TAppServerStub

Hierarchy Properties Methods See also

TAppServerStub is the stub class for a CORBA data module.

Unit

corbastd

Description

The CORBA Data Module Wizard adds code to define and register a descendant of TAppServerStub that can marshal code to a CORBA data module.

Applications do not instantiate stub objects directly. Instead, the CORBA connection component creates a stub object for the data module and assigns its interface to the AppServer property.

TAppServerStub properties

Derived from TInterfacedObject

- RefCount

TAppServerStub properties

TAppServerStub By object Legend

- RefCount

TAppServerStub methods

TAppServerStub Alphabetically

In TAppServerStub

- AS_ApplyUpdates
- AS_DataRequest
- AS_Execute
- AS_GetParams
- AS_GetProviderNames
- AS_GetRecords
- AS_RowRequest

Derived from TCorbaStub

- Create
- Destroy

Derived from TInterfacedObject

- BeforeDestruction

Derived from TObject

- AfterConstruction
- ClassInfo
- ClassName
- ClassNameIs
- ClassParent
- ClassType
- CleanupInstance
- DefaultHandler
- Dispatch
- FieldAddress
- Free
- FreeInstance
- GetInterface
- GetInterfaceEntry
- GetInterfaceTable
- InheritsFrom
- InitInstance
- InstanceSize
- MethodAddress
- MethodName
- NewInstance
- SafeCallException

TAppServerStub methods

TAppServerStub By object

- AfterConstruction
- AS_ApplyUpdates
- AS_DataRequest
- AS_Execute
- AS_GetParams
- AS_GetProviderNames
- AS_GetRecords
- AS_RowRequest
- BeforeDestruction
- ClassInfo
- ClassName
- ClassNameIs
- ClassParent
- ClassType
- CleanupInstance
- Create
- DefaultHandler
- Destroy
- Dispatch
- FieldAddress
- Free
- FreeInstance
- GetInterface
- GetInterfaceEntry
- GetInterfaceTable
- InheritsFrom
- InitInstance
- InstanceSize
- MethodAddress
- MethodName
- NewInstance
- SafeCallException

TAppServerStub.AS_ApplyUpdates

TAppServerStub See also

Applies updates received from a client dataset using a specified provider.

```
function AS_ApplyUpdates (const ProviderName: (*VT_8:0*)WideString; Delta:
  (*VT_12:0*)OleVariant; MaxErrors: (*VT_3:0*)Integer; out ErrorCount:
  (*VT_3:0*)Integer; var OwnerData: (*VT_12:0*)OleVariant):
  (*VT_12:0*)OleVariant; safecall;
```

Description

AS_ApplyUpdates marshals IAppserver's AS_Updates method.

ProviderName specifies the name of the provider component that manages the update operation. Delta is a Variant containing the updated, inserted, and deleted records to write to the database. MaxErrors indicates the maximum number of errors to permit before stopping the update operation; ErrorCount indicates the actual number of errors encountered during the update operation. OwnerData is information supplied by the client application in a BeforeApplyUpdates event handler and returned to the client application in an AfterApplyUpdates event handler. It is passed to the provider's BeforeApplyUpdates event handler and returned from the provider's AfterApplyUpdates event handler.

AS_ApplyUpdates returns a Variant that is a data packet containing all records that could not be applied to the database.

TAppServerStub.AS_DataRequest

TAppServerStub See also

Generates an OnDataRequest event on the specified provider component.

```
function AS_DataRequest (const ProviderName: (*VT_8:0*)WideString; Data:
  (*VT_12:0*)OleVariant): (*VT_12:0*)OleVariant; safecall;
```

Description

AS_DataRequest marshals IAppServer's AS_DataRequest method. This method allows application developers to customize the communication between a client application and a provider component. There is no predefined meaning for the Data parameter or the return value.

TAppServerStub.AS_Execute

TAppServerStub See also

Executes the query or stored procedure bound to a specified provider.

```
procedure AS_Execute (const ProviderName: (*VT_8:0*)WideString; const
  CommandText: (*VT_8:0*)WideString; var Params: (*VT_12:0*)OleVariant;  var
  OwnerData: (*VT_12:0*)OleVariant); safecall;
```

Description

AS_Execute marshals IAppserver's AS_Execute method. AS_Execute calls the Execute method of the specified provider after assigning any parameters. Output parameters that result from executing a query or stored procedure are returned in the Params parameter.

The ProviderName parameter specifies the name of the provider component associated with the query or stored procedure.

The CommandText parameter includes any SQL statement that overrides the query or stored procedure that would otherwise be executed. This statement is executed only if the specified provider includes poAllowCommandText in its Options property.

The Params parameter encodes any parameters expected by the query or stored procedure and returns any output parameters.

The OwnerData parameter contains custom information that appears as an argument to the provider's BeforeExecute and AfterExecute event handlers. This information originates in a client dataset's BeforeExecute event handler. The value returned in OwnerData is passed to the client dataset's AfterExecute event handler.

TAppServerStub.AS_GetParams

TAppServerStub See also

Fetches current parameter values from the dataset bound to a specified provider.

function **AS_GetParams (**const **ProviderName: (*VT_8:0*)WideString;** var **OwnerData: (*VT_12:0*)OleVariant): (*VT_12:0*)OleVariant;** safecall**;Description**

AS_GetParams marshals IAppServer's AS_GetParams method.

The ProviderName parameter specifies the name of the provider component whose dataset has the parameters. The OwnerData represents custom information that originates in a client dataset's BeforeGetParams event handler and returns information that is passed tot he client dataset's AfterGetParams event handler.

AS_GetParams returns the parameters, encoded as a Variant array. If the dataset has no parameters, or if the provider does not support parameter fetching, AS_GetParams returns a Null Variant.

AS_GetParams should not be used to retrieve output parameters from stateless application servers, because parameter values may be changed by other applications. When writing a stateless application server, obtain output parameters using the AS_Execute method handler instead.

TAppServerStub.AS_GetProviderNames

TAppServerStub See also

Returns a list of all the available providers on the remote data module.

```
function AS_GetProviderNames : (*VT_12:0*)OleVariant; safecall;
```

Description

AS_GetProviderNames marshals IAppServer's AS_GetProviderNames method. It returns a list of all providers available from a remote data module. These values indicate the possible values for the ProviderName property of a client dataset that connects to the remote data module. When calling any of the AS_XXXX methods, pass one of these names to indicate the provider that is the target of the method call.

AS_GetProviderNames returns the names of the providers in a Variant array. Client datasets are associated with a provider by setting their ProviderName property to one of these names.

Warning: AS_GetProviderNames returns a list that can include providers that are not exported. If a provider in the list has its Exported property set to False, trying to use that provider over an IAppServer interface results in an exception.

TAppServerStub.AS_GetRecords

TAppServerStub See also

Returns a data packet that contains the specified data.

function **AS_GetRecords (**const **ProviderName: (*VT_8:0*)WideString; Count: (*VT_3:0*)Integer;** out **RecsOut: (*VT_3:0*)Integer; Options: (*VT_3:0*)Integer;** const **CommandText: (*VT_8:0*)WideString;** var **Params: (*VT_12:0*)OleVariant;** var **OwnerData: (*VT_12:0*)OleVariant): (*VT_12:0*)OleVariant;** safecall**;Description**

AS_GetRecords marshals IAppServer's AS_GetRecords method. It returns the requested records, starting with the current record of the provider's dataset. When working with a stateless remote data module, you may need to reposition the cursor or re-execute a query or stored procedure in the provider's BeforeGetRecords event handler.

ProviderName gives the name of the provider component that provides the records.

Count indicates the number or type of records to retrieve. If Count is -1, all records are retrieved. If Count is 0, only metadata is retrieved. If Count is greater than 0, only Count records are retrieved.

RecsOut returns the actual number of records retrieved.

Options indicates what information should be added to the data packet in addition to data. It is an integer version of the TGetRecordOptions type. To convert the TGetRecordOptions value to the appropriate integer, use the Ord function. For example:

```
Ord(grMetaData) + Ord(grXML)
```

CommandText is an SQL statement that overrides the provider's default method for obtaining data. The provider's dataset generates its records using CommandText instead of its default mechanism. This allows clients to override the SQL statement of a query or stored procedure on the server. CommandText is only used if the provider's options include poAllowCommandText.

Params is a Variant containing any parameters that should be passed to the provider's dataset before it executes to generate the requested data. It returns any output parameters.

OwnerData contains custom information that is supplied by a client dataset's BeforeGetRecords event handler. This information is passed to the provider's BeforeGetRecords event handler. OwnerData returns information supplied by the provider's AfterGetRecords event handler.

Records are returned as a data packet in a Variant.

TAppServerStub.AS_RowRequest

TAppServerStub See also

Returns information from a specified record of the provider's dataset.

function **AS_RowRequest (**const **ProviderName: (*VT_8:0*)WideString; Row: (*VT_12:0*)OleVariant; RequestType: (*VT_3:0*)Integer;** var **OwnerData: (*VT_12:0*)OleVariant): (*VT_12:0*)OleVariant;** safecall**;Description**

AS_RowRequest marshals IAppServer's AS_RowRequest method.

The ProviderName parameter indicates the provider associated with the dataset from which information should be fetched.

The Row parameter is an OleVariant that describes the current record on the client dataset.

The RequestType parameter indicates the type of information required. It is an integer version of the TFetchOptions type. (An integer because the value must be compatible with a COM interface). To convert the TFetchOptions value to the appropriate integer, use the Ord function. For example:

```
Ord(foBlobs) + Ord(foDetails);
```

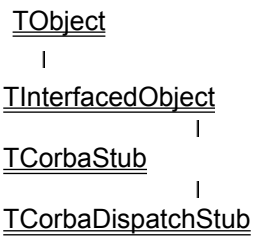
OwnerData contains custom information that is supplied by a client dataset's BeforeRowRequest event handler. This information is passed to the provider's BeforeRowRequest event handler. OwnerData returns custom information supplied by the provider's AfterRowRequest event handler.

The requested data is returned as a delta packet.

Accessibility

- Read-only

Hierarchy


TObject
TInterfacedObject
TCorbaStub
TCorbaDispatchStub

TAppServerSkeleton

Hierarchy Properties Methods See also

TAppServerSkeleton is the skeleton class for a CORBA data module.

Unit

corbastd

Description

The CORBA Data Module Wizard adds code to define and register a descendant of TAppServerSkeleton. The data broker skeleton receives incoming interface calls, unmarshals parameters, forwards the call to a CORBA data module, and then marshals any return values.

Applications do not instantiate skeleton objects directly. A CORBA factory object creates the skeleton when it is needed.

TAppServerSkeleton properties

Derived from TInterfacedObject

- RefCount

TAppServerSkeleton properties

TAppServerSkeleton By object Legend

- RefCount

TAppServerSkeleton methods

TAppServerSkeleton Alphabetically Legend

In TAppServerSkeleton

- Create
- GetImplementation

Derived from TCorbaSkeleton

- Destroy

Derived from TInterfacedObject

- BeforeDestruction

Derived from TObject

- AfterConstruction
- ClassInfo
- ClassName
- ClassNameIs
- ClassParent
- ClassType
- CleanupInstance
- DefaultHandler
- Dispatch
- FieldAddress
- Free
- FreeInstance
- GetInterface
- GetInterfaceEntry
- GetInterfaceTable
- InheritsFrom
- InitInstance
- InstanceSize
- MethodAddress
- MethodName
- NewInstance
- SafeCallException

TAppServerSkeleton methods

TAppServerSkeleton By object Legend

- AfterConstruction
- BeforeDestruction
- ClassInfo
- ClassName
- ClassNameIs
- ClassParent
- ClassType
- CleanupInstance
- Create
- DefaultHandler
- Destroy
- Dispatch
- FieldAddress
- Free
- FreeInstance
- GetImplementation
- GetInterface
- GetInterfaceEntry
- GetInterfaceTable
- InheritsFrom
- InitInstance
- InstanceSize
- MethodAddress
- MethodName
- NewInstance
- SafeCallException

TAppServerSkeleton.Create

TAppServerSkeleton See also

Creates an instance of TAppServerSkeleton to represent a CORBA data module.

```
constructor Create(const InstanceName: string; const Impl: IUnknown);
  override;
```

Description

Do not directly instantiate TAppServerSkeleton objects. They are created by the associated CORBA factory.

The InstanceName parameter specifies an optional name for the CORBA skeleton instance. The Impl parameter is the interface of an instance of a TCorbaDataModule descendant.

The data broker skeleton passes interface calls on to the Impl interface after unmarshaling parameters and then marshals any return values.

TAppServerSkeleton.GetImplementation

TAppServerSkeleton See also

Returns an interface for the associated CORBA data module.

```
type IObject = System.IUnknown;
procedure GetImplementation(out Impl: IObject); override; stdcall;
```

Description

Use GetImplementation to obtain an interface for the CORBA data module instance that actually handles interface calls marshaled by this TAppServerSkeleton.

Accessibility

- Read-only

Scope

Published

Hierarchy

TObject
|
TInterfacedObject
|
TCorbaSkeleton

TBOA

Hierarchy Methods See also

TBOA represents the CORBA Basic Object Adaptor.

Unit

corbaobj

Description

TBOA is the type of the global BOA variable. Applications do not instantiate TBOA objects. Instead, they use the global BOA variable to communicate with CORBA using the Basic Object Adaptor.

TBOA introduces methods to indicate when clients can call the CORBA server application and to fetch principal data sent to the server from client applications.

TBOA methods

TBOA Alphabetically

In TBOA

- Deactivate
- GetPrincipal
- ImplIsReady
- Initialize
- ObjIsReady

Derived from TObject

- AfterConstruction
- BeforeDestruction
- ClassInfo
- ClassName
- ClassNameIs
- ClassParent
- ClassType
- CleanupInstance
- Create
- DefaultHandler
- Destroy
- Dispatch
- FieldAddress
- Free
- FreeInstance
- GetInterface
- GetInterfaceEntry
- GetInterfaceTable
- InheritsFrom
- InitInstance
- InstanceSize
- MethodAddress
- MethodName
- NewInstance
- SafeCallException

TBOA methods

- AfterConstruction
- BeforeDestruction
- ClassInfo
- ClassName
- ClassNameIs
- ClassParent
- ClassType
- CleanupInstance
- Create
- Deactivate
- DefaultHandler
- Destroy
- Dispatch
- FieldAddress
- Free
- FreeInstance
- GetInterface
- GetInterfaceEntry
- GetInterfaceTable
- GetPrincipal
- ImplIsReady
- InheritsFrom
- Initialize
- InitInstance
- InstanceSize
- MethodAddress
- MethodName
- NewInstance
- ObjIsReady
- SafeCallException

TBOA.Deactivate

TBOA See also

Makes an object instance unavailable to CORBA clients.

```
procedure Deactivate(const Obj: IObject);
```

Description

Call Deactivate to deactivate an object that was made available using the ObjIsReady method. Calling Deactivate removes the object from the Smart Agent's list of objects offered by the application.

The Obj parameter is the interface of the skeleton object that is deactivated.

TBOA.GetPrincipal

TBOA See also

Returns a TCorbaPrincipal value for an object.

```
type TCorbaPrincipal = array of Byte;
function GetPrincipal(const Obj: IObject): TCorbaPrincipal;
```

Description

A TCorbaPrincipal represents information about client applications that are making operation requests on an object implementation. Client applications set this value using the CORBA stub's SetPrincipal method. Use GetPrincipal in the server application to read this information.

The Obj parameter is the interface of the CORBA server object to which the Principal was written. This is the object represented by the stub object whose SetPrincipal method wrote the information.

TBOA.ImplIsReady

TBOA See also

Allows the server application to begin receiving messages.

```
procedure ImplIsReady;
```

Description

Call ImplIsReady when the CORBA console application is ready to receive messages from clients. When a CORBA server application starts up, it instantiates the objects that receive messages from clients, calling ObjIsReady for each object when it is instantiated. When the BOA is informed about all objects that are instantiated at startup, console applications must call ImplIsReady to allow the application to receive messages.

Windows applications should not call ImplIsReady because the windows message loop serves the same purpose.

TBOA.Initialize

TBOA

Sends startup options to the CORBA Basic Object Adaptor.

```
type
  TArgv = array of string;
  TCommandLine = TArgv;

class procedure Initialize(const CommandLine: TCommandLine);
```

Description

CORBA server applications call Initialize to indicate options such as the desired thread policy or the TCP/IP port number to be used. Appropriate values are sent automatically when the CORBA application is created using a CORBA Wizard.

To create a set of options to send to the BOA, treat the CommandLine parameter like a dynamic array. Use code such as the following to set options:

```
SetLength(CommandLine, 2);
CommandLine[0] := '-OAconnectionMax 100';
CommandLine[1] := '-OAid TSession';
```

The following table summarizes the possible options:

Option	Purpose
OAConnectionMax	The maximum number of connections (only applies when -OAid TSession is selected.)
OAconnectionMaxIdle	The time (in seconds) which a connection can be idle without any trafic. Connections that idle beyond this time are shut down by VisiBroker.
OAid	The thread policy for multithreaded servers. This value is TSession (where each client has a dedicated thread), or TPool (where a single client can use multiple threads and clients can reuse threads after other clients have finished with them.)
OAipAddr	The host name or IP address of the BOA. Use this option if the server machine has multiple network interfaces and the BOA is associated with only one.
OALocalIPC	The inter-process communication method. When this value is 1, clients on the same host as the server application use the same shared memory space. When this value is 0, sockets are always used.
OAPort	The port number which the BOA uses to listen for new connections.
OArcvbufsize	The size of the buffer (in bytes) used to receive messages.
OAsendbufsize	The size of the buffer (in bytes) used to send messages.
OAtcpNoDelay	The policy on sending requests. If this value is 1, sockets immediately send requests. If this value is 0, sockets send requests in batches as buffers fill.
OAThreadMax	The maximum number of threads. (only applies when -OAid TPool is selected.)
OAThreadMaxIdle	The time (in seconds) which a thread can exist without servicing any requests.
OAThreadStackSize	The maximum thread stack size (in bytes). (only applies when -OAid TPool is selected.)

TBOA.ObjIsReady

TBOA See also

Informs the Basic Object Adaptor when an object is instantiated.

```
procedure ObjIsReady(const Obj: IObject);
```

Description

When a CORBA server application starts up, it instantiates the objects that receive messages from clients, calling ObjIsReady after each object is instantiated.

The Obj parameter is an interface to the skeleton class for the object that receives client messages.

Note: In CORBA applications created by the CORBA Wizards, ObjIsReady is called automatically for the CORBA factory objects.

Hierarchy

TObject

TCorbaComObjectFactory

Hierarchy Properties Methods See also

TCorbaComObjectFactory creates a COM object in response to a request from a CORBA client.

Unit

comcorba

Description

When a developer right-clicks an Automation server and selects Expose As Corba Object, a TCorbaComObjectFactory is automatically added to the server application. This acts like a TCorbaObjectFactory object, except that it creates an instance of a COM object as the CORBA implementation object.

By using a CORBA COM object Factory, the server can respond to requests by CORBA clients as well as COM clients.

TCorbaComObjectFactory properties

TCorbaComObjectFactory Alphabetically Legend

In TCorbaComObjectFactory

- ImplementationClass

Derived from TCorbaFactory

- InstanceName
  - Instancing
  - InterfaceName
  - RepositoryID
  - ThreadModel

Derived from TInterfacedObject

- RefCount

TCorbaComObjectFactory properties

- ImplementationClass
  - InstanceName
  - Instancing
  - InterfaceName
  - RefCount
  - RepositoryID
  - ThreadModel

TCorbaComObjectFactory.ImplementationClass

TCorbaComObjectFactory Methods See also

Indicates the class of the object which implements the associated interface.

```
type TComClass = class of TComObject;
property ImplementationClass: TComClass;
```

Description

ImplementationClass specifies the class that implements the Interface for which the factory creates objects. This is the COM object, not the corresponding skeleton.

TCorbaComObjectFactory methods

TCorbaComObjectFactory Alphabetically

In TCorbaComObjectFactory

- Create

Derived from TCorbaFactory

- Destroy

Derived from TInterfacedObject

- BeforeDestruction

Derived from TObject

- AfterConstruction
- ClassInfo
- ClassName
- ClassNameIs
- ClassParent
- ClassType
- CleanupInstance
- DefaultHandler
- Dispatch
- FieldAddress
- Free
- FreeInstance
- GetInterface
- GetInterfaceEntry
- GetInterfaceTable
- InheritsFrom
- InitInstance
- InstanceSize
- MethodAddress
- MethodName
- NewInstance
- SafeCallException

TCorbaComObjectFactory methods

TCorbaComObjectFactory By object

- AfterConstruction
- BeforeDestruction
- ClassInfo
- ClassName
- ClassNameIs
- ClassParent
- ClassType
- CleanupInstance
- Create
- DefaultHandler
- Destroy
- Dispatch
- FieldAddress
- Free
- FreeInstance
- GetInterface
- GetInterfaceEntry
- GetInterfaceTable
- InheritsFrom
- InitInstance
- InstanceSize
- MethodAddress
- MethodName
- NewInstance
- SafeCallException

TCorbaComObjectFactory.Create

TCorbaComObjectFactory Methods See also

Creates an instance of TCorbaComObjectFactory associated with a specified server interface.

```
type
  TComClass = class of TComObject;
  TCORBAInstancing = (iSingleInstance, iMultiInstance);
  TCorbaThreadModel = (tmMultiThreaded, tmSingleThread);
constructor Create(const InterfaceName, InstanceName, RepositoryId: string;
  const ImplGUID: TGUID; ImplementationClass: TComClass; Instancing:
  TCorbaInstancing = iMultiInstance; ThreadModel: TCorbaThreadModel =
  tmSingleThread);
```

Description

A call to the constructor for TCorbaComObjectFactory is automatically added to server applications when the developer exposes a COM automation server as a CORBA object.

The InterfaceName parameter specifies the name of the factory's interface (not the name of the interface for which the factory creates objects).

The InstanceName parameter specifies an instance identifier that is used by client applications to locate a specific factory instance.

The RepositoryId parameter specifies the CORBA Repository ID of the factory. This string has the form 'IDL:Modulename/FactoryInterfaceName:1.0'.

The ImplGUID parameter specifies the interface type (or GUID) for which the factory creates object instances.

The ImplementationClass parameter specifies the class that implements ImplGUID. This is the COM implementation class, not the corresponding skeleton class.

The Instancing parameter indicates whether the factory creates a single object instance that all CORBA clients share, or whether the factory creates a separate instance for each CORBA client.

The ThreadModel parameter indicates whether the ORB should serialize calls to a single factory instance or whether multiple clients can call the same factory simultaneously on multiple threads.

Accessibility

- Read-only

Hierarchy

TObject

TInterfacedObject

TCorbaFactory

TCorbaDispatchStub

Hierarchy Properties Methods See also

TCorbaDispatchStub is the base class for all automatically generated CORBA stub objects.

Unit

corbaobj

Description

TCorbaDispatchStub is the base class for automatically generated stub objects. It adds a (protected) dummy implementation of the IDispatch interface to the properties and methods inherited from TCorbaStub. This dummy implementation is provided so that CORBA server applications can use the Type Library editor to define CORBA interfaces.

Do not use TCorbaDispatchStub when defining custom stub objects. Instead, use TCorbaStub, which does not support an unused IDispatch interface. For details on how to define custom stub objects, see the TCorbaStub documentation.

Applications do not instantiate stub objects directly. Instead, the stub object is registered with the global stub manager. The code to register automatically generated stub objects is added to the _TLB unit when a server interface is defined. When the client application needs an interface for the CORBA server, it calls the CreateInstance method of the automatically generated CORBA stub factory.

TCorbaDispatchStub properties

TCorbaDispatchStub Alphabetically Legend

Derived from TInterfacedObject

- RefCount

TCorbaDispatchStub properties

- RefCount

TCorbaDispatchStub methods

TCorbaDispatchStub Alphabetically

Derived from TCorbaStub

- Create
- Destroy

Derived from TInterfacedObject

- BeforeDestruction

Derived from TObject

- AfterConstruction
- ClassInfo
- ClassName
- ClassNameIs
- ClassParent
- ClassType
- CleanupInstance
- DefaultHandler
- Dispatch
- FieldAddress
- Free
- FreeInstance
- GetInterface
- GetInterfaceEntry
- GetInterfaceTable
- InheritsFrom
- InitInstance
- InstanceSize
- MethodAddress
- MethodName
- NewInstance
- SafeCallException

TCorbaDispatchStub methods

- AfterConstruction
- BeforeDestruction
- ClassInfo
- ClassName
- ClassNameIs
- ClassParent
- ClassType
- CleanupInstance
- Create
- DefaultHandler
- Destroy
- Dispatch
- FieldAddress
- Free
- FreeInstance
- GetInterface
- GetInterfaceEntry
- GetInterfaceTable
- InheritsFrom
- InitInstance
- InstanceSize
- MethodAddress
- MethodName
- NewInstance
- SafeCallException

Accessibility

- Read-only

Hierarchy

TObject

TInterfacedObject

TCorbaStub

CorbaFactoryCreateStub Example

This example shows how to use CorbaFactoryCreateStub to obtain an interface for a CORBA server object.

```
var
  TheServerObject: IMyServerObject;
begin
  TheServerObject :=
  CorbaFactoryCreateStub('IDL:MyServer/MyServerObjectFactory:1.0',
  'MyServerFactory', 'Instance1', '',IMyServerObject) as IMyServerObject;
  ...
```

BindStub, CreateStub Example

This example shows how to use BindStub to obtain an interface for a CORBA server object. After obtaining an interface for the server object, it calls the CORBA stub manager to create a stub object for marshaling interface calls. The client application should use StubIntf to talk to the server so that the stub class can handle the marshaling.

```
var
  ServerIntf: IStub;
  StubIntf: IObject;
begin
  BindStub(PChar(Pointer('IDL:ServerIntf/MyObject:1.0')),
  PChar(Pointer('AnInstanceName')), PChar(Pointer('')), ORB, 0, ServerIntf);
  StubIntf := CORBAStubManager.CreateStub(IMyObject, ServerIntf);
...
```

CorbaBind, RegisterInterface Example

This example uses late (DII) binding to update the salary information of an employee named in Edit1. Edit2 is an edit control that indicates the percentage raise.

```
var
  HR, Emp, Payroll, Salary: TAny;
begin
  HR := CorbaBind('IDL:CompanyInfo/HR:1.0');
  Emp := HR.LookupEmployee(Edit1.Text);
  Payroll := CorbaBind('IDL:CompanyInfo/Payroll:1.0');
  Salary := Payroll.GetEmployeeSalary(Emp);
  Payroll.SetEmployeeSalary(Emp, Salary + (Salary * StrToInt(Edit2.Text) /
  100));
end;
```

Note: Before you can use CorbaBind, the relationship between each Repository ID and its interface type must be registered with the global CorbaInterfaceIDManager:

```
CorbaInterfaceIDManager.RegisterInterface(IHR, 'IDL:CInfo/HR:1.0');
CorbaInterfaceIDManager.RegisterInterface(IPayroll,
  'IDL:CInfo/Payroll:1.0');
```

ObjectToString, StringToObject Example

This example uses the global ORB variable to convert CORBA stub interfaces to strings that can be displayed to the user and strings chosen by the user to stub interfaces. The following code displays the names of three objects in a list box, given interface instances for their corresponding stub objects.

```
var
  Dept1, Dept2, Dept3: IDepartment;
begin
  Dept1 := TDepartmentFactory.CreateInstance('Sales');
  Dept1.SetDepartmentCode(120);
  Dept2 := TDepartmentFactory.CreateInstance('Marketing');
  Dept2.SetDepartmentCode(98);
  Dept3 := TSecondFactory.CreateInstance('Payroll');
  Dept3.SetDepartmentCode(49);
  ListBox1.Items.Add(ORB.ObjectToString(Dept1));
  ListBox1.Items.Add(ORB.ObjectToString(Dept2));
  ListBox1.Items.Add(ORB.ObjectToString(Dept3));
end;
```

The advantage of letting the ORB create strings for your objects is that you can use the StringToObject method to reverse this procedure:

```
var
  Dept: IDepartment;
begin
  Dept := ORB.StringToObject(ListBox1.Items[ListBox1.ItemIndex]);
  ... { do something with the selected department }
```

Bind, FindTypeCode, MakeStructure Example

This example uses the dynamic invocation interface (DII) to update an employee's salary using objects on a CORBA server. The FindTypeCode method obtains a description of the EmployeeName record (a CORBA structure consisting of first and last name), and the MakeStructure method creates the appropriate TAny value for a method that takes this structure as a parameter.

```
var
  HR, Name, Emp, Payroll, Salary: TAny;
begin
  with ORB do
  begin
    HR := Bind('IDL:CInfo/HR:1.0'); { bind to  HR object }
    { create a TAny record describing the employee }
    Name := MakeStructure(FindTypeCode('IDL:CInfo/EmployeeName:1.0',
                                  [Edit1.Text, Edit2.Text]));
    Emp := HR.LookupEmployee(Name); { use DII to get Employee object }
    Payroll := Bind('IDL:CInfo/Payroll:1.0'); { bind to  Payroll object }
  end;
  Salary := Payroll.GetEmployeeSalary(Emp); { look up current salary }
  { apply percentage increase from Edit3 and update salary }
  Payroll.SetEmployeeSalary(Emp,
    Salary + (Salary * StrToInt(Edit3.Text)/ 100));
end;
```

GetStub, CreateRequest, PutLong, Invoke, GetLong Example

This example shows how a stub object implements an interface call using its IStub interface to obtain marshaling buffers.

```
function TMyStub.AddTwoNums(Param1, Param2: Integer): Integer;
var
  OutBuf: IMarshalOutBuffer;
  InBuf: IMarshalInBuffer;
  TheStub: IStub;
begin
  GetStub(TheStub);
  TheStub.CreateRequest('AddTwoNums', 1, OutBuf);
  OutBuf.PutLong(Param1);
  OutBuf.PutLong(Param2);
  TheStub.Invoke(OutBuf, InBuf);
  Result := InBuf.GetLong;
end;
```

UnmarshalObject, UnmarshalText Example

This example shows how to use UnmarshalObject to obtain the interface for an object that was returned by a CORBA interface method.

```
function TLocalStub.GetASubObj(out ObjName: string): ISubObj;
var
  OutBuf: IMarshalOutBuffer;
  InBuf: IMarshalInBuffer;
  StubIntf: IStub;
begin
  GetStub(StubIntf);
  StubIntf.CreateRequest('GetASubObj', 1, OutBuf);
  StubIntf.Invoke(OutBuf, InBuf);
  Result := UnmarshalObject(InBuf, ISubObj) as ISubObj;
  ObjName := UnmarshalText(InBuf); { parameters come after return value }
end;
```

TCorbaFactory

Hierarchy Properties Methods See also

TCorbaFactory is the base class for objects instantiated remotely by the global CorbaBind function.

Unit

corbaobj

Description

TCorbaFactory descendants are created remotely by CORBA client applications when they call the global CorbaBind function.

Interfaces

TCorbaFactory implements the ISkeletonObject interface.

TCorbaFactory properties

TCorbaFactory Alphabetically Legend

In TCorbaFactory

- InstanceName
  - Instancing
  - InterfaceName
  - RepositoryID
  - ThreadModel

Derived from TInterfacedObject

- RefCount

TCorbaFactory properties

- InstanceName
  - Instancing
  - InterfaceName
  - RefCount
  - RepositoryID
  - ThreadModel

TCorbaFactory.InstanceName

TCorbaFactory Methods See also

Indicates the InstanceName for this factory.

```
property InstanceName: string;
```

Description

InstanceName is an optional identifier provided by client applications to identify this factory instance. It is passed to the CORBA factory constructor.

TCorbaFactory.Instancing

TCorbaFactory Methods See also

Indicates whether the factory creates an object instance for each CORBA client, or only a single object instance that handles all clients.

```
type TCorbaInstancing = (iSingleInstance, iMultiInstance);
property Instancing: TCorbaInstancing;
```

Description

Read Instancing to determine the instancing model specified when the CORBA factory was created. When Instancing is iSingleInstance, the CORBA factory creates a single, shared object instance that handles all clients. When Instancing is iMultiInstance, the CORBA factory creates a separate object instance for each client.

TCorbaFactory.InterfaceName

Indicates the name of the factory's interface.

```
property InterfaceName: string;
```

Description

Read Interfacename to determine the name of the factory's interface as it appears in the IDL specification. This value is assigned when the CORBA factory is created.

TCorbaFactory.RepositoryID

Indicates the factory's Repository ID.

```
property RepositoryID: string;
```

Description

Read RepositoryID to determine the factory's Repository ID. This value is assigned when the CORBA factory is created.

TCorbaFactory.ThreadModel

Indicates the threading model supported by the CORBA server.

```
type TCorbaThreadModel = (tmMultiThreaded, tmSingleThread);
property RepositoryID: string;
```

Description

Read ThreadModel to determine what thread model the Orb uses when calling the CORBA factory and the objects it creates. When ThreadModel is tmMultiThreaded, object instances can receive calls from multiple clients simultaneously on separate threads. When ThreadModel is tmSingleThread, each object instance is guaranteed to receive only one interface call at a time (although other objects in the server application may be servicing calls on other threads).

TCorbaFactory methods

TCorbaFactory Alphabetically

In TCorbaFactory

- Create
- Destroy

Derived from TInterfacedObject

- BeforeDestruction

Derived from TObject

- AfterConstruction
- ClassInfo
- ClassName
- ClassNameIs
- ClassParent
- ClassType
- CleanupInstance
- DefaultHandler
- Dispatch
- FieldAddress
- Free
- FreeInstance
- GetInterface
- GetInterfaceEntry
- GetInterfaceTable
- InheritsFrom
- InitInstance
- InstanceSize
- MethodAddress
- MethodName
- NewInstance
- SafeCallException

TCorbaFactory methods

- AfterConstruction
- BeforeDestruction
- ClassInfo
- ClassName
- ClassNameIs
- ClassParent
- ClassType
- CleanupInstance
- Create
- DefaultHandler
- Destroy
- Dispatch
- FieldAddress
- Free
- FreeInstance
- GetInterface
- GetInterfaceEntry
- GetInterfaceTable
- InheritsFrom
- InitInstance
- InstanceSize
- MethodAddress
- MethodName
- NewInstance
- SafeCallException

TCorbaFactory.Create

TCorbaFactory Methods See also

Creates an instance of TCorbaFactory associated with a specified server interface.

```
type
  TCORBAInstancing = (iSingleInstance, iMultiInstance);
  TCorbaThreadModel = (tmMultiThreaded, tmSingleThread);

constructor Create(const InterfaceName, InstanceName, RepositoryId: string;
  const ImplGUID: TGUID; Instancing: TCorbaInstancing = iMultiInstance;
  ThreadModel: TCorbaThreadModel = tmSingleThread);
```

Description

Call Create to create an instance of TCorbaFactory that can respond to client interface calls.

The InterfaceName parameter specifies the name of the factory's interface (not the name of the interface for which the factory creates objects).

The InstanceName parameter specifies an instance identifier that is used by client applications to locate a specific factory instance.

The RepositoryId parameter specifies the CORBA Repository ID of the factory. This string has the form 'IDL:ModuleName/FactoryInterfaceName:1.0'.

The ImplGUID parameter specifies the interface type (or GUID) for which the factory creates object instances.

The Instancing parameter indicates whether the factory creates a single object instance that is shared by all CORBA clients, or whether the factory creates a separate instance for each CORBA client.

The ThreadModel parameter indicates whether the ORB should serialize calls to a single factory instance or whether multiple clients can call the same factory simultaneously on multiple threads.

TCorbaFactory.Destroy

Frees an instance of TCorbaFactory.

```
destructor Destroy; override;
```

Description

Do not call the CORBA factory destructor. CORBA factories are freed automatically when they are not needed.

Accessibility

- Read-only

Hierarchy

TObject

TInterfacedObject

TCorbaInterfaceIDManager

Hierarchy Methods See also

TCorbaInterfaceIDManager keeps track of which Repository IDs represent which interfaces.

Unit

corbaobj

Description

TCorbaInterfaceIDManager is the type of the global CorbaInterfaceIDManager variable. Applications do not create CORBA interface ID managers. Instead, they use the global CorbaInterfaceIDManager variable to register the association between a CORBA interface and its repository ID. This registered association is then used when an application calls the global CorbaBind function or the ORB's Bind method.

Note: In automatically generated stub-and-skeleton units (_TLB.pas files), the code to register interfaces with the interface ID manager is added automatically.

TCorbaInterfaceIDManager methods

TCorbaInterfaceIDManager Alphabetically

In TCorbaInterfaceIDManager

- FindGUID
- FindID
- RegisterInterface
- SearchGUID
- SearchID

Derived from TCorbaListManager

- Create
- Destroy

Derived from TObject

- AfterConstruction
- BeforeDestruction
- ClassInfo
- ClassName
- ClassNameIs
- ClassParent
- ClassType
- CleanupInstance
- DefaultHandler
- Dispatch
- FieldAddress
- Free
- FreeInstance
- GetInterface
- GetInterfaceEntry
- GetInterfaceTable
- InheritsFrom
- InitInstance
- InstanceSize
- MethodAddress
- MethodName
- NewInstance
- SafeCallException

TCorbaInterfaceIDManager methods

TCorbaInterfaceIDManager By object

- AfterConstruction
- BeforeDestruction
- ClassInfo
- ClassName
- ClassNameIs
- ClassParent
- ClassType
- CleanupInstance
- Create
- DefaultHandler
- Destroy
- Dispatch
- FieldAddress
- FindGUID
- FindID
- Free
- FreeInstance
- GetInterface
- GetInterfaceEntry
- GetInterfaceTable
- InheritsFrom
- InitInstance
- InstanceSize
- MethodAddress
- MethodName
- NewInstance
- RegisterInterface
- SafeCallException
- SearchGUID
- SearchID

TCorbaInterfaceIDManager.FindGUID

TCorbaInterfaceIDManager See also

Returns the GUID for an interface identified by its Repository ID.

```
function FindGUID(const RepositoryID: string): TGUID;
```

Description

Call FindGUID to obtain a GUID for an interface, given its Repository ID. The Repository ID parameter specifies this Repository ID.

Note: Before FindGUID can locate the GUID for an interface, the interface must be registered. Otherwise, FindGUID raises an ECorbaException. Use the RegisterInterface method to register an interface.

TCorbaInterfaceIDManager.FindID

TCorbaInterfaceIDManager See also

Returns the Repository ID for an interface type.

```
function FindID(const IID: TGUID): string;
```

Description

Call FindID to obtain a Repository ID for an interface. The IID parameter specifies the GUID of the interface (or the interface type).

Note: Before FindID can locate the Repository ID for an interface, the interface must be registered. Otherwise, FindID raises an ECorbaException. Use the RegisterInterface method to register an interface.

TCorbaInterfaceIDManager.RegisterInterface

TCorbaInterfaceIDManager See also Example

Registers an interface with the CORBA interface ID manager.

```
procedure RegisterInterface(const IID: TGUID; const RepositoryID: string);
```

Description

Call RegisterInterface to register the association between an interface type and its Repository ID. Specify the interface type, or its GUID, as the IID parameter. Specify the Repository ID as the RepositoryID parameter.

When defining interfaces in a CORBA server application, code to register the interfaces is automatically added to the initialization section of the stub-and-skeleton unit (_TLB.pas file).

Note: Although CORBA interfaces do not use globally unique identifiers (GUIDs) for COM support, they must still be declared using an interface identifier so that the CORBA interface ID manager can keep track of them.

TCorbaInterfaceIDManager.SearchGUID

TCorbaInterfaceIDManager See also

Returns the GUID for an interface identified by its Repository ID.

```
function SearchGUID(const RepositoryID: string; out IID: TGUID): Boolean;
```

Description

Call SearchGUID to obtain a GUID for an interface, given its Repository ID. The Repository ID parameter specifies this Repository ID.

If the specified Repository ID has not been registered with the interface ID manager, SearchGUID returns False. If the specified Repository ID has been registered, SearchGUID returns True, passing the requested GUID as the IID parameter.

TCorbaInterfaceIDManager.SearchID

TCorbaInterfaceIDManager See also

Returns the Repository ID for an interface type.

```
function SearchID(const IID: TGUID; out RepositoryID: string): Boolean;
```

Description

Call SearchID to obtain a Repository ID for an interface. The IID parameter specifies the GUID of the interface (or the interface type).

If the specified GUID has not been registered with the interface ID manager, SearchID returns False. If the specified GUID has been registered, SearchID returns True, passing the requested Repository ID as the RepositoryID parameter.

Hierarchy

TObject

TCorbaListManager

TCorbaImplementation

Hierarchy Methods See also

TCorbaImplementation is the base class for classes that implement CORBA server interfaces.

Unit

corbaobj

Description

The CORBA Object Wizard defines a descendant of TCorbaImplementation to represent each object defined in the public interface of a CORBA server application. Developers add the body to this TCorbaImplementation descendant's methods to create a CORBA server.

TCorbaImplementation introduces protected methods to implement the IDispatch and IUnknown interfaces. This is because automatically defined CORBA interfaces must be descendants of IDispatch in order to work with the Type Library editor.

TCorbaImplementation methods

TCorbaImplementation Alphabetically

In TCorbaImplementation

- Create

Derived from TObject

- AfterConstruction
- BeforeDestruction
- ClassInfo
- ClassName
- ClassNameIs
- ClassParent
- ClassType
- CleanupInstance
- DefaultHandler
- Destroy
- Dispatch
- FieldAddress
- Free
- FreeInstance
- GetInterface
- GetInterfaceEntry
- GetInterfaceTable
- InheritsFrom
- InitInstance
- InstanceSize
- MethodAddress
- MethodName
- NewInstance
- SafeCallException

TCorbaImplementation methods

- AfterConstruction
- BeforeDestruction
- ClassInfo
- ClassName
- ClassNameIs
- ClassParent
- ClassType
- CleanupInstance
- Create
- DefaultHandler
- Destroy
- Dispatch
- FieldAddress
- Free
- FreeInstance
- GetInterface
- GetInterfaceEntry
- GetInterfaceTable
- InheritsFrom
- InitInstance
- InstanceSize
- MethodAddress
- MethodName
- NewInstance
- SafeCallException

TCorbaImplementation.Create

TCorbaImplementation See also

Creates and initializes an instance of TCorbaImplementation.

```
constructor Create(Controller: IObject; AFactory: TCorbaFactory); virtual;
```

Description

Call Create to create an instance of the CORBA implementation class.

In most cases, the Controller parameter is **nil**. However, if the CORBA implementation is a sub-object of a COM-style aggregate, the Controller parameter specifies the controlling aggregate object.

When the CORBA implementation is instantiated by a factory object, the factory passes a reference to itself as the AFactory parameter.

Hierarchy

TObject

TCorbaListManager

Hierarchy Methods See also

TCorbaListManager is the base class for classes that keep track of available CORBA classes and interfaces.

Unit

corbaobj

Description

Do not use TCorbaListManager in applications. TCorbaListManager is intended only as a base class for global objects in a CORBA application. It provides thread support for the global objects that manage lists of classes and interfaces for CORBA clients and servers.

TCorbaListManager methods

TCorbaListManager Alphabetically

In TCorbaListManager

- Create
- Destroy

Derived from TObject

- AfterConstruction
- BeforeDestruction
- ClassInfo
- ClassName
- ClassNameIs
- ClassParent
- ClassType
- CleanupInstance
- DefaultHandler
- Dispatch
- FieldAddress
- Free
- FreeInstance
- GetInterface
- GetInterfaceEntry
- GetInterfaceTable
- InheritsFrom
- InitInstance
- InstanceSize
- MethodAddress
- MethodName
- NewInstance
- SafeCallException

TCorbaListManager methods

TCorbaListManager By object

- AfterConstruction
- BeforeDestruction
- ClassInfo
- ClassName
- ClassNameIs
- ClassParent
- ClassType
- CleanupInstance
- Create
- DefaultHandler
- Destroy
- Dispatch
- FieldAddress
- Free
- FreeInstance
- GetInterface
- GetInterfaceEntry
- GetInterfaceTable
- InheritsFrom
- InitInstance
- InstanceSize
- MethodAddress
- MethodName
- NewInstance
- SafeCallException

TCorbaListManager.Create

TCorbaListManager See also

Creates and initializes an instance of TCorbaListManager.

```
constructor Create;
```

Description

Most applications do not create CORBA list managers. Instead, these objects are instantiated automatically as global variables.

Create creates a TMulitReadExclusiveWriteSynchronizer for the CORBA list manager so that it can efficiently handle threading issues for the list it maintains.

TCorbaListManager.Destroy

TCorbaListManager See also

Frees the TCorbaListmanager instance.

```
destructor Destroy; override;
```

Description

Do not call Destroy directly in an application. Instead, call Free. Free verifies that the list manager is not already freed, and only then calls Destroy.

Most applications do not need to create or destroy list manager instances. Instead, these objects are available as global objects in the Corbaobj unit.

Hierarchy

TObject

TCorbaObjectFactory

Hierarchy Properties Methods See also

TCorbaObjectFactory creates a CORBA server object.

Unit

corbaobj

Description

The CORBA Object Wizard adds code to create an instance of TCorbaObjectFactory for every object exposed in a CORBA server application. In response to requests from CORBA client applications, this factory instance creates CORBA skeletons and implementation classes.

By using a Factory object for each CORBA server object, the CORBA server can control the lifetime of objects that are created for a specific client and which should be freed when client's have finished with them.

TCorbaObjectFactory properties

TCorbaObjectFactory Alphabetically Legend

In TCorbaObjectFactory

- ImplementationClass

Derived from TCorbaFactory

- InstanceName
  - Instancing
  - InterfaceName
  - RepositoryID
  - ThreadModel

Derived from TInterfacedObject

- RefCount

TCorbaObjectFactory properties

- ImplementationClass
  - InstanceName
  - Instancing
  - InterfaceName
  - RefCount
  - RepositoryID
  - ThreadModel

TCorbaObjectFactory.ImplementationClass

TCorbaObjectFactory Methods See also

Indicates the class of the object which implements the associated interface.

```
type TCorbaImplementationClass = class of TCorbaImplementation;
property ImplementationClass: TCorbaImplementationClass;
```

Description

ImplementationClass specifies the class that implements the Interface for which the factory creates objects. This is the actual implementation class, not the corresponding skeleton class.

TCorbaObjectFactory methods

TCorbaObjectFactory Alphabetically

In TCorbaObjectFactory

- Create

Derived from TCorbaFactory

- Destroy

Derived from TInterfacedObject

- BeforeDestruction

Derived from TObject

- AfterConstruction
- ClassInfo
- ClassName
- ClassNameIs
- ClassParent
- ClassType
- CleanupInstance
- DefaultHandler
- Dispatch
- FieldAddress
- Free
- FreeInstance
- GetInterface
- GetInterfaceEntry
- GetInterfaceTable
- InheritsFrom
- InitInstance
- InstanceSize
- MethodAddress
- MethodName
- NewInstance
- SafeCallException

TCorbaObjectFactory methods

TCorbaObjectFactory By object

- AfterConstruction
- BeforeDestruction
- ClassInfo
- ClassName
- ClassNameIs
- ClassParent
- ClassType
- CleanupInstance
- Create
- DefaultHandler
- Destroy
- Dispatch
- FieldAddress
- Free
- FreeInstance
- GetInterface
- GetInterfaceEntry
- GetInterfaceTable
- InheritsFrom
- InitInstance
- InstanceSize
- MethodAddress
- MethodName
- NewInstance
- SafeCallException

TCorbaObjectFactory.Create

Creates an instance of TCorbaObjectFactory associated with a specified server interface.

```
type
  TCorbaImplementationClass = class of TCorbaImplementation;
  TCORBAInstancing = (iSingleInstance, iMultiInstance);
  TCorbaThreadModel = (tmMultiThreaded, tmSingleThread);

constructor Create(const InterfaceName, InstanceName, RepositoryId: string;
  const ImplGUID: TGUID; ImplementationClass: TCorbaImplementationClass;
  Instancing: TCorbaInstancing = iMultiInstance; ThreadModel:
  TCorbaThreadModel = tmSingleThread);
```

Description

A call to the constructor for TCorbaObjectFactory is automatically added to CORBA server applications when the developer defines CORBA interfaces.

The InterfaceName parameter specifies the name of the factory's interface (not the name of the interface for which the factory creates objects).

The InstanceName parameter specifies an instance identifier that is used by client applications to locate a specific factory instance.

The RepositoryId parameter specifies the CORBA Repository ID of the factory. This string has the form 'IDL:Modulename/FactoryInterfaceName:1.0'.

The ImplGUID parameter specifies the interface type (or GUID) for which the factory creates object instances.

The ImplementationClass parameter specifies the class that implements ImplGUID. This is the actual implementation class, not the corresponding skeleton class.

The Instancing parameter indicates whether the factory creates a single object instance that all CORBA clients share, or whether the factory creates a separate instance for each CORBA client.

The ThreadModel parameter indicates whether the ORB should serialize calls to a single factory instance or whether multiple clients can call the same factory simultaneously on multiple threads.

Accessibility

- Read-only

Hierarchy

TObject

TInterfacedObject

TCorbaFactory

TCorbaSkeleton

Hierarchy Properties Methods See also

TCorbaSkeleton is the base class for all CORBA skeleton objects.

Unit

corbaobj

Description

Most applications do not need to work directly with CORBA skeleton objects. TCorbaSkeleton objects are automatically declared and implemented for each server interface in a CORBA server application. To define a server interface, use the Type Library editor. The TCorbaSkeleton class is defined in the stub and skeleton unit (_TLB.pas file).

TCorbaSkeleton descendants handle the details of marshaling interface calls and communicating with the Orb in a CORBA server application. They do not actually implement the server interface. Instead, they pass calls on to an object instance for which they hold an interface.

Interfaces

TCorbaSkeleton implements the ISkeletonObject interface.

TCorbaSkeleton properties

TCorbaSkeleton Alphabetically Legend

Derived from TInterfacedObject

- RefCount

TCorbaSkeleton properties

TCorbaSkeleton By object Legend

- RefCount

TCorbaSkeleton methods

TCorbaSkeleton Alphabetically Legend

In TCorbaSkeleton

- Create
- Destroy
- Execute
- GetImplementation
- GetSkeleton
- InitSkeleton

Derived from TInterfacedObject

- BeforeDestruction

Derived from TObject

- AfterConstruction
- ClassInfo
- ClassName
- ClassNameIs
- ClassParent
- ClassType
- CleanupInstance
- DefaultHandler
- Dispatch
- FieldAddress
- Free
- FreeInstance
- GetInterface
- GetInterfaceEntry
- GetInterfaceTable
- InheritsFrom
- InitInstance
- InstanceSize
- MethodAddress
- MethodName
- NewInstance
- SafeCallException

TCorbaSkeleton methods

- AfterConstruction
- BeforeDestruction
- ClassInfo
- ClassName
- ClassNameIs
- ClassParent
- ClassType
- CleanupInstance
- Create
- DefaultHandler
- Destroy
- Dispatch
- Execute
- FieldAddress
- Free
- FreeInstance
- GetImplementation
- GetInterface
- GetInterfaceEntry
- GetInterfaceTable
- GetSkeleton
- InheritsFrom
- InitInstance
- InitSkeleton
- InstanceSize
- MethodAddress
- MethodName
- NewInstance
- SafeCallException

TCorbaSkeleton.Create

Creates an instance of TCorbaSkeleton to represent a specified server interface.

```
type IObject = System.IUnknown;
constructor Create(InstanceName: string; const Impl: IObject); virtual;
```

Description

Do not directly instantiate CORBA skeleton objects. TCorbaSkeleton objects are created by the associated CORBA factory.

The InstanceName parameter specifies an optional name for the CORBA skeleton instance. The Impl parameter is the interface to an object instance that implements the interface which client applications call.

TCorbaSkeleton.Destroy

Frees an instance of TCorbaSkeleton.

```
destructor Destroy; override;
```

Description

Do not call the CORBA skeleton destructor directly. CORBA skeletons are destroyed automatically.

TCorbaSkeleton.Execute

Executes a specified interface call.

```
function Execute(Operation: PChar; const Strm: IMarshalInBuffer; Cookie:
  Pointer): CorbaBoolean; stdcall;
```

Description

Applications do not usually call the execute method. It is called automatically when the Orb passes in an interface invocation from a client application.

The Operation parameter specifies the interface member's name. The Strm parameter is a marshaling buffer that contains all the parameter values received from the client. The Cookie parameter specifies an internal identifier that allows the skeleton object to keep track of which marshaling buffers are associated with which interface calls.

If the TCorbaSkeleton descendant does not implement a method of the name Operation, or if an exception is raised when it tries to execute the specified method, Execute returns 0 (False). Otherwise, Execute calls the specified method and returns 1 (True).

TCorbaSkeleton.GetImplementation

Returns an interface for the associated implementation object.

```
type IObject = System.IUnknown;
procedure GetImplementation(out Impl: IObject); virtual; stdcall;
```

Description

CORBA skeletons use an implementation object rather than directly implementing the server object. An interface to this implementation object is passed to the TCorbaSkeleton descendant when it is created. GetImplementation returns this interface as the Impl parameter.

Most applications do not need to call GetImplementation. Component writers deriving their own TCorbaSkeleton descendants can use GetImplementation for forwarding method calls to the implementation object once they are unmarshaled.

TCorbaSkeleton.GetSkeleton

Returns an ISkeleton interface for marshaling interface calls.

```
procedure GetSkeleton(out servant: IServant); stdcall;
```

Description

The CORBA factory calls GetSkeleton to obtain the implementation interface and marshaling buffer for return values.

TCorbaSkeleton.InitSkeleton

TCorbaSkeleton Methods See also

Initializes the ISkeleton interface for a skeleton instance.

```
type TCorbaThreadModel = (tmMultiThreaded, tmSingleThread);
procedure InitSkeleton(const InterfaceName, InstanceName, RepositoryID:
  string; ThreadModel: TCorbaThreadModel; ClientRefCount: Boolean);
```

Description

Most applications do not need to call InitSkeleton because it is called from the constructor of automatically generated TCorbaSkeleton descendants. Component writers should add a call to InitSkeleton when implementing the constructor of a TCorbaSkeleton descendant.

The InterfaceName specifies the base name of the interface represented by the skeleton object. This is the interface name as it appears in the IDL specification.

The InstanceName parameter is an optional name for the skeleton instance.

The RepositoryID is the Repository ID for the interface represented by the skeleton object.

The ThreadModel parameter indicates whether the Orb ensures that each skeleton instance only receives one client request at a time (tmSingleThread), or whether a single skeleton instance may receive multiple client requests simultaneously (tmMultiThreaded). This value is ignored if the CORBA object instance is instantiated indirectly using a CORBA factory object. Instead, the factory's threading model takes precedence.

The ClientRefCount parameter indicates whether a reference count should be maintained for client connections.

Accessibility

- Read-only

Scope

- Protected

Hierarchy

TObject

TInterfacedObject

TCorbaSkeletonManager

Hierarchy Methods See also

TCorbaSkeletonManager keeps track of which skeleton classes represent which interfaces.

Unit

corbaobj

Description

TCorbaSkeletonManager is the type of the global CorbaSkeletonManager variable. Applications do not create CORBA skeleton managers. Instead, they use the global CorbaSkeletonManager variable to register the association between a TCorbaSkeleton descendant and the interface for which it provides marshaling.

TCorbaSkeletonManager methods

TCorbaSkeletonManager Alphabetically

In TCorbaSkeletonManager

- CreateSkeleton
- RegisterSkeleton

Derived from TCorbaListManager

- Create
- Destroy

Derived from TObject

- AfterConstruction
- BeforeDestruction
- ClassInfo
- ClassName
- ClassNameIs
- ClassParent
- ClassType
- CleanupInstance
- DefaultHandler
- Dispatch
- FieldAddress
- Free
- FreeInstance
- GetInterface
- GetInterfaceEntry
- GetInterfaceTable
- InheritsFrom
- InitInstance
- InstanceSize
- MethodAddress
- MethodName
- NewInstance
- SafeCallException

TCorbaSkeletonManager methods

TCorbaSkeletonManager By object

- AfterConstruction
- BeforeDestruction
- ClassInfo
- ClassName
- ClassNameIs
- ClassParent
- ClassType
- CleanupInstance
- Create
- CreateSkeleton
- DefaultHandler
- Destroy
- Dispatch
- FieldAddress
- Free
- FreeInstance
- GetInterface
- GetInterfaceEntry
- GetInterfaceTable
- InheritsFrom
- InitInstance
- InstanceSize
- MethodAddress
- MethodName
- NewInstance
- RegisterSkeleton
- SafeCallException

TCorbaSkeletonManager.CreateSkeleton

TCorbaSkeletonManager See also

Instantiates a skeleton object given an interface ID and an implementation interface.

```
type IObject = System.IUnknown;
function CreateSkeleton(IID: TGUID; const InstanceName: string; const Impl:
  IObject): ISkeletonObject;
```

Description

Do not call CreateSkeleton directly. CreateSkeleton is called automatically when the CORBA server needs to create an instance of the skeleton object. When this call is made depends on the instancing option chosen in the CORBA Automation Object Wizard or CORBA Data Module Wizard.

The IID parameter identifies the interface type which is made available to client applications. The InstanceName parameter optionally specifies an instance name for the skeleton object. The Impl parameter is an instance of the implementation class's interface. CreateSkeleton returns an interface for the skeleton object instance.

Note: Before CreateSkeleton can instantiate a skeleton object and return its interface, the skeleton class must be registered. Use the RegisterSkeleton method to register a skeleton class.

TCorbaSkeletonManager.RegisterSkeleton

TCorbaSkeletonManager See also

Registers a skeleton class with the CORBA skeleton manager.

```
type
  TCorbaSkeletonClass = class of TCorbaSkeleton;
procedure RegisterSkeleton(IID: TGUID; Skeleton: TCORBASkeletonClass);
```

Description

Call RegisterSkeleton to register a custom skeleton class with the CORBA skeleton manager. The code to register automatically generated skeleton classes is added to the initialization section of the stub and skeleton unit (_TLB.pas file) when you create a server interface using the Type Library editor.

RegisterSkeleton records the association between a skeleton class (specified by the Skeleton parameter) and an interface (specified by the IID parameter). The Interface type can be passed for the IID parameter instead of the GUID alone. This makes for more readable code such as the following:

```
CORBASkeletonManager.RegisterSkeleton(IMyServerObject,
  TMyServerObjectSkeleton);
```

Note: Although CORBA interfaces do not use globally unique identifiers (GUIDs) for COM support, they must still be declared using an interface identifier so that the CORBA skeleton manager can keep track of them.

Hierarchy

TObject

TCorbaListManager

TCorbaStub

Hierarchy Properties Methods See also

TCorbaStub is the base class for all CORBA stub objects.

Unit

corbaobj

Description

Use TCorbaStub as a base class when deriving custom stub objects. Stub objects handle the marshaling of interface calls for a CORBA client application. Each TCorbaStub object obtains an IStub interface from its constructor. This IStub interface can be used to marshal interface calls.

To define a custom stub object, add properties and methods to the TCorbaStub descendant for each property and method in the interface it represents. When implementing the methods, obtain IMarshalInBuffer and IMarshalOutBuffer interfaces from the IStub interface of the TCorbaStub. Use these marshaling interfaces to push parameters before making an IStub call and pop return values when the call is complete.

Applications do not instantiate stub objects directly. Instead, the stub object is registered with the global stub manager that can be accessed using the global CorbaStubManager variable. The code to register automatically generated stub objects is added to the _TLB unit when a server interface is saved. When the client application needs an interface for the CORBA server, it calls the CreateInstance method of the automatically generated CORBA stub factory.

Applications that use custom stub objects must add the code to register these stub objects by calling the RegisterStub method of the CORBA stub manager and the RegisterInterface method of the global CorbaInterfaceIDManager. Once the stub object is registered, client applications can get an interface by calling the global CorbaBind function or the global BindStub procedure.

Interfaces

TCorbaStub implements the IStubObject and ICorbaObject interfaces.

TCorbaStub properties

Derived from TInterfacedObject

- RefCount

TCorbaStub properties

TCorbaStub By object Legend

- RefCount

TCorbaStub methods

TCorbaStub Alphabetically Legend

In TCorbaStub

- Create
- Destroy
- GetStub
- Hash
- IsA
- NonExistent
- SetPrincipal

Derived from TInterfacedObject

- BeforeDestruction

Derived from TObject

- AfterConstruction
- ClassInfo
- ClassName
- ClassNameIs
- ClassParent
- ClassType
- CleanupInstance
- DefaultHandler
- Dispatch
- FieldAddress
- Free
- FreeInstance
- GetInterface
- GetInterfaceEntry
- GetInterfaceTable
- InheritsFrom
- InitInstance
- InstanceSize
- MethodAddress
- MethodName
- NewInstance
- SafeCallException

TCorbaStub methods

TCorbaStub By object Legend

- AfterConstruction
- BeforeDestruction
- ClassInfo
- ClassName
- ClassNameIs
- ClassParent
- ClassType
- CleanupInstance
- Create
- DefaultHandler
- Destroy
- Dispatch
- FieldAddress
- Free
- FreeInstance
- GetInterface
- GetInterfaceEntry
- GetInterfaceTable
- GetStub
- Hash
- InheritsFrom
- InitInstance
- InstanceSize
- IsA
- MethodAddress
- MethodName
- NewInstance
- NonExistent
- SafeCallException
- SetPrincipal

TCorbaStub.Create

TCorbaStub Methods See also

Creates an instance of TCorbaStub to represent a specified server interface.

```
constructor Create(const Stub: IStub); virtual;
```

Description

Do not directly instantiate CORBA stub objects. Instead, call the global CorbaBind function or the CORBA stub manager's CreateStub method to create the stub object. Before the CORBA stub manager can create a CORBA stub object, the stub class must be registered using the CORBA stub manager's RegisterStub method. If you are using CorbaBind function, the associated interface type must be registered with the global CORBA interface ID manager as well.

The IStub parameter specifies an instance of the server interface which the CORBA stub object represents. TCorbaStub descendants use this interface for marshaling calls to the CORBA server application.

TCorbaStub.Destroy

Frees an instance of TCorbaStub.

```
destructor Destroy; override;
```

Description

Do not call the CORBA stub destructor directly. The CORBA stub manager handles the creation and destruction of all stub objects.

TCorbaStub.GetStub

TCorbaStub Methods See also Example

Returns the interface to an object on the CORBA server.

```
procedure GetStub(out stub: IStub); stdcall;
```

Description

Client applications do not call this protected method. Component writers use GetStub to obtain the server interface when implementing methods that marshal interface calls.

Instead of using the server interface in a CORBA client application, call the methods of the TCorbaStub descendant, which then marshals the calls. Applications can obtain an interface to the TCorbaStub descendant using the global CorbaBind function or the CORBA stub manager's CreateStub method.

TCorbaStub.Hash

Returns a hash value for the server object instance.

```
function Hash(Maximum: Integer): Integer;
```

Description

Call Hash to obtain a hash value for the CORBA object associated with the CORBA stub. This value is not guaranteed to be unique, but will remain consistent through the lifetime of the object reference.

TCorbaStub.IsA

Indicates whether the CORBA stub represents a particular type.

```
function IsA(const LogicalTypeId: string): Boolean;
```

Description

Call IsA to check whether the associated server object is of a specified type. Specify the type by passing its base name as the LogicalTypeId parameter This is the name specified in the object's IDL specification. IsA returns True if the object on the CORBA server is an instance of the specified type or one of its descendants. IsA returns False otherwise.

TCorbaStub.NonExistent

Indicates whether the server object is instantiated.

```
function NonExistent: Boolean;
```

Description

Call NonExistent to test whether the object on the server is still instantiated. NonExistent returns False if the server object is still available. NonExistent returns True if the server instance has been freed.

TCorbaStub.SetPrincipal

TCorbaStub Methods See also

Sends an array of bytes to the server

```
type TCorbaPrincipal = array of Byte;
procedure SetPrincipal(const Principal: TCorbaPrincipal);
```

Description

Call SetPrincipal to send an arbitrary value to the server application. Once the client calls SetPrincipal, the server application can read this value by calling the Boa's GetPrincipal method. SetPrincipal and GetPrincipal allow CORBA clients to pass custom state information to servers, or to pass information that identifies the client.

Note: To obtain a TCorbaPrincipal for the Principal parameter, use the global MakePrincipal function.

Accessibility

- Read-only

Scope

- Protected

Hierarchy

TObject

TInterfacedObject

TCorbaStubManager

Hierarchy Methods See also

TCorbaStubManager keeps track of which stub classes represent specific server interfaces.

Unit

corbaobj

Description

TCorbaStubManager is the type of the global CorbaStubManager variable. Applications do not create CORBA stub managers. Instead, they use the global CorbaStubManager variable to register the association between a TCorbaStub descendant and the interface of a CORBA server object.

Once a TCorbaStub class is registered with the global CorbaStubManager, the CORBA stub manager can instantiate stub objects which provide marshaling support for server interfaces.

TCorbaStubManager methods

TCorbaStubManager Alphabetically

In TCorbaStubManager

- CreateStub
- RegisterStub

Derived from TCorbaListManager

- Create
- Destroy

Derived from TObject

- AfterConstruction
- BeforeDestruction
- ClassInfo
- ClassName
- ClassNameIs
- ClassParent
- ClassType
- CleanupInstance
- DefaultHandler
- Dispatch
- FieldAddress
- Free
- FreeInstance
- GetInterface
- GetInterfaceEntry
- GetInterfaceTable
- InheritsFrom
- InitInstance
- InstanceSize
- MethodAddress
- MethodName
- NewInstance
- SafeCallException

TCorbaStubManager methods

TCorbaStubManager By object

- AfterConstruction
- BeforeDestruction
- ClassInfo
- ClassName
- ClassNameIs
- ClassParent
- ClassType
- CleanupInstance
- Create
- CreateStub
- DefaultHandler
- Destroy
- Dispatch
- FieldAddress
- Free
- FreeInstance
- GetInterface
- GetInterfaceEntry
- GetInterfaceTable
- InheritsFrom
- InitInstance
- InstanceSize
- MethodAddress
- MethodName
- NewInstance
- RegisterStub
- SafeCallException

TCorbaStubManager.CreateStub

TCorbaStubManager See also Example

Instantiates a stub object given a CORBA server interface instance.

```
type IObject = System.IUnknown;
function CreateStub(IID: TGUID; const Stub: IStub): IObject;
```

Description

Call CreateStub to create a stub object for marshaling calls over an interface that was obtained using the global BindStub procedure. Before CreateStub can instantiate a stub object and return its interface, the stub class must be registered. Use the RegisterStub method to register a stub class.

The IID parameter specifies the globally unique identifier of the server interface class. Use either the GUID or the interface type for this parameter.

The Stub parameter specifies a server interface instance which can be obtained from a call to BindStub. CreateStub returns an interface for the instantiated stub object.

TCorbaStubManager.RegisterStub

TCorbaStubManager See also

Registers a stub class with the CORBA stub manager.

```
type
  TCorbaStubClass = class of TCorbaStub;
procedure RegisterStub(IID: TGUID; Stub: TCorbaStubClass);
```

Description

Call RegisterStub to register a custom stub class with the CORBA stub manager. The code to register automatically generated stub classes is added to the initialization section of the stub and skeleton unit (_TLB.pas file) when you create a server interface.

RegisterStub records the association between a stub class (specified by the Stub parameter) and an interface type (specified by the IID parameter). The Interface type can be passed instead of the GUID for the IID parameter . This makes for more readable code such as the following:

```
CORBAStubManager.RegisterStub(IMyServerObject, TMyServerObjectStub);
```

Note: Although CORBA interfaces do not need globally unique identifiers (GUIDs) for COM support, they must still be declared using an interface identifier so that the CORBA stub manager can keep track of them.

Hierarchy

TObject

TCorbaListManager

TCorbaVCLComponentFactory

Hierarchy Properties Methods See also

TCorbaVCLComponentFactory creates a CORBA server object.

Unit

corbaobj

Description

The CORBA Data Module Wizard adds code to create an instance of TCorbaVCLComponentFactory to the initialization section of the CORBA data module's unit. In response to requests from CORBA client applications, this factory instance creates or locates the CORBA data module and forwards incoming interface calls.

By using a Factory object for the CORBA data module, the CORBA server can support the model where a single CORBA data module instance is devoted to each client connection. Even when the CORBA server uses a single shared instance of the CORBA data module, the factory object is used so as to provide a more uniform architecture.

TCorbaVCLComponentFactory properties

TCorbaVCLComponentFactory Alphabetically Legend

In TCorbaVCLComponentFactory

- ComponentClass

Derived from TCorbaFactory

- InstanceName
  - Instancing
  - InterfaceName
  - RepositoryID
  - ThreadModel

Derived from TInterfacedObject

- RefCount

TCorbaVCLComponentFactory properties

- ComponentClass
  - InstanceName
  - Instancing
  - InterfaceName
  - RefCount
  - RepositoryID
  - ThreadModel

TCorbaVCLComponentFactory.ComponentClass

Indicates the TCorbaDataModule descendant that the factory creates.

```
type TComponentClass = class of TComponent;
property ComponentClass: TComponentClass;
```

Description

Read ComponentClass to determine the implementation class associated with the CORBA factory object. This class is a descendant of TCorbaDataModule. In response to client requests, the CORBA VCL component factory creates or locates an object of this class.

TCorbaVCLComponentFactory methods

TCorbaVCLComponentFactory Alphabetically

In TCorbaVCLComponentFactory

Create

Derived from TCorbaFactory

Destroy

Derived from TInterfacedObject

BeforeDestruction

Derived from TObject

AfterConstruction
ClassInfo
ClassName
ClassNameIs
ClassParent
ClassType
CleanupInstance
DefaultHandler
Dispatch
FieldAddress
Free
FreeInstance
GetInterface
GetInterfaceEntry
GetInterfaceTable
InheritsFrom
InitInstance
InstanceSize
MethodAddress
MethodName
NewInstance
SafeCallException

TCorbaVCLComponentFactory methods

TCorbaVCLComponentFactory By object

- AfterConstruction
- BeforeDestruction
- ClassInfo
- ClassName
- ClassNameIs
- ClassParent
- ClassType
- CleanupInstance
- Create
- DefaultHandler
- Destroy
- Dispatch
- FieldAddress
- Free
- FreeInstance
- GetInterface
- GetInterfaceEntry
- GetInterfaceTable
- InheritsFrom
- InitInstance
- InstanceSize
- MethodAddress
- MethodName
- NewInstance
- SafeCallException

TCorbaVCLComponentFactory.Create

Creates an instance of TCorbaVCLComponentFactory associated with a specified server interface.

```
type
  TComponentClass = class of TComponent;
  TCORBAInstancing = (iSingleInstance, iMultiInstance);
  TCorbaThreadModel = (tmMultiThreaded, tmSingleThread);
constructor Create(const InterfaceName, InstanceName, RepositoryId: string;
  const ImplGUID: TGUID; AComponentClass: TComponentClass; Instancing:
  TCorbaInstancing = iMultiInstance; ThreadModel: TCorbaThreadModel =
  tmSingleThread);
```

Description

The CORBA Data Module Wizard automatically adds a call to the constructor for TCorbaVCLComponentFactory in the initialization section of the CORBA data module's unit.

The InterfaceName parameter specifies the name of the factory's interface (not the name of the data module's interface).

The InstanceName parameter specifies an instance identifier that is used by client applications to locate a specific factory instance.

The RepositoryId parameter specifies the CORBA Repository ID of the factory. This string has the form 'IDL:Modulename/FactoryInterfaceName:1.0'.

The ImplGUID parameter specifies the interface type (or GUID) of the CORBA data module. This is a descendant of IDataBroker.

The AComponentClass parameter specifies the TCorbaDataModule descendant that implements the CORBA server's interface.

The Instancing parameter indicates whether the factory creates a single object instance that all CORBA clients share, or whether the factory creates a separate instance for each CORBA client.

The ThreadModel parameter indicates whether the ORB should serialize calls to a single factory instance or whether multiple clients can call the same factory simultaneously on multiple threads.

Accessibility

- Read-only

Hierarchy

TObject

TInterfacedObject

TCorbaFactory

ECorbaDispatch

Hierarchy Properties Methods

ECorbaDispatch is the exception class that is raised when a CORBA application encounters a problem using DII.

Unit

provider

Description

ECorbaDispatch is raised when a problem occurs with a late-bound call to a CORBA server object. Such problems include

- Calling interfaces that are not registered with a running Interface Repository.
- Calling an interface with the wrong number of parameters or parameters that can't be converted to the expected type.
- Attempting to call a CORBA server using a variant that does not hold a proper CORBA interface.

ECorbaDispatch properties

ECorbaDispatch Alphabetically

Derived from Exception

- HelpContext
- Message

ECorbaDispatch properties

ECorbaDispatch By object

- HelpContext
- Message

ECorbaDispatch methods

ECorbaDispatch Alphabetically

Derived from Exception

- Create
- CreateFmt
- CreateFmtHelp
- CreateHelp
- CreateRes
- CreateResFmt
- CreateResFmtHelp
- CreateResHelp

Derived from TObject

- AfterConstruction
- BeforeDestruction
- ClassInfo
- ClassName
- ClassNameIs
- ClassParent
- ClassType
- CleanupInstance
- DefaultHandler
- Destroy
- Dispatch
- FieldAddress
- Free
- FreeInstance
- GetInterface
- GetInterfaceEntry
- GetInterfaceTable
- InheritsFrom
- InitInstance
- InstanceSize
- MethodAddress
- MethodName
- NewInstance
- SafeCallException

ECorbaDispatch methods

ECorbaDispatch By object

- AfterConstruction
- BeforeDestruction
- ClassInfo
- ClassName
- ClassNameIs
- ClassParent
- ClassType
- CleanupInstance
- Create
- CreateFmt
- CreateFmtHelp
- CreateHelp
- CreateRes
- CreateResFmt
- CreateResFmtHelp
- CreateResHelp
- DefaultHandler
- Destroy
- Dispatch
- FieldAddress
- Free
- FreeInstance
- GetInterface
- GetInterfaceEntry
- GetInterfaceTable
- InheritsFrom
- InitInstance
- InstanceSize
- MethodAddress
- MethodName
- NewInstance
- SafeCallException

Hierarchy

TObject

Exception

ECorbaException

Hierarchy Properties Methods See also

ECorbaException is the exception class that is raised when a CORBA application can't perform a requested action.

Unit

CorbaObj

Description

ECorbaException is raised when an application tries to bind to an unregistered interface or create an object instance with an ill-formed CORBA factory. In addition, ECorbaException is the base class for ECorbaUserException, which represents application-specific CORBA exceptions.

ECorbaException properties

ECorbaException Alphabetically Legend

In ECorbaException

- Name

Derived from Exception

HelpContext

Message

ECorbaException properties

ECorbaException By object Legend

- HelpContext
- Message
- Name

ECorbaException.Name

ECorbaException See also

Indicates the type of exception that occurred.

```
property Name: string;
```

Description

Name is a string that identifies the CORBA exception. When the CORBA exception is a user-defined exception (ECORBAUserException), the value of Name is assigned by the constructor. For other exceptions raised by the ORB or generated by the CorbaInterfaceIDManager or CorbaFactory, Name is one of the values from the following table:

Value	Meaning
CORBA interface not registered	The association between a requested interface and its repository ID is not registered with the global CorbaInterfaceIDManager variable.
CORBA Repository ID <id> not registered	The association between a requested repository ID and the interface type it represents is not registered with the global CorbaInterfaceIDManager variable.
CORBA Factory did not implement CreateInterface.	The TCorbaFactory descendant on the server did not implement a CreateInterface method. This method is automatically created by the CORBA object wizards.
NO_IMPLEMENT	The ORB could not bind to the requested interface. Usually this is because the server is down.
BAD_CONTEXT	The context information (which describes the client application) could not be interpreted.
BAD_INV_ORDER	Calls to the ORB software were made out of order.
BAD_OPERATION	The client attempted an invalid operation.
BAD_PARAM	An invalid parameter was passed.
BAD_TYPECODE	The client sent an Invalid typecode when making a DII call.
COMM_FAILURE	Network communication failure
DATA_CONVERSION	Data conversion error.
FREE_MEM	Unable to free memory.
IMP_LIMIT	Implementation limit violated.
INITIALIZE	The ORB failed to initialize.
INTERNAL	ORB internal error.
INTF_REPOS	Error accessing the interface repository. Usually this is because the repository is not running.
INV_FLAG	An invalid flag was passed to the ORB.
INV_INDENT	Invalid identifier syntax.
INV_OBJREF	An invalid object reference was specified.
MARSHAL	Error on the server marshaling parameter or result.
NO_MEMORY	Dynamic memory allocation failure
NO_PERMISSION	The client attempted an operation for which it has insufficient rights.

NO_RESOURCES	Insufficient resources to process request
NO_RESPONSE	Response to request is not yet available
OBJ_ADAPTOR	The BOA detected an error.
OBJECT_NOT_EXIST	The requested object is not available.
PERSIST_STORE	Persistent storage failure
TRANSIENT	Transient failure
UNKNOWN	Unknown exception

ECorbaException methods

ECorbaException Alphabetically

Derived from Exception

- Create
- CreateFmt
- CreateFmtHelp
- CreateHelp
- CreateRes
- CreateResFmt
- CreateResFmtHelp
- CreateResHelp

Derived from TObject

- AfterConstruction
- BeforeDestruction
- ClassInfo
- ClassName
- ClassNameIs
- ClassParent
- ClassType
- CleanupInstance
- DefaultHandler
- Destroy
- Dispatch
- FieldAddress
- Free
- FreeInstance
- GetInterface
- GetInterfaceEntry
- GetInterfaceTable
- InheritsFrom
- InitInstance
- InstanceSize
- MethodAddress
- MethodName
- NewInstance
- SafeCallException

ECorbaException methods

ECorbaException By object

- AfterConstruction
- BeforeDestruction
- ClassInfo
- ClassName
- ClassNameIs
- ClassParent
- ClassType
- CleanupInstance
- Create
- CreateFmt
- CreateFmtHelp
- CreateHelp
- CreateRes
- CreateResFmt
- CreateResFmtHelp
- CreateResHelp
- DefaultHandler
- Destroy
- Dispatch
- FieldAddress
- Free
- FreeInstance
- GetInterface
- GetInterfaceEntry
- GetInterfaceTable
- InheritsFrom
- InitInstance
- InstanceSize
- MethodAddress
- MethodName
- NewInstance
- SafeCallException

Accessibility

- Read-only

Hierarchy

TObject

Exception

ECorbaUserException

Hierarchy Properties Methods See also

ECorbaUserException represents exceptions that are defined by the CORBA application developer.

Unit

CorbaObj

Description

Use ECorbaUserException as a base class for exceptions that are defined as part of a CORBA interface. CORBA user exceptions represent exceptions that are defined as part of a CORBA interface.

To use an ECorbaUserException descendant, the exception must be registered with the ORB, using the global RegisterUserException function. When registering the exception, provide the ORB with a function that returns the value of an exception's Proxy property. When the ORB needs to raise the ECorbaUserException, it calls this function, which should either create a new exception object and return its Proxy or simply return the Proxy of a global instance of the exception. When the function can no longer create user exception objects, call UnRegisterUserException so that the ORB doesn't try to raise the exception.

ECorbaUserException properties

ECorbaUserException Alphabetically Legend

In ECorbaUserException

- Proxy

Derived from ECorbaException

- Name

Derived from Exception

HelpContext

Message

ECorbaUserException properties

ECorbaUserException By object Legend

- HelpContext
- Message
- Name
  - Proxy

ECorbaUserException.Proxy

ECorbaUserException See also

Represents the exception instance to the ORB.

```
PUserExceptionProxy = type Pointer;
property Proxy: PUserExceptionProxy ;
```

Description

Use Proxy as the return value of a function that generates or accesses an ECorbaUserException instance. This function is passed as the Factory parameter when calling the global RegisterUerException function. When the ORB needs to raise the user-defined exception, it calls this function to obtain an instance of the exception.

Proxy enables the ORB to call the Copy method to copy the exception or the Throw method to raise the exception.

ECorbaUserException methods

ECorbaUserException Alphabetically

In ECorbaUserException

- Copy
- Create
- Throw

Derived from Exception

- CreateFmt
- CreateFmtHelp
- CreateHelp
- CreateRes
- CreateResFmt
- CreateResFmtHelp
- CreateResHelp

Derived from TObject

- AfterConstruction
- BeforeDestruction
- ClassInfo
- ClassName
- ClassNameIs
- ClassParent
- ClassType
- CleanupInstance
- DefaultHandler
- Destroy
- Dispatch
- FieldAddress
- Free
- FreeInstance
- GetInterface
- GetInterfaceEntry
- GetInterfaceTable
- InheritsFrom
- InitInstance
- InstanceSize
- MethodAddress
- MethodName
- NewInstance
- SafeCallException

ECorbaUserException methods

ECorbaUserException By object

- AfterConstruction
- BeforeDestruction
- ClassInfo
- ClassName
- ClassNameIs
- ClassParent
- ClassType
- CleanupInstance
- Copy
- Create
- CreateFmt
- CreateFmtHelp
- CreateHelp
- CreateRes
- CreateResFmt
- CreateResFmtHelp
- CreateResHelp
- DefaultHandler
- Destroy
- Dispatch
- FieldAddress
- Free
- FreeInstance
- GetInterface
- GetInterfaceEntry
- GetInterfaceTable
- InheritsFrom
- InitInstance
- InstanceSize
- MethodAddress
- MethodName
- NewInstance
- SafeCallException
- Throw

ECorbaUserException.Copy

ECorbaUserException See also

Copies the exception's properties from a CORBA marshaling buffer.

```
procedure Copy(const InBuf: IMarshalInBuffer); virtual; abstract;
```

Description

Override Copy when deriving an exception class from ECorbaUserException. The implementation of Copy must read the properties of the exception, as defined in the CORBA interface, and use them to set the properties of the ECorbaUserException instance.

Do not call the Copy method. This method is used internally to generate the value of the Proxy property. The ORB uses the Proxy property to call Copy when it instantiates an instance of a user-defined property.

ECorbaUserException.Create

ECorbaUserException See also

Creates and initializes an instance of ECorbaUserException.

```
constructor Create(const Name: string);
```

Description

Call Create to instantiate a user-defined CORBA exception. Typically, these exceptions are created in a function that is used as a parameter to the global RegisterUserException function.

Create assigns the Name parameter to the exception's Name property. It then uses the Throw method and the Copy method to generate a proxy object, which it assigns as the value of the Proxy property.

Note: Do not create instances of ECorbaUserException. The Copy method of ECorbaUserException is abstract and should not be used. Instead, create instances of an ECorbaUserException descendant that has defined an implementation of the Copy method.

ECorbaUserException.Throw

ECorbaUserException See also

Raises the instance of ECorbaUserException.

```
procedure Throw;
```

Description

The ORB uses the Throw method to raise ECorbaUserException. Because the ORB can't directly raise exceptions in Object Pascal, it requires the Throw method as an extra layer of indirection. When the ORB needs to raise an exception, it uses the Proxy property to access the Throw method, thereby raising the exception.

Accessibility

- Read-only

Hierarchy

TObject

Exception

ECorbaException

TAny type

See also

Represents a CORBA interface or a parameter in a CORBA interface.

Unit

CorbaObj

```
type TAny = Variant;
```

Description

Use TAny when writing CORBA clients that use the dynamic invocation interface (DII). When a TAny variable holds a CORBA interface, calling the interface methods on that variable causes a DII call which looks up the interface in the Interface Repository and invokes the server accordingly.

The parameters to any DII interface call must also be TAny values: DII uses late binding, so parameter values must carry their type information with them.

TCKind type

See also

Designates a native CORBA type.

Unit

Orbpas

```
type TCKind = (tk_null, tk_void, tk_short, tk_long, tk_ushort, tk_ulong,
  tk_float, tk_double, tk_boolean, tk_char, tk_octet, tk_any, tk_TypeCode,
  tk_Principal, tk_objref, tk_struct, tk_union, tk_enum, tk_string,
  tk_sequence, tk_array, tk_alias, tk_except, tk_longlong, tk_ulonglong,
  tk_longdouble, tk_wchar, tk_wstring, tk_fixed);
```

Description

Use TCKind to indicate a CORBA type. TCKind values indicate the types of elements in CORBA arrays, sequences, and structures. TCKind values can also be converted into an ITypeCode interface using the ORB's MakeTypeCode method.

The following table describes the possible values:

Value	Meaning
tk_null	a null (empty) type
tk_void	no type (such as an empty parameter list)
tk_short	a short (16-bit) signed integer
tk_long	a long (32-bit) signed integer
tk_ushort	a short (16-bit) unsigned integer
tk_ulong	a long (32-bit) unsigned integer
tk_float	Single (32-bit floating point value)
tk_double	Double (64-bit floating point value)
tk_boolean	Boolean
tk_char	Char
tk_octet	8-bit quantity guaranteed not to undergo conversion during transmission.
tk_any	TAny (a CORBA Variant)
tk_TypeCode	a type code
tk_Principal	TCorbaPrincipal (array of bytes)
tk_objref	CORBA object reference (ICorbaObject)
tk_struct	CORBA structure (a record)
tk_union	CORBA union (a record containing a case statement)
tk_enum	enumerated type
tk_string	sequence (dynamic array) of Char
tk_sequence	dynamic array
tk_array	fixed size array
tk_alias	a CORBA alias
tk_except	an exception
tk_longlong	signed 64-bit integer
tk_ulonglong	unsigned 64-bit integer
tk_longdouble	80-bit floating-point type

tk_wchar	WideChar
tk_wstring	WideString
tk_fixed	const

ITypeCode type

See also

ITypeCode.is an interface that describes a CORBA type.

Unit

orbpas

```
type ITypeCode = interface;
```

Description

Use ITypeCode values to create TAny values that represent complex CORBA types such as structures (records), arrays (fixed arrays) and sequences (dynamic arrays). ITypeCode is an interface to the Interface Repository that allows an ORB to access type information about an interface type.

ICorbaObject

Hierarchy Methods See also

ICorbaObject represents the interface of an object accessed through the ORB.

Unit

corbaobj

Description

TCorbaStub objects implement the ICorbaObject interface. This interface lets client applications obtain information about the server object and send custom information to the server as a TCorbaPrincipal.

ICorbaObject methods

ICorbaObject Alphabetically

In ICorbaObject

- Hash
- IsA
- NonExistent
- SetPrincipal

Derived from IUnknown

- _AddRef
- _Release
- QueryInterface

ICorbaObject methods

ICorbaObject By object

- _AddRef
- _Release
- Hash
- IsA
- NonExistent
- QueryInterface
- SetPrincipal

ICorbaObject.Hash

Returns a hash value for the server object instance.

```
function Hash(Maximum: Integer): Integer;
```

Description

Call Hash to obtain a hash value for the CORBA object instance. This value is not guaranteed to be unique, but will remain consistent through the lifetime of the object reference.

ICorbaObject.IsA

Indicates whether the CORBA stub represents a particular type.

```
function IsA(const LogicalTypeId: string): Boolean;
```

Description

Call IsA to check whether the associated server object is of a specified type. Specify the type by passing its base name as the LogicalTypeId parameter This is the name specified in the object's IDL specification. IsA returns True if the object on the CORBA server is an instance of the specified type or one of its descendants. IsA returns False otherwise.

ICorbaObject.NonExistent

Indicates whether the server object has been deactivated.

```
function NonExistent: Boolean;
```

Description

Call NonExistent to test whether the object on the server is still instantiated. NonExistent returns False if the server object is still available. NonExistent returns True if the server instance has been freed.

ICorbaObject.SetPrincipal

Sends an array of bytes to the server

```
TCorbaPrincipal = array of Byte;
procedure SetPrincipal(const Principal: TCorbaPrincipal);
```

Description

Call SetPrincipal to send an arbitrary value to the server application. Once the client calls SetPrincipal, the server application can read this value by calling the Boa's GetPrincipal method. SetPrincipal and GetPrincipal allow CORBA clients to pass custom state information to servers, or to pass information that identifies the client.

Note: To obtain a TCorbaPrincipal for the Principal parameter, use the global MakePrincipal function.

Hierarchy

IUnknown

ICorbaObj

Hierarchy Methods See also

ICorbaObj is the interface that represents a CORBA object.

Unit

orbpas

Description

Applications do not use ICorbaObj. Instead, they use a descendant interface when communicating with a CORBA object reference.

ICorbaObj is an interface. As such, it can't be instantiated. Instead, an ICorbaObj interface is obtained from the ORB to allow applications to talk to a CORBA object reference.

ICorbaObj methods

ICorbaObj Alphabetically

In ICorbaObj

- CorbaObject
- IsLocal

Derived from IUnknown

- _AddRef
- _Release
- QueryInterface

ICorbaObj methods

ICorbaObj By object

_AddRef
_Release
CorbaObject
IsLocal
QueryInterface

ICorbaObj.CorbaObject

Points to the ORB's representation of the CORBA object instance.

```
type
  PCorbaObject = type Pointer;
function CorbaObject: PCorbaObject; stdcall;
```

Description

Applications should not need to use the CorbaObject method. It is used internally.

ICorbaObj.IsLocal

Indicates whether the CORBA object this interface represents is instantiated locally.

```
function IsLocal: CorbaBoolean; stdcall;
```

Description

Call IsLocal to determine whether the object instance that implements ICorbaObj is local or remote. If IsLocal returns a nonzero value, the instance is local. This allows the application to optimize interface calls because they do not need to be sent across the network.

Hierarchy

IUnknown

IMarshalInBuffer

Hierarchy Methods See also Example

IMarshalInBuffer is the interface for reading return values from a marshaling buffer.

Unit

orbpas

Description

Use IMarshalInBuffer when implementing a custom CORBA stub object. Stub objects can obtain an IMarshalInBuffer interface by invoking methods using their IStub interface.

Read output parameters from the marshaling buffer in the order they appear in the interface method. If the interface method is a function call, the return value of the function call precedes any output parameters.

IMarshalInBuffer methods

IMarshalInBuffer Alphabetically

In IMarshalInBuffer

- Buffer
- GetAny
- GetChar
- GetDouble
- GetFloat
- GetLong
- GetObject
- GetShort
- GetText
- GetUnsignedChar
- GetUnsignedLong
- GetUnsignedShort
- GetWidechar
- GetWideText

Derived from IUnknown

- _AddRef
- _Release
- QueryInterface

IMarshalInBuffer methods

IMarshalInBuffer By object

_AddRef
_Release
Buffer
GetAny
GetChar
GetDouble
GetFloat
GetLong
GetObject
GetShort
GetText
GetUnsignedChar
GetUnsignedLong
GetUnsignedShort
GetWidechar
GetWideText
QueryInterface

IMarshalInBuffer.Buffer

IMarshalInBuffer

Returns the marshaling buffer that holds return values.

```
type
  PMarshalInbuffer = type Pointer;
function Buffer: PMarshalInbuffer; stdcall;
```

Description

Do not use the buffer function. Instead, use the various Getxxx methods of the IMarshalInBuffer interface to read from the marshaling buffer.

IMarshalInBuffer.GetAny

IMarshalInBuffer See also

Reads a CORBA Any value from the marshaling buffer.

```
type
  PCorbaAny = type Pointer;
function GetAny: PCorbaAny; stdcall;
```

Description

Do not use GetAny to read a value of the CORBA Any type. Instead, use the global UnmarshalAny function, which retrieves the value and converts it to a more usable Variant type.

IMarshalInBuffer.GetChar

IMarshalInBuffer See also

Reads a char value from the marshaling buffer.

```
function GetChar: Shortint; stdcall;
```

Description

Call GetChar to read a char value. GetChar returns a 16-bit integer which can then be cast to a Char value.

IMarshalInBuffer.GetDouble

IMarshalInBuffer See also

Reads a Double value from the marshaling buffer.

```
function GetDouble: Double; stdcall;
```

Description

Call GetDouble to read a Double value.

IMarshalInBuffer.GetFloat

IMarshalInBuffer See also

Reads a Single value from the marshaling buffer.

```
function GetFloat: Single; stdcall;
```

Description

Call GetFloat to read the value of a Single. These values are called float values in the CORBA IDL.

IMarshalInBuffer.GetLong

IMarshalInBuffer See also Example

Reads a 32-bit integer value from the marshaling buffer.

```
function GetLong: Integer; stdcall;
```

Description

Call GetLong to read the value of a 32-bit integer from the marshaling buffer.

IMarshalInBuffer.GetObject

IMarshalInBuffer See also

Reads an object interface from the marshaling buffer.

```
procedure GetObject(out Obj: ICorbaObj; ref_cnt: CorbaBoolean = True);
  stdcall;
```

Description

Do not call GetObject to read an object reference from the marshaling buffer. Use the global UnMarshalObject function instead. UnMarshalObject handles the details of creating a stub object for the interface that was returned in the marshaling buffer.

An interface for the object is returned in Obj.

ref_cnt indicates whether to increment the reference count for this object.

IMarshalInBuffer.GetShort

IMarshalInBuffer See also

Reads a 16-bit integer from the marshaling buffer.

```
function GetShort: Smallint; stdcall;
```

Description

Call GetShort to read a 16-bit integer from the marshaling buffer.

IMarshalInBuffer.GetText

IMarshalInBuffer See also

Reads a string from the marshaling buffer.

```
function GetText: PChar; stdcall;
```

Description

Do not call GetText to read a string from the marshaling buffer. Instead, call UnmarshalText which returns a string value and frees the memory pointed to by the GetText return value.

IMarshalInBuffer.GetUnsignedChar

IMarshalInBuffer See also

Reads a Byte value from the marshaling buffer.

```
function GetUnsignedChar: Byte; stdcall;
```

Description

Call GetUnsignedChar to read a Byte value from the marshaling buffer.

IMarshalInBuffer.GetUnsignedLong

IMarshalInBuffer See also

Reads an unsigned integer value from the marshaling buffer.

```
function GetUnsignedLong: UINT; stdcall;
```

Description

Call GetUnsignedLong to read a UINT value from the marshaling buffer.

IMarshalInBuffer.GetUnsignedShort

IMarshalInBuffer See also

Reads an unsigned 16-bit value from the marshaling buffer.

```
function GetUnsignedShort: Word; stdcall;
```

Description

Call GetUnsignedShort to read a Word value from the marshaling buffer.

IMarshalInBuffer.GetWideChar

IMarshalInBuffer See also

Reads a single UNICODE character from the marshaling buffer.

```
function GetWidechar: Word; stdcall;
```

Description

Call GetWidechar to read a UNICODE character from the marshaling buffer. The value is returned as a Word type.

IMarshalInBuffer.GetWideText

IMarshalInBuffer See also

Reads a wide string from the marshaling buffer.

```
function GetWideText: PWideChar; stdcall;
```

Description

Do not call GetWideText to read a UNICODE string from the marshaling buffer. Instead, use the global UnmarshalWideText function that returns a more usable WideString type and frees the memory in the buffer returned by GetWideText.

Hierarchy

IUnknown

IMarshalOutBuffer

Hierarchy Methods See also Example

IMarshalOutBuffer is the interface for writing arguments to a marshaling Buffer.

Unit

orbpas

Description

Use IMarshalOutBuffer when implementing a custom CORBA stub object. Stub objects can obtain an IMarshalOutBuffer interface from their IStub interface. After using the methods of the IMarshalOutBuffer interface to write parameter values to the marshaling Buffer, the stub object invokes the interface call.

Write arguments to the marshaling Buffer in the order they appear in the interface method.

IMarshalOutBuffer methods

IMarshalOutBuffer Alphabetically

In IMarshalOutBuffer

- Buffer
- PutAny
- PutChar
- PutDouble
- PutFloat
- PutLong
- PutObject
- PutShort
- PutText
- PutUnsignedChar
- PutUnsignedLong
- PutUnsignedShort
- PutWidechar
- PutWideText

Derived from IUnknown

- _AddRef
- _Release
- QueryInterface

IMarshalOutBuffer methods

IMarshalOutBuffer By object

_AddRef
_Release
Buffer
PutAny
PutChar
PutDouble
PutFloat
PutLong
PutObject
PutShort
PutText
PutUnsignedChar
PutUnsignedLong
PutUnsignedShort
PutWidechar
PutWideText
QueryInterface

IMarshalOutBuffer.Buffer

Returns the marshaling buffer that receives arguments.

```
type
  PMarshalOutbuffer = type Pointer;
function Buffer: PMarshalOutbuffer; stdcall;
```

Description

Do not use the Buffer function. Instead, use the various Putxxx methods of the IMarshalOutBuffer interface to write to the marshaling buffer.

IMarshalOutBuffer.PutAny

Adds a CORBA Any value to the marshaling buffer.

```
type
  PCorbaAny = type Pointer;
procedure PutAny(Value: PCorbaAny); stdcall;
```

Description

Do not use PutAny to add a Variant value to the marshaling buffer. Instead, use the global MarshalAny procedure, which converts a Variant value to a CORBA Any and adds it to the marshaling buffer.

IMarshalOutBuffer.PutChar

Adds a char value to the marshaling Buffer.

```
procedure PutChar(Value: Shortint); stdcall;
```

Description

Call PutChar to add the value of a char parameter. The character must first be converted to a Shortint before it can be put in the marshaling Buffer.

IMarshalOutBuffer.PutDouble

Adds a Double value to the marshaling Buffer.

```
procedure PutDouble(Value: Double); stdcall;
```

Description

Call PutDouble to add the value of a Double parameter.

IMarshalOutBuffer.PutFloat

Adds a Single value to the marshaling Buffer.

```
procedure PutFloat(Value: Single); stdcall;
```

Description

Call PutFloat to add the value of a Single parameter. These values are called float values in the CORBA IDL.

IMarshalOutBuffer.PutLong

Adds a 32-bit integer value to the marshaling Buffer.

```
procedure PutLong(Value: Integer); stdcall;
```

Description

Call PutLong to add the value of a 32-bit integer parameter.

IMarshalOutBuffer.PutObject

IMarshalOutBuffer Methods See also

Adds an object interface to the marshaling Buffer.

```
procedure PutObject(const Value: ICorbaObj); stdcall;
```

Description

Do not use PutObject to add an object reference to the marshaling Buffer. Instead, use the global MarshalObject procedure.

The Value parameter is an ICorbaObj interface for the object reference that is added to the marshaling Buffer. This interface can be obtained from a CORBA stub or skeleton object.

IMarshalOutBuffer.PutShort

Adds a 16-bit integer to the marshaling Buffer.

```
procedure PutShort(const Value: Smallint); stdcall;
```

Description

Call PutShort to pass a 16-bit integer as a parameter.

IMarshalOutBuffer.PutText

Adds a string to the marshaling Buffer.

```
procedure PutText(const Value: PChar); stdcall;
```

Description

Call PutText to pass a string as a parameter. First cast the string to a PChar to add it to the marshaling Buffer.

IMarshalOutBuffer.PutUnsignedChar

Adds a Byte value to the marshaling Buffer.

```
procedure PutUnsignedChar(const Value: Byte); stdcall;
```

Description

Call PutUnsignedChar to pass a Byte value as a parameter.

IMarshalOutBuffer.PutUnsignedLong

Adds an unsigned integer value to the marshaling Buffer.

```
procedure PutUnsignedLong(const Value: UINT); stdcall;
```

Description

Call PutUnsignedLong to pass a UINT value as a parameter.

IMarshalOutBuffer.PutUnsignedShort

Adds an unsigned 16-bit value to the marshaling Buffer.

```
procedure PutUnsignedShort(const Value: Word); stdcall;
```

Description

Call PutUnsignedShort to pass a Word value as a parameter.

IMarshalOutBuffer.PutWidechar

Adds a single UNICODE character to the marshaling Buffer.

```
procedure PutUnsignedShort(const Value: Word); stdcall;
```

Description

Call PutWidechar to pass a UNICODE character as a parameter.

IMarshalOutBuffer.PutWideText

Adds a wide string to the marshaling Buffer.

```
procedure PutWideText(const Value: PWideChar); stdcall;
```

Description

Call PutWideText to pass a UNICODE string as a parameter. First cast the string to a PWideChar in order to add it to the marshaling Buffer.

Hierarchy

IUnknown

ISkeleton

Hierarchy Methods See also

ISkeleton allows a CORBA skeleton to access its implementation class or obtain a marshaling buffer for return values.

Unit

orbpas

Description

TCorbaSkeleton descendants use ISkeleton when talking to the ORB. This allows them to access their CORBA object reference.

Note: To communicate in the other direction, the ORB talks to TCorbaSkeleton descendants using the ISkeletonObject interface.

ISkeleton methods

ISkeleton Alphabetically

In ISkeleton

- GetImplementation
- GetReplyBuffer

Derived from ICorbaObj

- CorbaObject
- IsLocal

Derived from IUnknown

- _AddRef
- _Release
- QueryInterface

ISkeleton methods

ISkeleton By object

_AddRef
_Release
CorbaObject
GetImplementation
GetReplyBuffer
IsLocal
QueryInterface

ISkeleton.GetImplementation

Returns an interface for the CORBA object implementation.

```
procedure GetImplementation(out impl: IUnknown); stdcall;
```

Description

The GetImplementation method returns the interface for a server object implementation. This method is used when unmarshaling objects from a buffer.

ISkeleton.GetReplyBuffer

ISkeleton Methods

Returns a marshaling buffer that can hold return values.

```
procedure GetReplyBuffer(cookie: Pointer; out Outbuf: IMarshalOutBuffer);
  stdcall;
```

Description

Skeleton objects call GetReplyBuffer to obtain a marshaling buffer for adding return values that can be sent to the client identified by the cookie parameter.

Hierarchy

IUnknown

ICorbaObj

ISkeletonObject

Hierarchy Methods See also

ISkeletonObject is the interface that represents a CORBA skeleton.

Unit

orbpas

Description

TCorbaSkeleton objects implement the ISkeletonObject interface. This interface allows the ORB to pass interface calls to the skeleton, which handles marshaling details and invokes the implementation class.

Note: Communication in the other direction (where the skeleton talks to the ORB) is handled using the ISkeleton interface.

ISkeletonObject methods

ISkeletonObject Alphabetically

In ISkeletonObject

- Execute
- GetImplementation
- GetSkeleton

Derived from IUnknown

- _AddRef
- _Release
- QueryInterface

ISkeletonObject methods

ISkeletonObject By object

_AddRef
_Release
execute
GetImplementation
GetSkeleton
QueryInterface

ISkeletonObject.Execute

Executes a specified interface call.

```
function Execute(Operation: PChar; const Strm: IMarshalInBuffer; Cookie:
  Pointer): CorbaBoolean; stdcall;
```

Description

Applications do not call the Execute method. It is called automatically when the ORB passes in an interface invocation from a client application.

The Operation parameter specifies the interface member's name. The Strm parameter is a marshaling buffer that contains all the parameter values received from the client. The Cookie parameter specifies an internal identifier that allows the skeleton object to keep track of which marshaling buffers are associated with which interface calls.

The execute method returns 1 (True) if the call is successfully executed. It returns 0 (False) otherwise.

ISkeletonObject.GetImplementation

Returns an interface for the implementation object.

```
procedure GetImplementation(out Impl: IUnknown); virtual; stdcall;
```

Description

The GetImplemenation method returns an interface to the server implementation. This can be used to call the implementation after unmarshaling any parameters.

ISkeletonObject.GetSkeleton

ISkeletonObject Methods

Returns an ISkeleton interface for marshaling interface calls.

```
procedure GetSkeleton(out Skeleton: ISkeleton); stdcall;
```

Description

The GetSkeleton method obtains an ISkeleton interface for marshaling interface calls.

Hierarchy

IUnknown

IStub

Hierarchy Methods See also

IStub represents a remote server object for a CORBA stub.

Unit

orbpas

Description

The CORBA stub manager obtains a descendant of IStub when a client application wants to talk to a remote CORBA server object. Using the IStub interface, the CORBA stub manager creates a descendant of TCorbaStub. This CORBA stub object then uses the IStub interface to talk to the CORBA object reference.

Note: Communication in the other direction (where the ORB talks to the stub object) is handled using the IStubObject interface.

IStub methods

IStub Alphabetically

In IStub

- CreateRequest
- Dispatch
- GetInterface
- Hash
- Invoke
- IsA
- NonExistent
- RepositoryID
- SetPrincipal

Derived from ICorbaObj

- CorbaObject
- IsLocal

Derived from IUnknown

- _AddRef
- _Release
- QueryInterface

IStub methods

IStub By object

- _AddRef
- _Release
- CorbaObject
- CreateRequest
- Dispatch
- GetInterface
- Hash
- Invoke
- IsA
- IsLocal
- NonExistent
- QueryInterface
- RepositoryID
- SetPrincipal

IStub.CreateRequest

IStub Methods See also Example

Sets up an interface call and returns its marshaling buffer.

```
procedure CreateRequest(Operation: PChar; ResponseExpected: CorbaBoolean;
  out Outbuf: IMarshalOutBuffer); stdcall;
```

Description

Stub objects call CreateRequest to obtain a marshaling buffer into which they can write the arguments of an interface call.

The Operation parameter is a string that gives the name of the interface method. The ResponseExpected parameter indicates whether the method is returns any values (out parameters or a function return value). The Outbuf parameter returns a marshaling buffer into which the stub can write parameter values.

IStub.Dispatch

IStub Methods See also

Makes a late-bound (DII) call to a remote CORBA object.

```
function Dispatch(CallDesc: PCallDesc; Params: Pointer; out Result:
  Variant): Integer; stdcall;
```

Description

When client applications use the Dynamic Invocation Interface (DII) to call remote CORBA objects, the generic stub object calls Dispatch to send the call to the CORBA server after it has parsed the interface call. Applications do not use this method directly.

The CallDesc parameter is an internal representation of the interface call that is recognized by the ORB. The Params parameter contains all parameter values, after they have been converted to the CORBA Any type. The Result parameter returns any return values from the server.

Dispatch returns an error code if the interface call can’t be completed. It returns 0 if the call is successful.

IStub.GetInterface

Returns a pointer to the interface definition from the Interface Repository.

```
type
  PCorbaInterfaceDef = type Pointer;
function GetInterface: PCorbaInterfaceDef; stdcall;
```

Description

If the server object is registered with the Interface Repository, GetInterface returns a structure containing information about its interface. Otherwise, GetInterface returns NULL.

Applications do not need to use the interface definition returned by GetInterface. It is used internally to create DII calls at runtime for client applications that do not use static binding.

IStub.Hash

Returns a hash value for the server object instance.

```
function Hash(Maximum: CorbaULong): CorbaULong; stdcall;
```

Description

CORBA stub objects call Hash to implement their Hash method. Hash returns a value for the CORBA object instance. This value is not guaranteed to be unique, but will remain consistent through the lifetime of the object reference.

IStub.IsA

Indicates whether the CORBA stub represents a particular type.

```
function IsA(LogicalTypeId: PChar): CorbaBoolean; stdcall;
```

Description

Call IsA to check whether the associated server object is of a specified type. Specify the type by passing its base name as the LogicalTypeId parameter This is the name specified in the object's IDL specification. IsA returns True if the object on the CORBA server is an instance of the specified type or one of its descendants. IsA returns False otherwise.

IStub.NonExistent

IStub Methods See also

Indicates whether the server object has been destroyed.

```
function NonExistent: CorbaBoolean; stdcall;
```

Description

Call NonExistent to test whether the object on the server is still instantiated. NonExistent returns False if the server object is still available. NonExistent returns True if the server instance has been freed.

IStub.RepositoryID

Returns the Repository ID of the associated CORBA object.

```
function RepositoryID: PChar; stdcall;
```

Description

Call RepositoryID to obtain the Repository ID of the CORBA object accessed using this IStub interface.

IStub.SetPrincipal

Sends information about the client to the server application.

```
procedure SetPrincipal(Bytes: Pointer; Length: CorbaULong); stdcall;
```

Description

Call SetPrincipal to send an arbitrary value to the server application. Once the client calls SetPrincipal, the server application can read this value by calling the Boa's GetPrincipal method. SetPrincipal and GetPrincipal allow CORBA clients to pass custom state information to servers, or to pass information that identifies the client.

The Bytes parameter is a pointer to an array of bytes, called a "principal". The Length parameter specifies the number of bytes in the array.

IStub.Invoke

IStub Methods See also Example

Sends an interface call to the remote server and obtains a result.

```
procedure Invoke(const Inbuf: IMarshalOutBuffer; out Outbuf:
  IMarshalInBuffer); stdcall;
```

Description

Call Invoke to send a remote interface call to the CORBA server. The Inbuf parameter specifies a marshaling buffer that contains the arguments for the method call. The Outbuf parameter receives any output parameters and the return value of function calls.

Note: Stub objects use Invoke to call an interface that is marshaled by the client application. Late-bound calls (that use DII) must use the Dispatch method instead.

Hierarchy

IUnknown

ICorbaObj

IStubObject

Hierarchy Methods See also

IStubObject is the interface that represents a CORBA stub.

Unit

orbpas

Description

TCorbaStub objects implement the IStubObject interface. This interface allows other objects to obtain the interface to the CORBA server object given an interface to the CORBA stub.

Because IStubObject is an interface, it can't be instantiated. Client applications can get IStubObject descendants by calling the global function CorbaFactoryCreateStub, the ORB's Bind method, or The CORBA stub manager's CreateStub method.

IStubObject methods

IStubObject Alphabetically

In IStubObject

GetStub

Derived from IUnknown

_AddRef

_Release

QueryInterface

IStubObject methods

IStubObject By object

_AddRef
_Release
GetStub
QueryInterface

IStubObject.GetStub

Returns the interface to an object on the CORBA server.

```
procedure GetStub(out stub :IStub); stdcall;
```

Description

Client applications do not call GetStub to obtain an interface for the CORBA server application. Instead, Component writers can use GetStub to obtain the server interface when implementing methods that marshal interface calls.

Hierarchy

IUnknown

TORB

Hierarchy Methods See also

TORB represents the CORBA Object Request Broker.

Unit

corbaobj

Description

Applications do not instantiate TORB objects. Instead, they use the global ORB function to obtain a global object that can communicate with CORBA

Use the methods of TORB in CORBA client applications to obtain interfaces to objects on a CORBA server.

TORB methods

TORB Alphabetically

In TORB

- Bind
- FindTypeCode
- Initialize
- MakeAlias
- MakeAliasTypeCode
- MakeArray
- MakeObjectRefTypeCode
- MakeSequence
- MakeSequenceTypeCode
- MakeStructure
- MakeStructureTypeCode
- MakeTypeCode
- ObjectToString
- Shutdown
- StringToObject

Derived from TObject

- AfterConstruction
- BeforeDestruction
- ClassInfo
- ClassName
- ClassNameIs
- ClassParent
- ClassType
- CleanupInstance
- Create
- DefaultHandler
- Destroy
- Dispatch
- FieldAddress
- Free
- FreeInstance
- GetInterface
- GetInterfaceEntry
- GetInterfaceTable
- InheritsFrom
- InitInstance
- InstanceSize
- MethodAddress
- MethodName
- NewInstance
- SafeCallException

TORB methods

TORB By object

- AfterConstruction
- BeforeDestruction
- Bind
- ClassInfo
- ClassName
- ClassNameIs
- ClassParent
- ClassType
- CleanupInstance
- Create
- DefaultHandler
- Destroy
- Dispatch
- FieldAddress
- FindTypeCode
- Free
- FreeInstance
- GetInterface
- GetInterfaceEntry
- GetInterfaceTable
- InheritsFrom
- Initialize
- InitInstance
- InstanceSize
- MakeAlias
- MakeAliasTypeCode
- MakeArray
- MakeObjectRefTypeCode
- MakeSequence
- MakeSequenceTypeCode
- MakeStructure
- MakeStructureTypeCode
- MakeTypeCode
- MethodAddress
- MethodName
- NewInstance
- ObjectToString
- SafeCallException
- Shutdown
- StringToObject

TORB.Bind

TORB See also Example

Makes an object instance available to CORBA clients.

```
function Bind(const RepositoryID: string; const ObjectName: string = '';
  const HostName: string = ''): IObject; overload;
function Bind(const InterfaceID: TGUID; const ObjectName: string = ''; const
  HostName: string = ''): IObject; overload;
```

Description

Call Bind to obtain an interface for calling a remote server object. Bind obtains an interface from the ORB and uses it to create a stub class that can marshal interface calls. It then returns an interface to this stub class.

Bind can be called in two ways: specifying the Repository ID of the server interface, or specifying its interface type. Use the first parameter to specify either the Repository ID or the Interface type (GUID). To call Bind using an interface type (GUID), the Interface type must be registered with the global CORBAInterfaceIDManager.

The last two parameters of Bind are optional. ObjectName specifies an instance name for the server object. HostName specifies a host name of a server machine for cases where you want to use a particular server.

Before the Bind method can obtain an interface and create a stub, the server interface type and Repository ID must be registered with the global CorbaInterfaceIDManager. In addition, if a stub class for the interface is not be registered with the global CorbaStubManager, Bind returns a generic stub interface. Generic stub interfaces can be used for calling servers using the dynamic invocation interface (DII), but can’t be cast to the interface type specified by the InterfaceID parameter.

TORB.FindTypeCode

TORB See also Example

Retrieves an interface that supplies type information for a CORBA interface.

```
function FindTypeCode(const RepositoryID: string): ITypeCode;
```

Description

Call FindTypeCode to obtain an ITypeCode value that can be used for creating complex TAny structures. These TAny structures are specialized variants that are used as parameters when making dynamic (late) binding calls with the Dynamic Invocation Interface (DII).

Pass the Interface Repository ID for the desired structure as the RepositoryID parameter. The ORB looks up the indicated type in the Interface Repository and returns an interface for it.

Note: The specified type must be registered with the Interface Repository or FindTypeCode returns NULL.

TORB.Initialize

TORB See also

Sends startup options to the ORB.

```
type TCommandLine = ORBPAS.TArgv;
class procedure Initialize; overload;
class procedure Initialize(const CommandLine: TCommandLine); overload;
```

Description

CORBA client applications call Initialize to indicate options such as the IP address and port number of the Smart Agent to be used. If Initialize is called with no arguments, the command line arguments to the CORBA client application are sent.

To create a set of options to send to the ORB, treat the CommandLine parameter like a dynamic array. Use code such as the following to set options:

```
SetLength(CommandLine, 2);
CommandLine[0] := '-ORBagentAddr 145.199.24.57';
CommandLine[1] := '-ORBagentPort 1457';
```

The following table summarizes the possible options:

Option	Purpose
ORBagentAddr	The host name or IP address of the host running the Smart Agent the client should use.
ORBagentPort	The port number of the Smart Agent.
ORBir_name	The name of the Interface Repository to use for DII.
ORBir_ior	The IOR of the Interface Repository to use for DII.
ORBrcvbufsize	The size of the buffer (in bytes) used to receive responses.
ORBsendbufsize	The size of the buffer (in bytes) used to send messages.
ORBshmsize	The size of the send and receive segments (in bytes) of shared memory.
ORBtcpNodelay	Indicates whether messages can be batched when buffers fill (0), or whether they must be sent immediately when the client makes a request.

TORB.MakeAlias

TORB See also

Creates an alias.

```
function MakeAlias(const RepositoryID, TypeName: string; Value, Test: TAny):
  TAny;
```

Description

Call MakeAlias to create an alias. The RepositoryID specifies the identifier of the resulting alias. The TypeName parameter names the type of the alias to be created. The Value parameter is the original for which the alias is created. The Test parameter is ignored.

MakeAlias returns the new alias as a TAny value.

TORB.MakeAliasTypeCode

TORB See also

Creates a type code interface for an alias.

```
function MakeAliasTypeCode(const RepositoryID, Name: string; const TC:
  ITypeCode): ITypeCode;
```

Description

Call MakeAliasTypeCode to get a type code interface for an alias. The RepositoryID specifies the alias. The Name parameter is the name of the alias's type. The TC parameter is the type code of the original for which the alias is created.

MakeAliasTypeCode returns a type code interface for the alias.

TORB.MakeArray

TORB See also

Returns a CORBA array built from a set of TAny values.

```
function MakeArray(Kind: TCKind; const Elements: array of TAny): TAny;
function MakeArray(TypeCode: ITypeCode; const Elements: array of TAny):
  TAny;
```

Description

Call MakeArray to create a TAny value for a CORBA array (a fixed-length array). This array can then be passed as a parameter to an interface using the dynamic invocation interface (DII).

The first parameter indicates the type of array elements. This can be either a TCKind value or an ITypeCode that describes these elements.

The Elements parameter is an array of TAny values that are assigned to the elements of the array.

TORB.MakeObjectRefTypeCode

TORB See also

Creates a type code interface for an object.

```
function MakeObjectRefTypeCode(const RepositoryID, Name: string): ITypeCode;
```

Description

Call MakeObjectRefTypeCode to get a type code interface for an object. The RepositoryID specifies the identifier for the object reference. The Name parameter is the name of the object's type. MakeObjectRefTypeCode returns the type code interface for the object.

TORB.MakeSequence

TORB See also

Returns a CORBA sequence built from a set of TAny values.

```
function MakeSequence(Kind: TCKind; const Elements: array of TAny): TAny;
function MakeSequence(TypeCode: ITypeCode; const Elements: array of TAny):
  TAny;
```

Description

Call MakeStructure to create a TAny value for a CORBA sequence (a dynamic array). This sequence can then be passed as a parameter to an interface using the dynamic invocation interface (DII).

The first parameter indicates the type of array elements. This can be either a TCKind value or an ITypeCode that describes these elements.

The Elements parameter is an array of TAny values that are assigned to the elements of the array.

TORB.MakeSequenceTypeCode

TORB See also

Creates a type code interface for a CORBA sequence (dynamic array).

```
function MakeObjectRefTypeCode(Bound: CorbaULong; const TC: ITypeCode):
  ITypeCode;
```

Description

Call MakeSequenceTypeCode to get a type code interface for a sequence. Bound specifies the maximum number of elements in the sequence. TC is the type code interface that indicates the type of the sequence elements. MakeSequenceTypeCode returns the type code interface for the sequence.

TORB.MakeStructure

TORB See also Example

Returns a CORBA structure built from a set of TAny values.

```
function MakeStructure(TypeCode: ITypeCode; const Elements: array of TAny):
  TAny;
```

Description

Call MakeStructure to create a TAny value for a CORBA structure (a record). The TypeCode parameter is an ITypeCode that describes the fields of the structure. The Elements parameter is an array of TAny values that are assigned to the fields of the structure.

Note: To obtain an ITypeCode that describes the CORBA structure, use the FindTypeCode method.

TORB.MakeStructureTypeCode

TORB See also

Creates a type code interface for a CORBA structure (record).

```
type
  TStructMember = record
    Name: string;
    TC: ITypeCode;
  end;

  TStructMembers = array of TStructMember;
function MakeStructureTypeCode(RepositoryID, Name: string; Members:
  TStructMembers): ITypeCode;
```

Description

Call MakeStructureTypeCode to get a type code interface for a structure. RepositoryID specifies the identifier of the structure. Name is the name of the structure type. Members lists the members of the structure, where each TStructMember specifies the name and type of the structure member. MakeStructureTypeCode returns the type code interface for the structure.

TORB.MakeTypeCode

TORB See also

Converts a TCKind value to a CORBA ITypeCode interface.

```
function MakeTypeCode(Kind: TCKind): ITypeCode;
```

Description

Call MakeTypeCode to get a type code interface for a specified type. Kind indicates the type that the interface should represent.

TORB.ObjectToString

TORB See also Example

Returns a string representation of an object.

```
function ObjectToString(const Obj: IObject): string;
```

Description

Call ObjectToString to convert an object reference to a string. The string can be converted back to an object reference using the StringToObject method.

TORB.Shutdown

TORB

Terminates a connection to the server.

```
procedure Shutdown;
```

Description

Call Shutdown from a client application to close a connection to the CORBA server.

TORB.StringToObject

TORB See also Example

Converts a string representation of a CORBA object back to an interface.

```
function StringToObject(const ObjectString: string): IObject;
```

Description

Call StringToObject to obtain an interface for an object that was previously converted using the ObjectToString method.

Note: Before calling StringToObject, the object's interface and Repository ID must be registered with the global CorbaInterfaceIDManager.

Hierarchy

TObject

AnyToObject function

See also

Converts a TAny value that contains a CORBA object reference to an object interface.

Unit

orbpas

```
function AnyToObject(Any: TAny; IID: TGUID): IObject;
```

Description

Call AnyToObject to obtain an interface that can be used to access the methods of the object referenced by the Any parameter.

If Any refers to an object that is instantiated locally, AnyToObject returns the interface of its implementation object. This is the same as calling TCorbaSkeleton.GetImplementation.

If Any refers to a remote object, IID must specify the globally unique identifier of the server interface class, and this GUID must be registered with the CORBA stub manager. AnyToObject returns the interface to its stub object. This is the same as calling TCorbaStubManager.CreateStub.

If Any does not contain a CORBA object reference, AnyToObject raises an exception.

BindStub procedure

See also Example

Obtains the interface to a CORBA server object as an IStub interface.

Unit

orbpas

```
procedure BindStub(RepositoryID, InstanceName, HostName: PChar; const Orb:
  IORB; RefCountServer: CorbaBoolean; out Stub: IStub);
```

Description

Call BindStub to obtain an interface for an object on a CORBA server that was not generated using Delphi. This interface does not include the marshaling support provided by a stub class. Client applications can either use a marshaling buffer directly when calling into this interface or they can call the CreateStub method of the global CorbaStubManager. CreateStub creates a stub object that handles marshaling of interface calls.

The RepositoryID parameter specifies the repository ID for the CORBA server interface. The InstanceName parameter specifies an optional instance name for the object instance on the CORBA server. The HostName parameter (also optional) allows the client application to identify the server machine by host name or IP address. The Orb parameter must take the global ORB variable, which is available if the unit includes CorbaInit in its uses clause. The RefCountServer parameter indicates whether the CORBA server maintains a reference count of client connections. The Stub parameter returns an interface for the server object.

RegisterUserException function

See also

Registers a user-defined exception with the ORB.

Unit

orbpas

```
type
  PExceptionDescription = type Pointer;
  PUserExceptionProxy = type Pointer;
  TUserExceptionFactoryProc = function : PUserExceptionProxy; cdecl;
function RegisterUserException(Name, RepositoryID: PChar; Factory:
  TUserExceptionFactoryProc): PExceptionDescription;
```

Description

Use RegisterUserException to enable the ORB to raise user-defined exceptions in your application. For every user-defined exception defined in the server's IDL file, you must define a corresponding descendant of ECorbaUserException. Then, for that ECorbaUserException descendant, create a function (of type TUserExceptionFactoryProc) that instantiates the exception object and returns the value of its Proxy property. Finally, to enable the ORB to raise these exceptions in your application, call RegisterUserException with the following arguments:

- Name is the name of the exception. This is the same as the value of ECorbaException.Name.
- RepositoryID is the repository ID of the user-defined exception in the IDL file.
- Factory is the function that instantiates an ECorbaUserException object and returns the value of its Proxy property.

RegisterUserException returns a pointer to an exception description. Use this pointer when you want to unregister the user exception.

BOA function

Returns the CORBA Basic Object Adaptor object.

Unit

CorbaObj

```
function BOA: TBOA;
```

Description

CORBA server applications use BOA to specify options, retrieve Principal values written by client applications, and to make CORBA server objects available or unavailable to clients.

CorbaBind function

See also Example

Returns an interface to let clients talk to a CORBA server object.

Unit

corbaobj

```
function CorbaBind(const RepositoryID: string; const ObjectName: string =
  ''; const HostName: string = ''): IObject;
function CorbaBind(const InterfaceID: TGUID; const ObjectName: string = '';
  const HostName: string = ''): IObject;
```

Description

Call CorbaBind in a CORBA client application to obtain an interface for talking to a CORBA server object. CorbaBind obtains an interface from the ORB and uses that to instantiate an appropriate stub object.

The RepositoryID parameter specifies the Repository ID for an object on the CORBA server. As an alternate way to call CorbaBind, the InterfaceID specifies the desired interface type on the CORBA server. Before specifying the desired interface using its type (GUID), the interface must be registered with the global CORBAInterfaceIDManager.

The ObjectName parameter optionally specifies the instance name of the server object that implements the interface.

The HostName parameter optionally specifies the server machine that runs the CORBA server. If you pass an empty string for HostName, a Smart Agent on the local network will bind to an object instance on the first server found which implements the requested interface (and object).

CorbaBind returns an interface that the client application can use to talk to the implementation object on the server.

Note: If the returned value is used for early binding, the stub class for the specified interface must be registered with the global CorbaStubManager. If the stub class is not registered, CorbaBind returns a generic stub interface that can only be used for late binding calls.

CorbaFactoryCreateStub function

See also Example

Returns an interface to let clients talk to a CORBA server object.

Unit

corbaobj

```
function CorbaFactoryCreateStub(const RepId, FactoryId, InstanceName,
  HostName: string; IID: TGUID): IObject;
```

Description

Call CorbaFactoryCreateStub in a CORBA client application to obtain an interface for talking to a CORBA server object. CorbaFactoryCreateStub obtains an interface from the ORB and uses that to instantiate an appropriate stub object.

The RepId parameter specifies the Repository ID for a factory object on the CORBA server. The FactoryId parameter specifies the Instance name of that factory object. This factory object creates an object instance on the server that supports the interface specified by the IID parameter. The InstanceName parameter gives an optional instance name for the object instance created by the factory object on the server. The HostName parameter optionally specifies the server's host name or IP address.

CorbaFactoryCreateStub returns an interface that the client application can use to talk to the implementation object on the server.

Before calling CorbaFactoryCreateStub, the stub class for the specified interface must be registered with the global CorbaStubManager.

Note: To use CorbaFactoryCreateStub, the CORBA server must be running an instance of TCorbaFactory or one of its descendants. Component writers writing custom stubs for servers that do not include a factory object should use the BindStub procedure instead.

MakePrincipal function

See also

Creates a TCorbaPrincipal type from an array of bytes.

Unit

corbaobj

```
type TCorbaPrincipal = array of Byte;
function MakePrincipal(const Bytes: array of Byte): TCorbaPrincipal;
```

Description

Call MakePrincipal to convert an array of bytes to a TCorbaPrincipal value that can be used when calling a CORBA stub's SetPrincipal method.

MarshalAny procedure

See also

Adds a Variant value to a marshaling buffer.

Unit

corbaobj

```
procedure MarshalAny(const OutBuf: IMarshalOutBuffer; const OV: Variant);
```

Description

Use MarshalAny to pass a Variant value using a CORBA marshaling buffer. Variants appear as CORBA Any types in the interface definition. MarshalAny handles the details of converting a Variant to a CORBA Any type and adding it to the marshaling buffer.

The OutBuf parameter is the marshaling buffer that receives the Variant value. The OV parameter is the Variant value that is added to the buffer.

MarshalObject procedure

See also

Adds an object interface to a marshaling buffer.

Unit

corbaobj

```
procedure MarshalObject(const OutBuf: IMarshalOutBuffer; IID: TGUID; const
  Intf: IObject);
```

Description

Use MarshalObject to pass an object reference using a CORBA marshaling buffer. The OutBuf parameter is the marshaling buffer that should receive the object reference. The IID parameter is the GUID or interface type for that object. The Intf parameter is the interface instance as it is returned by a skeleton class.

Note: MarshalObject assumes that the application contains either a stub object that implements the interface or the interface is registered with the CORBA skeleton manager.

MarshalWordBool procedure

See also!Alink(UnMarshalWordBool_function)

Adds a boolean value to a marshaling buffer.

Unit

corbaobj

```
procedure MarshalWordBool(const OutBuf: IMarshalOutBuffer; Value: WordBool);
```

Description

Use MarshalWordBool to pass a boolean value using a CORBA marshaling buffer. The OutBuf parameter is the marshaling buffer that should receive the boolean value. The Value parameter is equal to True or False.

MarshalWordBool converts Object Pascal WordBool values to unsigned characters.

ORB function

See also Example

Returns an interface to the CORBA ORB.

Unit

corbaobj

```
function ORB: TORB;
```

Description

Corba client applications use the CORBA ORB for binding to objects, disconnecting from servers, and obtaining string representations for object interfaces. Use ORB when calling routines, such as BindStub, that require an interface to the CORBA ORB.

Note: Calling ORB initializes the CORBA ORB if it is not already initialized.

SequenceToVariantArray function

See also

Converts a CORBA sequence type to an Object Pascal Variant array.

Unit

corbaobj

```
function SequenceToVariantArray(Sequence: TAny): Variant;
```

Description

Use SequenceToVariantArray to convert CORBA sequences that are represented as TAny values into a Variant array.

UnmarshalAny function

See also

Reads an Any value from a marshaling buffer into a Variant.

Unit

corbaobj

```
function UnmarshalAny(InBuf: IMarshalInBuffer): Variant;
```

Description

Use UnmarshalAny to receive a CORBA Any value using a CORBA marshaling buffer. The InBuf parameter is the marshaling buffer that holds the Any. UnmarshalAny converts the Any value to an OleVariant and returns it.

UnmarshalObject function

See also Example

Reads an object interface from a marshaling buffer.

Unit

corbaobj

```
function UnmarshalObject(InBuf: IMarshalInBuffer; IID: TGUID): IObject;
```

Description

Use UnmarshalObject to receive an object reference using a CORBA marshaling buffer. The InBuf parameter is the marshaling buffer that holds the object reference. The IID parameter is the GUID or interface type for that object. UnmarshalObject creates a stub object for the interface in the marshaling buffer and returns its interface.

Note: Unmarshal object assumes that either the server object is local or the interface has a stub class registered with the CORBA stub manager.

UnmarshalText function

See also Example

Reads a string from a marshaling buffer.

Unit

corbaobj

```
function UnmarshalText(InBuf: IMarshalInBuffer): string;
```

Description

Use UnmarshalText to retrieve a string using a CORBA marshaling buffer. The InBuf parameter is the marshaling buffer that holds the string. UnmarshalText retrieves the string from the marshaling buffer, frees the associated buffer, and returns the value as an AnsiString.

UnmarshalWideText function

See also

Reads a UNICODE string from a marshaling buffer.

Unit

corbaobj

```
function UnmarshalWideText(InBuf: IMarshalInBuffer): WideString;
```

Description

Use UnmarshalWideText to retrieve a UNICODE string using a CORBA marshaling buffer. The InBuf parameter is the marshaling buffer that holds the string. UnmarshalWideText retrieves the string from the marshaling buffer, frees the associated buffer, and returns the value as a WideString.

UnmarshalWordBool function

See also

Reads a boolean value from a marshaling buffer.

Unit

corbaobj

```
function UnmarshalWordBool(InBuf: IMarshalInBuffer): WordBool;
```

Description

Use UnmarshalWordBool to retrieve a boolean value from a CORBA marshaling buffer. The InBuf parameter is the marshaling buffer that holds the boolean value. UnmarshalWordBool retrieves the value from the marshaling buffer and converts it from an unsigned character into an Object Pascal WordBool.

UnRegisterUserException procedure

See also

Unregisters a user-defined exception with the ORB.

Unit

orbpas

```
type PExceptionDescription = type Pointer;
procedure UnRegisterUserException(Description: PExceptionDescription);
```

Description

Use UnRegisterUserException to prevent the ORB from raising a specific class of user-defined exceptions in your application.

The Description parameter is the description that is returned by the RegisterUserException function when you register the exception.

VariantArrayToSequence function

See also

Converts a Variant array to a CORBA sequence.

Unit

corbaobj

```
function VariantArrayToSequence(TypeCode: ITypeCode; const VariantArray:
  Variant): TAny;
```

Description

Use VariantArrayToSequence to create a CORBA sequence from a Variant array.

The TypeCode parameter is an ITypeCode interface that represents the type of the elements in the array. Use the ORB's MakeTypeCode method to obtain a value for this argument.

The VariantArray parameter is the Variant array that should be converted.

VariantArrayToSequence returns the CORBA sequence as a TAny value.

CorbaInterfaceIDManager variable

See also

CorbaInterfaceIDManager keeps track of the association between CORBA Repository IDs and interfaces.

Unit

corbaobj

```
var CorbaInterfaceIDManager: TCorbaInterfaceIDManager;
```

Description

CORBA client applications register the Repository ID for each CORBA interface with the global CorbaSkeletonManger so that the CorbaBind method (or the ORB variable's Bind method) can look up the necessary information for binding to a CORBA server object. The code to perform this registration is automatically added to the stub and skeleton unit (_TLB.pas file).

CorbaSkeletonManager variable

See also

CorbaSkeletonManger keeps track of the association between CORBA skeleton classes and interfaces.

Unit

corbaobj

```
var CorbaSkeletonManager: TCorbaSkeletonManager;
```

Description

CORBA server applications must register their CORBA skeleton classes (TCorbaSkeleton descendants) with the global CorbaSkeletonManger so that the server can instantiate the appropriate object when clients request a server interface. The code to perform this registration is automatically added to the stub and skeleton unit (_TLB.pas file).

CorbaStubManager variable

See also

CorbaStubManger keeps track of the association between CORBA stub classes and server interfaces.

Unit

corbaobj

```
var CorbaStubManager: TCorbaStubManager;
```

Description

CORBA client applications must register their CORBA stub classes (TCorbaStub descendants) with the global CorbaStubManger so that stub objects can be created for server interface instances.

The code to register stub classes is automatically added to the stub and skeleton unit (_TLB.pas file). However, developers using custom stub classes rather than the classes generated while creating the server must add the code to register these classes with CorbaStubManger.

Comcorba unit

Other objects and components, by unit

TCorbaComObjectFactory

TCorbaComObjectFactory creates a COM object in response to a request from a CORBA client.

Corbaobj unit

Corbaobj routines
Other objects and components, by unit

ECorbaException
ECorbaException is the exception class that is raised when a CORBA application can't perform a requested action.

ECorbaUserException
ECorbaUserException represents exceptions that are defined by the CORBA application developer.

ICorbaObject
ICorbaObject represents the interface of an object accessed through the ORB.

TBOA
TBOA represents the CORBA Basic Object Adaptor.

TCorbaDispatchStub
TCorbaDispatchStub is the base class for all automatically generated CORBA stub objects.

TCorbaFactory
TCorbaFactory is the base class for objects instantiated remotely by the global CorbaBind function.

TCorbaImplementation
TCorbaImplementation is the base class for classes that implement CORBA server interfaces.

TCorbaInterfaceIDManager
TCorbaInterfaceIDManager keeps track of which Repository IDs represent which interfaces.

TCorbaListManager
TCorbaListManager is the base class for classes that keep track of available CORBA classes and interfaces.

TCorbaObjectFactory
TCorbaObjectFactory creates a CORBA server object.

TCorbaSkeleton
TCorbaSkeleton is the base class for all CORBA skeleton objects.

TCorbaSkeletonManager
TCorbaSkeletonManager keeps track of which skeleton classes represent which interfaces.

TCorbaStub
TCorbaStub is the base class for all CORBA stub objects.

TCorbaStubManager
TCorbaStubManager keeps track of which stub classes represent specific server interfaces.

TCorbaVCLComponentFactory
TCorbaVCLComponentFactory creates a CORBA server object.

TORB
TORB represents the CORBA Object Request Broker.

Corbastd unit

Other objects and components, by unit

TAppServerSkeleton

TAppServerSkeleton is the skeleton class for a CORBA data module.

TAppServerStub

TAppServerStub is the stub class for a CORBA data module.

Orbpas unit

Orbpas routines
Other objects and components, by unit

ICorbaObj
ICorbaObj is the interface that represents a CORBA object.

IMarshalInBuffer
IMarshalInBuffer is the interface for reading return values from a marshaling buffer.

IMarshalOutBuffer
IMarshalOutBuffer is the interface for writing arguments to a marshaling Buffer.

ISkeleton
ISkeleton allows a CORBA skeleton to access its implementation class or obtain a marshaling buffer for return values.

ISkeletonObject
ISkeletonObject is the interface that represents a CORBA skeleton.

IStub
IStub represents a remote server object for a CORBA stub.

IStubObject
IStubObject is the interface that represents a CORBA stub.

Provider unit

Other objects and components, by unit

ECorbaDispatch

ECorbaDispatch is the exception class that is raised when a CORBA application encounters a problem using DII.

Corbaobj routine

Corbaobj objects/components
Other routines, by unit

BOA function
Returns the CORBA Basic Object Adaptor object.

CorbaBind function
Returns an interface to let clients talk to a CORBA server object.

CorbaFactoryCreateStub function
Returns an interface to let clients talk to a CORBA server object.

CorbaInterfaceIDManager variable
CorbaInterfaceIDManager keeps track of the association between CORBA Repository IDs and interfaces.

CorbaSkeletonManager variable
CorbaSkeletonManger keeps track of the association between CORBA skeleton classes and interfaces.

CorbaStubManager variable
CorbaStubManger keeps track of the association between CORBA stub classes and server interfaces.

MakePrincipal function
Creates a TCorbaPrincipal type from an array of bytes.

MarshalAny procedure
Adds a Variant value to a marshaling buffer.

MarshalObject procedure
Adds an object interface to a marshaling buffer.

MarshalWordBool procedure
Adds a boolean value to a marshaling buffer.

ORB function
Returns an interface to the CORBA ORB.

SequenceToVariantArray function
Converts a CORBA sequence type to an Object Pascal Variant array.

UnmarshalAny function
Reads an Any value from a marshaling buffer into a Variant.

UnmarshalObject function
Reads an object interface from a marshaling buffer.

UnmarshalText function
Reads a string from a marshaling buffer.

UnmarshalWideText function
Reads a UNICODE string from a marshaling buffer.

UnmarshalWordBool function
Reads a boolean value from a marshaling buffer.

VariantArrayToSequencefunction
Converts a Variant array to a CORBA sequence.

Orbpas routine

Orbpas objects/components
Other routines, by unit

AnyToObject function
Converts a TAny value that contains a CORBA object reference to an object interface.

BindStub procedure
Obtains the interface to a CORBA server object as an IStub interface.

RegisterUserException function
Registers a user-defined exception with the ORB.

UnRegisterUserException procedure
Unregisters a user-defined exception with the ORB.

All elements are displayed here
No additional elements exist for this unit or routine.