentObjectData *somThis =
    SOMPPersistentObjectGetData(somSelf);
SOMPPersistentObjectMethodDebug(
   ”SOMPPersistentObject”,”sompPassivate);

_persistentData = _transientData ;
_parent_sompPassivate(somSelf, ev);
}
```

Original Class

SOMPPersistentObject

Related Information

Methods: sompActivated

sompSetDirty Method

Purpose

Sets the state of a persistent object to “dirty.”

IDL Syntax

void sompSetDirty ( );

Description

The **sompSetDirty** method sets the state of the receiving object to “dirty.” This method is typically used by implementors of I/O Group Managers.

Parameters

receiver A pointer to an object of class **SOMPPersistentObject** representing the persistent object to be marked “dirty.”

ev A pointer to the **Environment** structure for the caller.

Return Value

None.

Example

```
SOM_Scope long  SOMLINK addEntry(phoneDir somSelf,
  dirEntry entry)
{
  Environment *ev;

  phoneDirData *somThis = phoneDirGetData(somSelf);
  phoneDirMethodDebug(”phoneDir”,”addEntry”);

  /* ... */
  _sompSetDirty (somSelf, ev);
  /* ... */
}
```

Original Class

SOMPPersistentObject

Related Information:

Methods: sompGetDirty, sompIsDirty, sompMarkForCompaction

sompSetEncoderDecoderName Method

Purpose

Binds an Encoder/Decoder class name to a persistent object.

IDL Syntax

void sompSetEncoderDecoderName (
in string *myName***);**

Description

The **sompSetEncoderDecoderName** method binds an Encoder/Decoder class name (specified by *myName*) to a persistent object (*receiver*). If this method is not invoked, the Encoder/Decoder class name is the one returned by the **sompGetClassLevelEncoderDecoderName** method. This method is typically used by client programs.

Parameters

receiver	A pointer to an object of class **SOMPPersistentObject**.
ev	A pointer to the **Environment** structure for the caller.
myName	The name of a class derived from class **SOMPEncoderDecoderAbstract**.

Return Value

None.

Example

```
SOMPPersistentId pid;
dirEntry  name1, name2;
phoneDir  mylist;
string    fp;
Environment *ev;

mylist = phoneDirNew();
name1  = dirEntryNew();
name2  = dirEntryNew();

_sompSetEncoderDecoderName (mylist, ev, ”dirED”);
_sompSetClassLevelEncoderDecoderName (_somGetClass(name1), ev,
                                      ”entryED”);
```

Original Class

SOMPPersistentObject

Related Information

Methods: sompGetEncoderDecoderName, sompGetClassLevelEncoderDecoderName, sompSetClassLevelEncoderDecoderName

sompSetState Method

Purpose

Sets a persistent object to a specified state.

IDL Syntax

void sompSetState (
in unsigned short *state***);**

Description

The **sompSetState** method sets the receiver's state. Possible values for *state* include the following persistent object states:

SOMP_STATE_UNDEFINED
: Indicates that the object exists, but has not yet been initialized.

SOMP_STATE_UNSTABLE
: Indicates the object was restored in the name space without reading the instance data.

SOMP_STATE_STABLE
: Indicates that the object exists, that instance data is complete, and that the object is ready to use.

SOMP_STATE_DIRTY
: Indicates that the object can be stored to persistent media.

SOMP_STATE_COMPACT
: Indicates that the entire I/O Group should be compacted the next time this object is stored.

This method is typically used by implementors of I/O Group Managers.

Parameters

receiver	A pointer to an object of class **SOMPPersistentObject**.
ev	A pointer to the **Environment** structure for the caller.
state	An unsigned short designating a persistent object state.

Return Value

None.

Example

```
SOMPPersistentObject thisPO;
Environment *ev;
/* ... */
_sompSetState(thisPo, ev, SOMP_STATE_UNSTABLE);
/* ... */
```

Original Class

SOMPPersistentObject

Related Information

Methods: sompCheckState, sompClearState, sompSetDirty

SOMPPersistentStorageMgr Class

Description

The **SOMPPersistentStorageMgr** class is the base class for a Persistent Storage Manager. The Persistent Storage Manager object is the primary interface through which clients of the Persistence Framework read and write objects to and from storage media.

Only a single instance of this class exists in the Persistence Framework, by virtue of its metaclass **SOMMSingleInstance.**

Important: The client program must *not* **somFree** the **SOMPPersistentStorageMgr** object. If the client does free it, the Framework will forget about previously initialized and restored objects, and a segmentation fault can result.

File Stem

psma

Base

SOMObject

Metaclass

SOMMSingleInstance

Ancestor Classes

SOMObject

New Methods

sompStoreObject
sompRestoreObject
sompStoreObjectWithoutChildren
sompRestoreObjectWithoutChildren
sompObjectExists
sompDeleteObject
sompAddObjectToWriteSet
sompAddIdToReadSet
sompRestoreObjectFromIdString

Overriding Methods

somInit
somUninit

sompAddIdToReadSet Method

Purpose

Adds the specified persistent ID to the set of IDs that represent persistent objects to be restored by the **sompRestoreObject** method as part of this read request.

IDL Syntax

void sompAddIdToReadSet (
in SOMPPersistentId *objectId***);**

Description

The **sompAddIdToReadSet** method adds the given persistent ID to the set of IDs that represent persistent objects to be restored by the **sompRestoreObject** method (the *read set*) as part of this read request. If an error occurs, the ID is not added to the read set.

IDs can only be added to the read set following a call to **sompRestoreObject.** Encoder/decoder writers can make use of this method as follows:

- Decoder reads object ID in string form
- Decoder creates a persistent ID object based on the string
- Decoder calls **sompAddIdToReadSet** using the persistent ID.

The **sompAddIdToReadSet** method may raise a **sompException** exception. This method is typically used by implementors of Encoders/Decoders.

Parameters

receiver	A pointer to an object of class **SOMPPersistentStorageMgr.**
ev	A pointer to the **Environment** structure for the caller.
objectId	A pointer to an object of class **SOMPPersistentId** representing the ID to be added to the read set**.**

Return Value

None.

Example

```
SOMPersistentObject so
Environment *ev;
SOMPPersistentStorageMgr *psm;
/* ... */
_sompAddIdToReadSet(psm, ev, _sompGetPersistentId(*so, ev));
```

Original Class

SOMPPersistentStorageMgr

Related Information

Methods: sompStoreObject, **sompStoreObjectWithoutChildren**, **sompRestoreObject**, **sompRestoreObjectFromIdString, sompRestoreObjectWithoutChildren**, **sompObjectExists**, **sompDeleteObject**

sompAddObjectToWriteSet Method

Purpose

Adds the specified persistent object to the set of objects to be written by the **sompStoreObject** method as part of this write request.

IDL Syntax

void sompAddObjectToWriteSet (
in SOMPPersistentObject *thisObject***);**

Description

The **sompAddObjectToWriteSet** method adds the given persistent object to the set of objects being written by the **sompStoreObject** method as part of this write request. An error occurs if this method is invoked before **sompStoreObject** is called.

This method provides a way for an object's Encoder/Decoder to add a persistent object to the set to be written. This is useful for saving persistent child objects that were not *registered* with their parent object.

The **sompAddObjectToWriteSet** method may raise a **sompException** exception. This method is typically used by implementors of Encoders/Decoders.

Parameters

receiver	A pointer to an object of class **SOMPPersistentStorageMgr.**
ev	A pointer to the **Environment** structure for the caller.
thisObject	A pointer to an object of class **SOMPPersistentObject.**

Return Value

None.

Example

```
SOMPPersistentObject thisObject;
Environment *ev;
SOMPPersistentStorageMgr psm;
/* ... */
_sompAddObjectToWriteSet(psm, ev, thisObject);
```

Original Class

SOMPPersistentStorageMgr

Related Information

Methods: sompStoreObject, **sompStoreObjectWithoutChildren**, **sompRestoreObject**, **sompRestoreObjectFromIdString**, **sompRestoreObjectWithoutChildren**, **sompObjectExists**, **sompDeleteObject, sompAddIdToReadSet**

sompDeleteObject Method

Purpose

Deletes a specified persistent object from persistent storage.

IDL Syntax

void sompDeleteObject (
in SOMPPersistentId *objectID***);**

Description

The **sompDeleteObject** method deletes the specified persistent object from persistent storage.

If the object is an in-memory object only, it is removed from the Name Space Manager and its in-memory I/O Group object and is then freed. If the object also previously existed in a stored I/O Group, then the stored I/O Group is modified to indicate that the object has been removed.

This method may raise a **sompException** exception. This method is typically used by client programs.

Parameters

receiver	A pointer to an object of class **SOMPPersistentStorageMgr.**
ev	A pointer to the **Environment** structure for the caller.
objectID	A pointer to the ID of the object to be deleted.

Return Value

None.

Example

```
SOMPPersistentStorageMgr psm = SOMPPersistentStorageMgrNew();
SOMPPersistentId pid = SOMPPersistentIdNew();
Environment *ev;

_somutSetIdString(pid, ev, ”SOMPAscii:/mydogs/dog.dat:0”);
_sompDeleteObject(psm, ev, pid);
_somFree(pid);
```

Original Class

SOMPPersistentStorageMgr

Related Information

Methods: sompStoreObject, **sompStoreObjectWithoutChildren**, **sompRestoreObject**, **sompRestoreObjectFromIdString**, **sompRestoreObjectWithoutChildren**, **sompObjectExists**

sompObjectExists Method

Purpose

Determines whether a persistent object exists in persistent storage.

IDL Syntax

boolean sompObjectExists (
in SOMPPersistentId *objectID***);**

Description

The **sompObjectExists** method checks to see if the specified persistent object exists in persistent storage. If so, it returns TRUE; otherwise, it returns FALSE. This method is typically used by client programs.

Parameters

receiver	A pointer to an object of class **SOMPPersistentStorageMgr**.
ev	A pointer to the **Environment** structure for the caller.
objectID	A pointer to the ID of the object whose existence is questioned.

Return Value

The **sompObjectExists** method returns TRUE if the object exists in the persistent store; otherwise, it returns FALSE. If an I/O error occurs, the method raises a **sompException** exception.

Example

```
SOMPPersistentStorageMgr psm = SOMPPersistentStorageMgrNew();
SOMPPersistentId pid = SOMPPersistentIdNew();

_somutSetIdString(pid, ev, "SOMPAscii:/mydogs/dog.dat:0");
if (_sompObjectExists(psm, ev, pid)) {
   somPrintf("Oth object exists\n");
} else {
   somPrintf("Oth object does not exist\n");
}
 _somFree(pid);
```

Original Class

SOMPPersistentStorageMgr

Related Information

Methods: sompStoreObject, **sompStoreObjectWithoutChildren**, **sompRestoreObject**, **sompRestoreObjectFromIdString**, **sompRestoreObjectWithoutChildren**, **sompDeleteObject**

sompRestoreObject Method

Purpose

Restores a persistent object and all of its contained persistent objects to a stable state.

IDL Syntax

SOMObject sompRestoreObject (
in SOMPPersistentId *objectID***);**

Description

The **sompRestoreObject** method restores a persistent object and all of its contained persistent objects to a stable state. This method may raise a **sompException** exception. This method is typically used by client programs.

Parameters

receiver	A pointer to an object of class **SOMPPersistentStorageMgr**.
ev	A pointer to the **Environment** structure for the caller.
objectID	A pointer to the ID of the object to be restored**.**

Return Value

The **sompRestoreObject** returns a pointer to the restored object. If there is an error, this method returns NULL and raises a **sompException** exception.

Example

```
SOMPPersistentStorageMgr psm = SOMPPersistentStorageMgrNew();
SOMPPersistentId pid = SOMPPersistentIdNew();
phoneDir mylist;

_somutSetIdString(pid, ev,
     ”SOMPAscii:/u/roger/data/myPhoneBook.dat:0”);
checkError(ev);
mylist = _sompRestoreObject (psm, ev, pid);
```

Original Class

SOMPPersistentStorageMgr

Related Information

Methods: sompStoreObject, **sompRestoreObjectFromIdString,**
sompStoreObjectWithoutChildren, **sompRestoreObjectWithoutChildren**,
sompObjectExists, **sompDeleteObject**

sompRestoreObjectFromIdString Method

Purpose

Restores a persistent object and all its persistent children to a stable state, given the object's string ID.

IDL Syntax

SOMObject sompRestoreObjectFromIdString (
in string *idString***);**

Description

The **sompRestoreObjectFromIdString** method instantiates a persistent ID from the given string, and then restores the persistent object and all its children to a stable state. This method may raise a **sompException** exception. This method is typically used by client programs.

Parameters

receiver	A pointer to an object of class **SOMPPersistentStorageMgr**.
ev	A pointer to the **Environment** structure for the caller.
idString	A string representing the ID of the object to be restored.

Return Value

The **sompRestoreObjectFromIdString** returns a pointer to the restored object. If there is an error, this method returns NULL and raises a **sompException** exception.

Example

```
SOMPPersistentStorageMgr psm;
animal pooh = animalNew();
Environment *ev;
/* ... */
pooh = _sompRestoreObjectFromIdString
   (psm, ev, ”SOMPAscii:./testio/test36:0”);
```

Original Class

SOMPPersistentStorageMgr

Related Information

Methods: sompStoreObject, **sompRestoreObject, sompStoreObjectWithoutChildren**, **sompRestoreObjectWithoutChildren**, **sompObjectExists**, **sompDeleteObject**

somp RestoreObjectWithoutChildren Method

Purpose

Restores a persistent object to a stable state. Contained objects are restored only in an unstable state.

IDL Syntax

SOMObject sompRestoreObjectWithoutChildren (
in SOMPPersistentId *objectID***);**

Description

The **sompRestoreObjectWithoutChildren** method restores a persistent object to a stable state. Any contained objects are only restored in an unstable state. If an error occurs, NULL is returned and a **sompException** exception is raised. This method is typically used by client programs.

Parameters

receiver	A pointer to an object of class **SOMPPersistentStorageMgr**.
ev	A pointer to the **Environment** structure for the caller.
objectID	A pointer to the ID of the object to be restored**.**

Return Value

The **sompRestoreObjectWithoutChildren** returns a pointer to the restored object. If there is an error, this method returns NULL and raises a **sompException** exception.

Example

```
SOMPPersistentStorageMgr psm = SOMPPersistentStorageMgrNew();
SOMPPersistentId pid = SOMPPersistentIdNew();
phoneDir mylist;
company myCompany;
person myPerson;

_somutSetIdString(pid, ev,
                    ”SOMPAscii:/u/roger/data/myPhoneBook:0”);
checkError(ev);
mylist = _sompRestoreObjectWithoutChildren (psm, ev, pid);
checkError(ev);
```

Original Class

SOMPPersistentStorageMgr

Related Information

Methods: sompStoreObject, **sompStoreObjectWithoutChildren**, **sompRestoreObject**, **sompRestoreObjectFromIdString, sompObjectExists**, **sompDeleteObject**

sompStoreObject Method

Purpose

Stores a persistent object and all its persistent children.

IDL Syntax

void sompStoreObject (
in SOMPPersistentObject *thisObject***);**

Description

The **sompStoreObject** method stores a persistent object and all its persistent children. This method may raise a **sompException** exception. This method is typically used by client programs.

Parameters

receiver	A pointer to an object of class **SOMPPersistentStorageMgr**.
ev	A pointer to the **Environment** structure for the caller.
thisObject	A pointer to the object to be stored.

Return Value

None.

Example

```
SOMPPersistentId pid = SOMPPersistentIdNew();
phoneDir mylist = phoneDirNew();
char idBuff[SOMPMAXIDSIZE];

strcpy(idBuff, "SOMPAscii:./pdata/phoneDir:0");
_somutSetIdString(pid, ev, idBuff);

/* ... */
_sompInitGivenId (mylist, ev, pid);
checkError(ev);
_somFree(pid);
_sompStoreObject (psm, ev, mylist);
```

Original Class

SOMPPersistentStorageMgr

Related Information

Methods: sompStoreObjectWithoutChildren, **sompRestoreObject**, **sompRestoreObjectFromIdString, sompRestoreObjectWithoutChildren**, **sompObjectExists**, **sompDeleteObject**

sompStoreObjectWithoutChildren Method

Purpose

Stores a persistent object, but does not store contained objects..

IDL Syntax

void sompStoreObjectWithoutChildren (
in SOMPPersistentObject *thisObject***);**

Description

The **sompStoreObjectWithoutChildren** method stores a persistent object, but does not store any contained objects. If an error occurs, the method raises a **sompException** exception. This method is typically used by client programs.

Parameters

receiver	A pointer to an object of class **SOMPPersistentStorageMgr**.
ev	A pointer to the **Environment** structure for the caller.
thisObject	A pointer to the object to be stored.

Return Value

None.

Example

```
SOMPPersistentId pid = SOMPPersistentIdNew();
phoneDir mylist = phoneDirNew();
char idBuff[SOMPMAXIDSIZE];

strcpy(idBuff, ”SOMPAscii:./pdata/phoneDir:0”);
_somutSetIdString(pid, ev, idBuff);

/* ... */
_sompInitGivenId (mylist, ev, pid);
checkError(ev);
_somFree(pid);
_sompStoreObjectWithoutChildren (psm, ev, mylist);
```

Original Class

SOMPPersistentStorageMgr

Related Information

Methods: sompStoreObject, **sompRestoreObject**,
sompObjectExists, **sompDeleteObject, sompRestoreObjectWithoutChildren**

SOMUTId Class

Description

The **SOMUTId** class is the base class for an ID. Instances of classes derived from **SOMUTId** represent a value that can be used to identify an object, a file, a network node, etc.

File Stem

somida

Base Class

SOMObject

Metaclass

SOMClass

Ancestor Classes

SOMObject

New Methods

somutSetIdId
somutEqualsId
somutCompareId
somutHashId

Overriding Methods

None.

somutCompareId Method

Purpose

Performs an ordered comparison of one ID with another.

IDL Syntax

short somutCompareId (
in SOMUTId *otherId***);**

Description

The **somutCompareId** method performs an ordered comparison of the *receiver* ID against *otherId.* If *receiver* is less than *otherId*, –1 is returned. If the IDs are equivalent, 0 is returned. If *receiver* is greater than *otherId*, 1 is returned. This method is typically used by implementors of I/O Group Managers.

Parameters

receiver	A pointer to an object of class **SOMUTId**.
ev	A pointer to the **Environment** structure for the caller.
otherId	A pointer to an object of class **SOMUTId**.

Return Value

If *receiver* is less than *otherId*, –1 is returned. If the IDs are equivalent, 0 is returned. If *receiver* is greater than *otherId*, 1 is returned.

Example

```
SOMPPersistentId id1;
SOMPPersistentId id2;
Environment *ev;
/* ... */
if (!_somutCompareId(id1, ev, id2))
    printf("Ids are the same\n");
```

Original Class

SOMUTId

Related Information

Methods: somutSetIdId, somutEqualsId, somutHashId

somutEqualsId Method

Purpose

Determines whether an ID is equal to the given ID.

IDL Syntax

boolean somutEqualsId (
in SOMUTId *otherId***);**

Description

The **somutEqualsId** method determines whether the *receiver* ID is equal to *otherId.* This method is typically used by implementors of I/O Group Managers.

Parameters

receiver	A pointer to an object of class **SOMUTId**.
ev	A pointer to the **Environment** structure for the caller.
otherId	A pointer to an object of class **SOMUTId**.

Return Value

The **somutEqualsId** method returns TRUE if the given IDs are equivalent, and FALSE otherwise.

Example

```
SOMPPersistentId id1;
SOMPPersistentId id2;
Environment *ev;
/* ... */
if (_somutCompareId(id1, ev, id2))
    printf("Ids are the same\n");
```

Original Class

SOMUTId

Related Information

Methods: somutSetIdId, somutCompareId, somutHashId

somutHashId Method

Purpose

Determines the hash value equivalent to a given ID.

IDL Syntax

unsigned long somutHashId ( );

Description

The **somutHashId** method computes the hash value of a given SOM ID. This method is typically overridden by implementors of persistent classes.

Parameters

receiver	A pointer to an object of class **SOMUTId**.
ev	A pointer to the **Environment** structure for the caller.

Return Value

The **somutHashId** method returns an unsigned long that is the equivalent hash value of the given ID.

Original Class

SOMUTId

Related Information

Methods: somutSetIdId, somutEqualsId, somutCompareId

somutSetIdId Method

Purpose

Sets an ID equal to a given ID.

IDL Syntax

void somutSetIdId (
in SOMUTId *otherId***);**

Description

The **somutSetIdId** method sets the receiver ID to be equal to *otherId.* This method is typically used by client programs.

Parameters

receiver	A pointer to an object of class **SOMUTId**, representing the ID to set.
ev	A pointer to the **Environment** structure for the caller.
otherId	A pointer to an object of class **SOMUTId**, representing the new value.

Return Value

None.

Example

```
SOMPPersistentId id1, id2;
Environment *ev;
/* ... */
_somutSetIdId(id1, ev, _sompGetPersistentId(id2, ev));
```

Original Class

SOMUTId

Related Information

Methods: somutEqualsId, somutCompareId, somutHashId

SOMUTStringId Class

Description

The **SOMUTStringId** class is the base class for an ID represented by an ASCII Z string (an ASCII string terminated with a zero). Instances of classes derived from **SOMUTStringId** represent a string value that can be used to identify an object, file, network node, etc.

File Stem

somsid

Base

SOMUTId

Metaclass

SOMClass

Ancestor Classes

SOMUTId, SOMObject

New Methods

somutSetIdString
somutGetIdString
somutGetIdStringLen
somutEqualsString
somutCompareString

Overriding Methods

somutSetIdId
somutEqualsId
somutCompareId
somutHashId

somutCompareString Method

Purpose

Performs an ordered comparison of an ID with a string.

IDL Syntax

short somutCompareString (
in string *IdString***);**

Description

The **somutCompareString** method performs an ordered comparison of the *receiver* ID against *IdString.* If *receiver* is less than *IdString*, –1 is returned. If *receiver* is equivalent to *IdString*, 0 is returned. If *receiver* is greater than *IdString*, 1 is returned. This method is typically used by client programs.

Parameters

receiver	A pointer to an object of class **SOMUTStringId** to compare.
ev	A pointer to the **Environment** structure for the caller.
IdString	A string with which to compare the *receiver.*

Return Value

If *receiver* is less than *IdString*, –1 is returned. If *receiver* is equivalent to *IdString*, 0 is returned. If *receiver* is greater than *IdString*, 1 is returned.

Example

```
SOMPPersistentId id;
Environment *ev;
/* ... */
if (!_somutCompareString(id, ev, ”SOMPAscii:/u/roger/dog:0”)
   /* ... */
```

Original Class

SOMUTStringId

Related Information

Methods: somutSetIdString, somutEqualsString, somutGetIdString, somutGetIdStringLen

somutEqualsString Method

Purpose

Determines whether an ID is equal to the given string.

IDL Syntax

boolean somutEqualsString (
in string *idString***);**

Description

The **somutEqualsString** method determines whether the *receiver* ID is equal to *idString.* This method is typically used by client programs.

Parameters

receiver	A pointer to an object of class **SOMUTStringId**.
ev	A pointer to the **Environment** structure for the caller.
idString	A string to be compared with *receiver*'s ID.

Return Value

The **somutEqualsString** method returns TRUE if the *receiver* is equivalent to *idString*, and FALSE otherwise.

Example

```
SOMPPersistentId id;
Environment *ev;
/* ... */
if (_somutEqualsString(id, ev, ”SOMPAscii:/u/roger/dog:0”)
   /* ... */
```

Original Class

SOMUTStringId

Related Information

Methods: somutSetIdString, somutGetIdString, somutGetIdStringLen, somutCompareString

somutGetIdString Method

Purpose

Converts an ID to a string.

IDL Syntax

string somutGetIdString (
in string *toBuffer***);**

Description

The **somutGetIdString** method stores *receiver* as a string into *toBuffer* and returns *toBuffer*. The **somutGetIdStringLen** method can be used to determine the appropriate size for *toBuffer*. This method is typically used by client programs.

Parameters

receiver	A pointer to an object of class **SOMUTStringId**, representing the ID to convert.
ev	A pointer to the **Environment** structure for the caller.
toBuffer	The buffer into which the string value of *receiver* will be stored.

Return Value

The **somutGetIdString** method returns the *toBuffer* string.

Example

```
Environment *ev
char buffer[SOMPMAXIDSIZE];
SOMPPersistentId pid;
/* ... */
printf(”ID: %s\n”, _somutGetIdString(pid, ev, buffer));
```

Original Class

SOMUTStringId

Related Information

Methods: somutSetIdString, somutGetIdStringLen, somutEqualsString, somutCompareString

somutGetIdStringLen Method

Purpose

Returns the length of an ID string.

IDL Syntax

long somutGetIdStringLen ( );

Description

The **somutGetIdStringLen** method returns the length of the string ID represented by *receiver.* The **somutGetIdStringLen** method can be used to determine the appropriate size buffer to pass to the **somutGetIdString** method. This method is typically used by client programs.

Parameters

receiver A pointer to an object of class **SOMUTStringId**, representing the ID whose string is required.

ev A pointer to the **Environment** structure for the caller.

Return Value

The **somutGetIdStringLen** method returns the length of the string ID represented by *receiver.*

Example

```
Environment *ev
string buffer;
long len;
SOMPPersistentId pid;
/* ... */
len = _somutGetIdStringLen(pid);
buffer = SOMMalloc(len+1);
printf(”ID: %s\n”, _somutGetIdString(pid, ev, buffer));
```

Original Class

SOMUTStringId

Related Information

Methods: somutSetIdString, somutGetIdString, somutEqualsString, somutCompareString

somutSetIdString Method

Purpose

Sets an ID equal to a given string.

IDL Syntax

long somutSetIdString (
in string *IdString***);**

Description

The **somutSetIdString** method sets the *receiver* ID to be equal to *IdString.* If *IdString* is not syntactically correct, a nonzero error code is returned. This method is typically used by client programs.

Parameters

receiver	A pointer to an object of class **SOMUTStringId**, representing the ID to set.
ev	A pointer to the **Environment** structure for the caller.
IdString	A string representing the new value for *receiver.*

Return Value

If *IdString* is not syntactically correct, a nonzero error code is returned. Otherwise, zero is returned.

Example

```
Environment *ev
SOMPPersistentId pid;
/* ... */
if(_somutSetIdString(pid, ev, "SOMPAscii:/u/roger/dog:0"));
```

Original Class

SOMUTStringId

Related Information

Methods: somutGetIdString, somutGetIdStringLen, somutEqualsString, somutCompareString