Repository Overview

See Also

The repository provides a common place to persist information about relationships between objects. In doing so, it provides a standard way to describe software tools:

- Software tools are composed of various classes of objects.
- Objects expose interfaces.
- Interfaces are composed of properties, methods, and collections.
- Collections relate interfaces to other interfaces and thus relate objects to other objects.

A software tool is modeled in the repository via a tool information model (TIM). Each TIM is stored in the repository database as a repository type library.

A significant aspect of the repository is that the database is dynamic and extensible. Very little of the actual structure is fixed; tables and columns of data are added as new models are defined, and as existing models are extended. This is significant because it enables any software tool or set of software tools to be modeled by the repository. It also enables existing models to be extended by a user.

For example, Visual Basic 5.0 provides a tool information model called the Microsoft Development Objects (MDO) Model. The MDO Model tracks Visual Basic projects and (their contents) in the repository. The set of existing Visual Basic projects is represented in the repository by the Contents collection.

The repository is not limited to software tools developed using Visual Basic. Visual Basic is only one of many potential clients that can use the repository to model a software system.

Repository Browser Overview

See Also

The Repository Browser represents a joint development effort between Crescent Division of Progress Software and Microsoft Corporation. The Repository Browser presents a hierarchical view of the contents of a repository. Crescent also offers an enhanced browser, called the Pro Browser.

One of the key features of the Repository Browser is it's ability to visualize data. Visualization is key to locating and working with repository data. Another key feature is dynamic data retrieval. As a repository database grows, this dynamic data retrieval feature enables optimal performance and resource utilization.

Repository Browser User Interface

See Also

The Repository Browser is designed to provide a look-and-feel that is similar to the Windows Explorer. This familiar interface allows a user to easily browse the contents of a repository.

The following tables describe the purpose of some of the browser menus:

File Menu	Description
Open	Opens a repository.
Properties	Displays a dialog to view object properties.

View Menu	Description
Toolbar	Toggles the display of the toolbar.
Status Bar	Toggles the display of the status bar.
Refresh	Updates the browser view.
Filters	Displays the Repository Filters dialog to enable you to change the repository view.

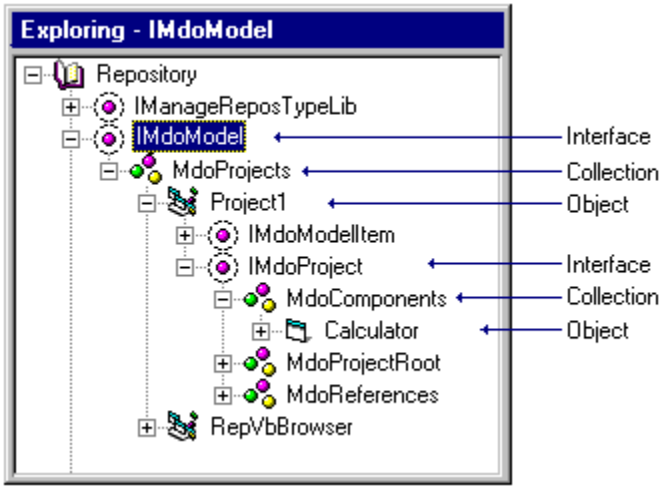
Repository Browser Hierarchy

See Also

All objects are normally related to the repository root, either directly, or indirectly through other objects and relationships. This makes it possible to navigate the repository hierarchically. The browser visualizes repository data in a hierarchical fashion. The Pro Browser provides an enhancement to this view as well other representations and visualizations.

Browser data may be thought of as having a set of repeating patterns: interfaces have collections; collections contain repository objects; repository objects have interfaces. The following figure illustrates this point.

To reduce the amount of information that is displayed, use the Repository Filter dialog to filter out interfaces from the browser display.



Tool Information Models in the Repository Browser

See Also

A *tool information model* (TIM) is an object model for a software tool (or a set of software tools). A TIM is composed of classes, relationship types, interfaces, property definitions, methods, and collection types.

Tool information models are stored in the repository. Each TIM describes the types of objects and relationships that make up a software tool. For example, Visual Basic 5.0 provides a TIM named “MdoTypeLib”. This TIM describes the forms, controls, and other objects that make up a Visual Basic project. It also describes the relationships between those objects; projects contain modules and forms, forms contain controls, and so on.

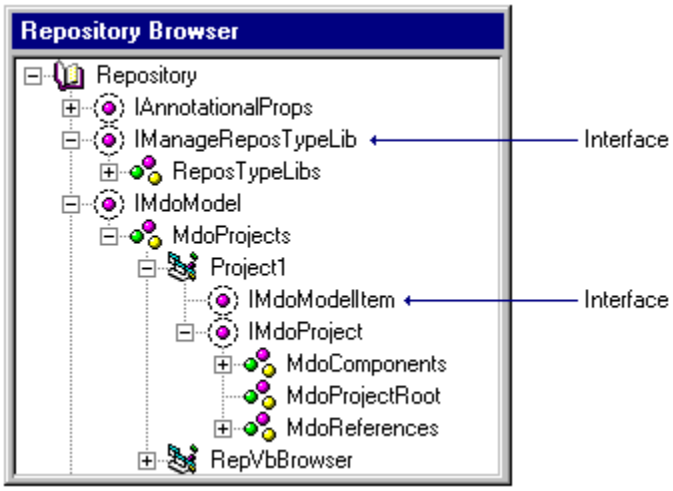
The *Type Information Model* is the tool information model that the repository uses to understand all other tool information models.

Interfaces in the Repository Browser

See Also

A class implements one or more interfaces. A repository object is an instance of a class, and therefore provides one or more interfaces, as defined by the class to which it conforms. The repository object is encapsulated and is only accessible through its interfaces. Each interface is composed of properties, methods, and collections.

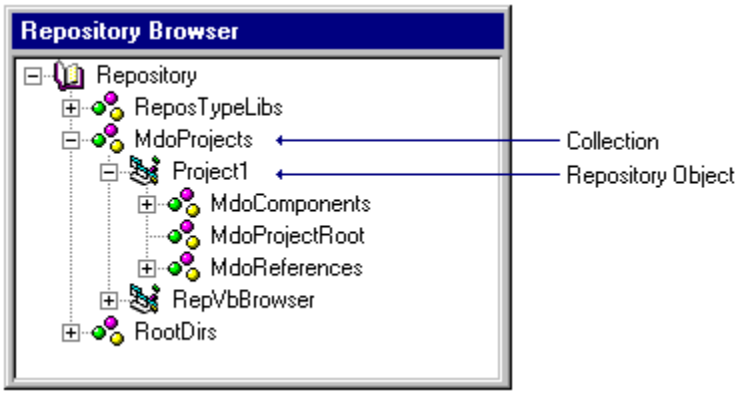
By default, the Repository Browser displays interfaces in the repository hierarchy. If you find the visualization of interfaces distracting, use the Repository Filter dialog to turn off the display of repository interfaces.



Repository Objects in the Repository Browser

See Also

A repository object is an instance of a class that has been defined in a repository tool information model. In the following figure “Project1” and “RepVbBrowser” are both instances of the class “MdoProject”, and they both belong to the collection entitled “MdoProjects”.



A repository object provides one or more interfaces. Each interface defines a set of more properties, methods, and collections. The definition of a repository object may be extended by adding additional interfaces to its corresponding class, or by adding additional properties, methods, or collections to an existing interface.

Relationships in the Repository Browser 

See Also

A relationship is a logical connection between one repository object, the origin object, and a second repository object, the destination object.

The Repository Browser visualizes the repository contents by navigating relationships from the repository root object (the source object) to all of the target objects in collections that are attached to the repository root. Each target object can then be the source object of another relationship collection, and so on.

Relationships themselves are not directly visualized in the Repository Browser. They are implied by the collections that are displayed in the Repository Browser's hierarchical tree metaphor. When an object is selected, relationship information for the object can be viewed by selecting the **Relationship** toolbar button. This information is also available by selecting the **Relationship** item on the pop-up context menu for the selected object.

Collections in the Repository Browser

See Also

A *relationship* collection is a set of relationships of the same relationship type that are all connected to a common source object. An *object* collection is a set of repository objects that are all connected to a common source object via a relationship collection.

Object collections are displayed in the browser as a node in the browser tree. The objects within a particular object collection are displayed underneath the collection node, and connected to it. Using Explorer as an analogy, the contents of a folder on your hard disk can be thought of as a collection of file and sub-folder objects. In this case, the common source object to which all of the files and sub-folders are connected is the parent folder object. In this analogy, only one type of collection is defined for a folder; namely, the collection of files and sub-folders. Consequently, in the Explorer, the actual collection object is not displayed to you. However, in the repository, multiple collections of different types can be attached to any single parent object. This is why the Repository Browser displays the collection between the parent object and the objects within the collection.

A relationship between objects is actually a relationship between the interfaces of two objects. This is a key point in working with repository data: *the repository contains collections of relationships between interfaces of objects*.

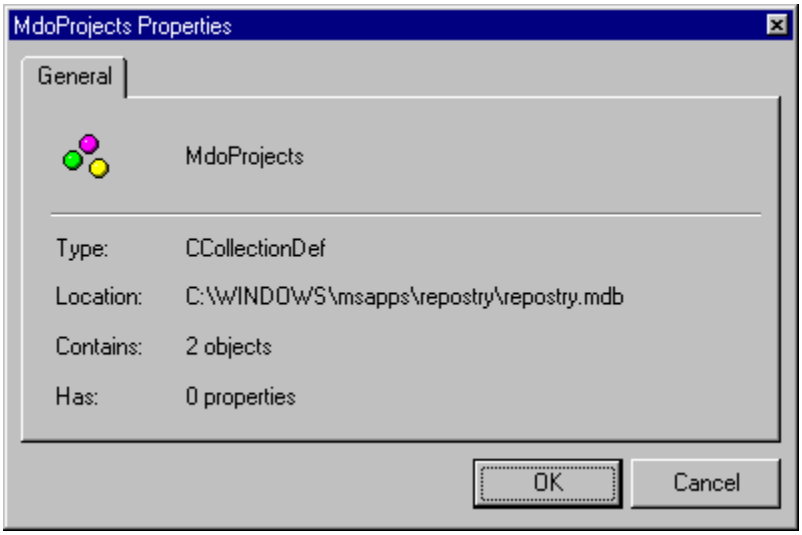
Object Properties in the Repository Browser

See Also

There are two kinds of properties that can be viewed by the Repository Browser; properties that are common to all repository objects, and properties that are specific to a particular class of repository object.

The Repository Browser displays the properties that are common to all repository objects in a special tabbed dialog. The table below describes the contents of this dialog.

Properties that are specific to a particular class of object are *not* displayed in this dialog. They are represented as members of the Members collection of the interfaces that are implemented by the object.

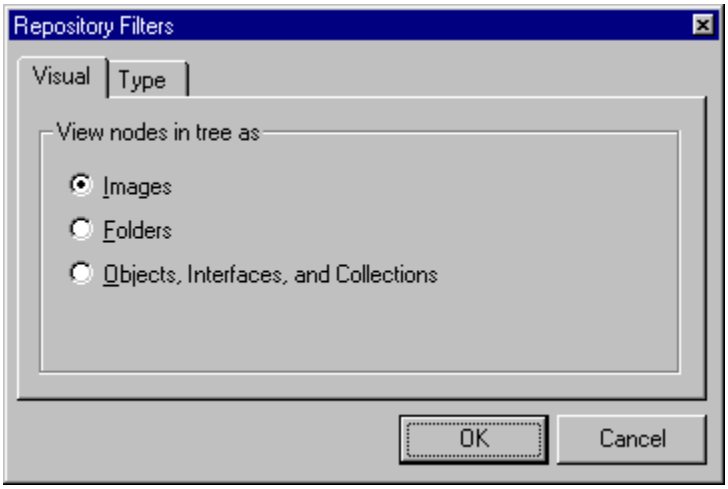


Property Name	Value
Type	The type of repository object as defined by the appropriate tool information model.
Location	The location of the Access MDB repository, or SQL Server name.
Contains	If viewing properties of an object collection, this property contains the count of objects contained within the collection.
Has	If viewing properties of a repository object, this property contains the count of members attached to the object.

Repository Browser Filter Dialog

See Also

The Repository Browser provides a simple way to filter data. This dialog is accessible under the *View* menu. The *Visual* tab provides a way to view tree nodes and list items using different types of images. The *Type* tab allows you to specify whether or not repository interfaces in the tree are displayed.



- If you change any option in the *Type* tab and press OK, the entire repository will be closed and redrawn.
- If you change an option in the *Visual* tab and press OK, the nodes and list items are merely redrawn.

Viewing Object Types in the Repository Browser

See Also

Every repository object conforms to a specific class, or object type. The type of an object can be viewed by selecting the object in the browser tree and viewing the type in the second status panel at the bottom of the browser, or by viewing the properties dialog. To view the properties dialog, click on the object using the right mouse button, and select Properties.

Repository Root and the Repository Browser

See Also

The repository root object is the root object for all repository data. The Repository Browser displays repository data by iterating through all of the interfaces attached to the root object, then iterating through the collections attached to those interfaces, and so on.

Each tool information model normally adds an interface either to the repository root object, or to some other object that is ultimately connected to the repository root object. The added interface is then defined to contain one or more collections that relate to other objects in that tool information model.

Upgrade to Pro Browser

See Also

An enhanced Pro Browser is available from Crescent Division of Progress Software. With the Pro Browser you can:

- Make use of enhanced filtering
- Perform powerful searches
- Launch multiple instances of the browser
- Create your own Tool Information Models
- Attach bug tracking and test cases to Visual Basic objects
- Display a WebView of repository data

To contact Crescent:

Crescent Division of Progress Software
14 Oak Park
Bedford, Massachusetts 01730

Phone: (617) 280-3000
Fax: (617) 280-4025
BBS: (617) 280-4221
Sales: (800) 35-BASIC

CompuServe: 70662, 2605
Internet Mail: crescent-support@progress.com
Web Site: http://crescent.progress.com
Pro Browser: http://crescent.progress.com/crescent/probrowser.html
FTP: ftp.progress.com/pub/crescent

About Repository Browser

The Repository Browser is a joint development effort between Microsoft Corporation and Crescent Division of Progress Software. An upgraded version, the Pro Browser, is available through Crescent.

ObjectCol Object

An object collection is a set of repository objects that can be enumerated. Two kinds of object collections are supported by the repository:

1. The collection of destination objects that correspond to the relationships in a relationship collection. Use the **RelationshipCol** object to manage this kind of collection.
2. The collection of all objects in the repository that conform to a particular class or expose a particular interface.

When to Use

Use the **ObjectCol** object to enumerate the collection of repository objects that conform to a particular class or expose a particular interface. With this object, you can:

- Get a count of the number of objects in the collection.
- Retrieve one of the objects in the collection.
- Refresh the cached image of the object collection.

Properties

Property	Description
Count	The count of the number of items in the collection.
Item	Retrieves the specified object from the collection.

Methods

Method	Description
Refresh	Refreshes the cached image of the collection.

ObjectCol Count Property

See Also

A long integer that contains the count of the number of items in the collection. This is a read-only property.

Syntax

object.**Count**

The **Count** property syntax has these parts:

Part	Description
object	The object collection.

ObjectCol Item Property

See Also

Use this property to retrieve an object from the collection. This is a read-only property. There are three variations of this property.

Syntax

Set *variable* **=** *object***.Item(** *index* **)**

Set *variable* **=** *object***.Item(** *objName* **)**

Set *variable* **=** *object***.Item(** *objId* **)**

The **Item** property syntax has these parts:

Part	Description
variable	A variable declared as a RepositoryObject. Receives the specified repository object.
object	The object collection.
index	The index of the repository object to be retrieved from the collection.
objName	The name associated with the repository object to be retrieved from the collection.
objId	The object identifier of the repository object to be retrieved from the collection.

Remarks

An object can only be retrieved by name if it is the destination object of a naming relationship.

ObjectCol Refresh Method

See Also

This method refreshes the cached image of the object collection. Only cache data that has not been changed *by the current process* is refreshed.

Syntax

Call *object*.**Refresh(** *milliSecs* **)**

The **Refresh** method syntax has these parts:

Part	Description
object	The object collection.
milliSecs	All unchanged data relating to the item that has been in the cache for longer than *millisecs* milliseconds is refreshed. Set to zero to refresh all unchanged data.

Relationship Object

See Also Properties Methods Collections

A relationship connects two repository objects in the repository database. A relationship has an origin repository object, a destination repository object, and a set of properties. Each relationship conforms to a particular relationship type.

When to Use

Use the **Relationship** object to manipulate the properties of a relationship, to delete a relationship, or to refresh the cached image of a relationship.

Properties

Property	Description
Destination	The destination object of the relationship.
Interface	The specified object interface.
Name	The name of the relationship's destination object.
Origin	The origin object of the relationship.
Repository	The open repository instance through which this relationship was instantiated.
Source	The source object of the relationship.
Target	The target object of the relationship.
Type	The type of the relationship.

Methods

Method	Description
Delete	Deletes a relationship.
Lock	Locks the relationship.
Refresh	Refreshes the cached image of a relationship.

Collections

Collection	Description
Properties	The collection of all of the properties that are attached to the relationship.

Relationship Delete Method

See Also

This method deletes the relationship from its relationship collection. The target object of the relationship is deleted, if it is also a destination object, and the relationship type indicates that deletes are to be propagated.

Syntax

Call *object*.**Delete**

The **Delete** method syntax has these parts:

Part	Description
object	An object expression that evaluates to a Relationship object.

Relationship Destination Property

See Also

This property is the destination object for the relationship. This is a read-only property.

Syntax

Set *variable* **=** *object***.Destination**

The **Destination** property syntax has these parts:

Part	Description
variable	A variable declared as a RepositoryObject. Receives the destination object for the relationship.
object	An object expression that evaluates to a Relationship object.

Relationship Interface Property

See Also

Use this property to obtain a view of the relationship object that uses an alternate interface as the default interface. This is a read-only property. There are two variations of this property.

Syntax

Set *variable* **=** *object***.Interface(** *interfaceId* **)**

Set *variable* **=** *object***.Interface(** *objectId* **)**

The **Interface** property syntax has these parts:

Part	Description
variable	An object variable. Receives the relationship object with the specified interface as the default interface.
object	An object expression that evaluates to a Relationship object.
interfaceId	The interface identifier for the interface to be retrieved.
objectId	The object identifier for the interface definition to which the interface to be retrieved conforms.

Relationship Lock Method

See Also

Use this method to lock the relationship. Locking the relationship prevents other processes from updating the relationship while you are working with it. The lock is released when you end the current transaction.

Call *object*.**Lock**

The **Lock** method syntax has these parts:

Part	Description
object	An object expression that evaluates to a Relationship object.

Relationship Name Property

See Also

A character string that contains the name that the relationship assigns to the destination object.

Syntax

object.**Name**

The **Name** property syntax has these parts:

Part	Description
object	An object expression that evaluates to a Relationship object.

Relationship Origin Property

See Also

This property is the origin object of the relationship. This is a read-only property.

Syntax

Set *variable* **=** *object***.Origin**

The **Origin** property syntax has these parts:

Part	Description
variable	A variable declared as a RepositoryObject. Receives the origin object for the relationship.
object	An object expression that evaluates to a Relationship object.

Relationship Refresh Method

See Also

This method refreshes the cached image of the relationship. Only cache data that has not been changed *by the current process* is refreshed.

Syntax

Call *object*.**Refresh(** *milliSecs* **)**

The **Refresh** method syntax has these parts:

Part	Description
object	An object expression that evaluates to a Relationship object.
milliSecs	All unchanged data relating to the item that has been in the cache for longer than *millisecs* milliseconds is refreshed. Set to zero to refresh all unchanged data.

Relationship Repository Property

See Also

The **Repository** property is the open repository instance through which this relationship was instantiated. This is a read-only property.

Syntax

Set *variable* **=** *object***.Repository**

The **Repository** property syntax has these parts:

Part	Description
variable	A variable declared as an instance of the Repository class. Receives the object that represents the open repository instance.
object	An object expression that evaluates to a Relationship object.

Relationship Properties Collection

See Also

A **Properties** collection contains all of the persistent properties and collections that are attached to an object via a particular interface. The Relationship object exposes two separate Properties collections. These collections are exposed by:

1. The IRelationship interface (the default).
2. The IAnnotationalProps interface.

Syntax

Set *variable* = *object*.**Properties(** *index* **)**

The **Properties** collection syntax has these parts:

Part	Description
variable	A variable declared as a ReposProperty object. Receives the specified property.
object	An object expression; evaluates to an object that exposes: • IRelationship, or • IAnnotationalProps as the default interface.
index	An integer index that identifies which property in the collection is to be addressed. The valid range is from one to the number of elements in the collection. The number of elements in the collection is specified by *object*.Properties.Count.

Remarks

For details on how to access a member of an interface that is not the default interface, see Accessing Automation Object Members.

Relationship Source Property

See Also

This property is the source object of the relationship. This is a read-only property.

Syntax

Set *variable* **=** *object***.Source**

The **Source** property syntax has these parts:

Part	Description
variable	A variable declared as a RepositoryObject. Receives the source object for the relationship.
object	An object expression that evaluates to a Relationship object.

Relationship Target Property

See Also

This property is the target object of the relationship. This is a read-only property.

Syntax

Set *variable* **=** *object***.Target**

The **Target** property syntax has these parts:

Part	Description
variable	A variable declared as a RepositoryObject. Receives the target object for the relationship.
object	An object expression that evaluates to a Relationship object.

Relationship Type Property

See Also

This property specifies the type of the relationship. More specifically, it is the object identifier of the relationship definition object for the relationship. **Type** is a read-only property. To copy this property to another variable, use a variable that is declared as a Variant.

Syntax

object.**Type**

The **Type** property syntax has these parts:

Part	Description
object	An object expression that evaluates to a Relationship object.

RelationshipCol Object

A relationship collection is the set of relationships that are attached to a particular source repository object. All of the relationships in the collection must conform to the same relationship type.

When to Use

Use this object to manage the relationships that belong to a particular relationship collection. With this object, you can:

- Get a count of the number of relationships in the collection.
- Add and remove relationships to and from the collection.
- If the collection is sequenced, place a relationship in a specific spot in the collection sequence.
- Retrieve a specific relationship or target object from the collection.
- Refresh the cached image of the collection.
- Obtain the type of the collection.

Properties

Property	Description
Count	The count of the number of items in the collection.
Item	Retrieves the specified relationship or target object from the collection.
Source	The source object for the relationship collection.
Type	The object identifier for the collection's definition object.

Methods

Method	Description
Add	Adds a relationship to the collection.
Insert	Inserts a relationship into a specific spot in a sequenced collection.
Move	Moves a relationship from one spot to another in a sequenced collection.
Refresh	Refreshes the cached image of the collection.
Remove	Removes a relationship from the collection.

RelationshipCol Add Method

See Also

This method is used to add a new item to a relationship collection, when the sequencing of relationships in the collection is not important. The new relationship connects the *reposObj* object to the source object of the collection. The new relationship is passed back to the caller.

Syntax

Set *variable* **=** *object***.Add(** *reposObj, objName* **)**

The **Add** method syntax has these parts:

Part	Description
variable	A variable declared as a Relationship. Receives the new relationship that is created for the *reposObj* repository object.
object	The relationship collection.
reposObj	The repository object whose relationship is to be added to the collection.
objName	The name that the new relationship is to use for the *reposObj* object. This parameter is optional.

RelationshipCol Count Property

See Also

A long integer that contains the count of the number of items in the collection. This is a read-only property.

Syntax

object.**Count**

The **Count** property syntax has these parts:

Part	Description
object	The relationship collection.

RelationshipCol Insert Method

See Also

This method adds a relationship to the collection at a specified point in the collection sequence. The new relationship connects the *reposObj* object to the source object of the collection. The new relationship is passed back to the caller.

Syntax

Set *variable* **=** *object***.Insert(** *reposObj, index, objName* **)**

The **Insert** method syntax has these parts:

Part	Description
variable	A variable declared as a Relationship. Receives the new relationship that is created for the *reposObj* repository object.
object	The relationship collection.
reposObj	The repository object whose relationship is to be added to the collection.
index	The index of the sequence location where the relationship is to be inserted. If another relationship is already present at this sequence location, the new relationship is inserted before the existing relationship.
objName	The name that the new relationship is to use for the *reposObj* object. This parameter is optional.

Remarks

This method can only be used with sequenced collections.

RelationshipCol Item Property

See Also

Use this property to retrieve a target object or relationship from the collection. This is a read-only property. There are three variations of this property.

Syntax

Set *variable* **=** *object***.Item(** *index* **)**

Set *variable* **=** *object***.Item(** *objName* **)**

Set *variable* **=** *object***.Item(** *objId* **)**

The **Item** property syntax has these parts:

Part	Description
variable	A variable declared as a RepositoryObject. Receives the target object of the specified relationship. See the Remarks section for an exception.
object	The relationship collection.
index	The index of the relationship to be retrieved from the collection.
objName	The name that the relationship uses to refer to its destination object. This variation can only be used when the target object is also the destination object, and when the collection requires unique names for destination objects.
objId	The object identifier for the target object to be retrieved from the collection.

Remarks

This property is available on two interfaces: the default interface, ITargetObjectCol, and a second interface, IRelationshipCol. If you choose to access the property that is exposed by the IRelationshipCol interface, then your *variable* will receive the specified relationship instead of the relationship's target object. In this case, you should declare your *variable* as a Relationship instead of a RepositoryObject.

For details on how to access a member of an interface that is not the default interface, see Accessing Automation Object Members.

RelationshipCol Move Method

See Also

This method moves a relationship from one point in the collection sequence to another point.

Syntax

Call *object*.**Move(** *indexFrom, indexTo* **)**

The **Move** method syntax has these parts:

Part	Description
object	The relationship collection.
indexFrom	The index of the relationship to be moved in the collection sequence.
indexTo	The index of the sequence location to which the relationship is to be moved.

Remarks

This method can only be used with sequenced collections.

RelationshipCol Refresh Method

See Also

This method refreshes the cached image of the relationship collection. Only cache data that has not been changed *by the current process* is refreshed.

Syntax

Call *object*.**Refresh(** *milliSecs* **)**

The **Refresh** method syntax has these parts:

Part	Description
object	The relationship collection.
milliSecs	All unchanged data relating to the item that has been in the cache for longer than *millisecs* milliseconds is refreshed. Set to zero to refresh all unchanged data.

RelationshipCol Remove Method

See Also

This method removes the specified relationship from the collection. The target object is also deleted, if it is also a destination object, and the relationship type indicates that deletes are to be propagated. There are two variations of this method.

Syntax

Call *object*.**Remove(** *index* **)**

Call *object*.**Remove(** *objName* **)**

The **Remove** method syntax has these parts:

Part	Description
object	The relationship collection.
index	The index of the relationship to be removed from the collection.
objName	The relationship that uses this name for its destination object is to be removed from the collection.

Remarks

The variation of this method that specifies the relationship by name can only be used for collections that require unique names for destination objects.

RelationshipCol Source Property

See Also

Use this property to retrieve the source object for the relationship collection. This is a read-only property.

Syntax

Set *variable* **=** *object***.Source**

The **Source** property syntax has these parts:

Part	Description
variable	A variable declared as a RepositoryObject. Receives the source object of the relationship collection.
object	The relationship collection.

RelationshipCol Type Property

See Also

This property specifies the type of the collection. More specifically, it is the object identifier of the collection definition object for the collection. **Type** is a read-only property. To copy this property to another variable, use a variable that is declared as a Variant.

Syntax

object.**Type**

The **Type** property syntax has these parts:

Part	Description
object	The relationship collection.

Repository Object

See Also Properties Methods Collections

When you define a tool information model, the classes, relationships, properties, and collections for the model are stored in a repository. Multiple tool information models may be stored in the same repository.

When you access a repository, you do so via a **Repository** instance.

When to Use

You can use a **Repository** instance to:

- Create a new repository database.
- Connect to a repository.
- Access the root repository object.
- Retrieve a specific repository object.
- Create new repository objects.
- Refresh cached repository data.
- Manage repository transactions.

Properties

Property	Description
ConnectionString	The ODBC connection string that the repository engine uses to obtain an ODBC connection. This property is not a default interface member.
Object	Retrieves the specified repository object.
ReposConnection	The ODBC connection handle that the repository engine is using to access the repository database. This property is not a default interface member.
RootObject	The root repository object of the currently open repository.
Transaction	The transaction processing interface.

Methods

Method	Description
Create	Creates a new repository database.
CreateObject	Creates a new repository object in the currently open repository.
FreeConnection	Releases an ODBC connection handle. This method is not a default interface member.
GetNewConnection	Obtains a new ODBC connection handle using the same connection settings that the repository engine is using to access the repository database. This method is not a default interface member.
InternalIDToObjectID	Converts an internal identifier into an

	object identifier.
ObjectIDToInternalID	Converts an object identifier into an internal identifier.
Open	Opens the specified repository.
Refresh	Refreshes the cached image of all data for the currently open repository.

Repository ConnectionString Property

See Also

This property contains the ODBC connection string that the repository engine uses to obtain an ODBC connection to the repository database. This is a read-only property.

This property is not attached to the default interface for the Repository Automation object; it is attached to the **IRepositoryODBC** interface. For details on how to access a member of an interface that is not the default interface, see Accessing Automation Object Members.

Syntax

object.**ConnectionString**

The **ConnectionString** property syntax has these parts:

Part	Description
object	The object that represents the open repository instance through which this program is interacting with the repository.

Remarks

The ODBC connection string can contain user identification and password information. Take care to protect this information from exposure to unauthorized access.

Accessing Automation Object Members

See Also

Microsoft Repository contains a number of Automation objects that support multiple interfaces. With each Automation object, one interface is defined to be the default interface, and the members (the properties, methods, and collections) that are attached to that interface are accessible via the standard Visual Basic mechanisms.

When accessing members that are attached to an interface that is *not* the default interface for an Automation object, a different access technique must be used. An additional reference to the object must be declared that explicitly calls for the non-default interface. The non-default interface members can then be accessed via the new object reference.

The following example illustrates how to access a property that is attached to an interface that is not the default interface for an Automation object. In this example, the connection string that is used to connect to the repository database is retrieved. **Repository** objects implement the **IRepositoryODBC** interface; this interface is not the default interface. The **ConnectionString** property is attached to the IRepositoryODBC interface. The ConnectionString property is the ODBC connection string that the repository uses when connecting to a database server.

```
Dim myRepos     As Repository
Dim nonDefIfc   As IRepositoryODBC

'  Initialize the myRepos object by opening a
'  connection to a repository database.

Set nonDefIfc = myRepos
connect$ = nonDefIfc.ConnectionString
```

In this example, the *nonDefIfc* object does not use up additional resources; it is just an alternate view of the *myRepos* object.

Repository Create Method

See Also

Use this method to create a new repository database. The fundamental repository tables are automatically created. The root repository object of the new repository is passed back to the caller.

Syntax

Set *variable* **=** *object***.Create(** *connect, user, password, flags* **)**

The **Create** method syntax has these parts:

Part	Description
variable	A variable declared as a RepositoryObject. Receives the root repository object for the new repository.
object	The instance of the Repository class that you are using to create the new repository database.
connect	The ODBC connection string to be used for accessing the database server that will host your new repository.
user	The user name to use for identification to the database server.
password	The password that matches the *user* input parameter.
flags	Flags that determine database access and caching behavior for the open repository. For details, see the ConnectionFlags Enumeration.

Connecting to a Repository Database

Microsoft Repository can access repository databases that are managed by either Microsoft Jet or Microsoft SQL Server. The ODBC connection string that is used to specify the location of the repository database varies, depending upon which database server is managing the repository database.

The ODBC connection string contains *keyword=keyValue* pairs, separated by semicolons.

Connecting to a Jet Repository Database

To connect to a Jet repository database, use the DBQ keyword to specify the path to the database file. The DBQ keyword must be the first keyword in the connection string, if it is present. If the DBQ keyword is not present, the connection string is assumed to contain only a database path specification. In this case, the repository will add the DBQ keyword to the front of the ODBC connection string before passing it on to the database server.

Connecting to a SQL Server Repository Database

To connect to a SQL Server repository database, use the SERVER keyword to specify the SQL Server name and (optionally) the database name. To specify the database name, use this syntax:

```
SERVER=serverName;DATABASE=databaseName
```

If the database name is not specified, the default database for the user performing the open is used.

Note If you choose to use a SQL Server repository database, you must create an empty database and specify which users can access the database. The repository engine will not automatically create this database for you.

Connecting via a Data Source Name

You can use the DSN keyword to specify a Data Source Name (DSN) to connect to either a Jet or SQL Server repository database. The DSN keyword specifies a Data Source Name that has been configured via the ODBC Data Source Administrator.

The Default Repository Database

If you do not specify the repository database via any of the mechanisms described above, a connection will be established to the *default repository database*. The location of the default repository database is stored in the system registry in this registry key:

```
HKEY_LOCAL_MACHINE\
   Software\
      Microsoft\
         Repository\
            Current Location
```

This registry key must contain either a DBQ keyword-value pair, SERVER keyword-value pair, DSN keyword-value pair, or just the path to a Jet repository database. The default value for this registry key is:

```
windowsDirectory\MsApps\Repostry\Repostry.mdb
```

where *windowsDirectory* is the path specification for the directory that contains either the Windows NT or Windows 95 installation. Unless you change this registry key value after installing Microsoft Repository, your default database server will be Microsoft Jet.

Note Microsoft Repository will create a default repository database for you when it is installed. This database is managed by Microsoft Jet. It's location is determined by the default value of the *Current*

Location registry key.

Repository CreateObject Method

See Also

This method creates a new repository object of the specified type.

Syntax

Set *variable* **=** *object***.CreateObject(** *typeId, objectId* **)**

The **CreateObject** method syntax has these parts:

Part	Description
variable	Declared as a RepositoryObject. Receives the new repository object.
object	The object that represents the open repository instance through which this program is interacting with the repository.
typeId	The type of the new object; that is, the object identifier of the object definition to which the new object conforms.
objectId	The object identifier to be assigned to the new object. Either pass in ObjID_NULL or do not supply this optional parameter to have the repository assign an object identifier for you.

Repository FreeConnection Method

See Also

This method frees an ODBC connection handle.

This property is not attached to the default interface for the Repository Automation object; it is attached to the **IRepositoryODBC** interface. For details on how to access a member of an interface that is not the default interface, see Accessing Automation Object Members.

Syntax

Call *object*.**FreeConnection(** *hdbc* **)**

The **FreeConnection** method syntax has these parts:

Part	Description
object	The object that represents the open repository instance through which this program is interacting with the repository.
hdbc	The ODBC connection handle to be released.

Remarks

Use this method to free the handle obtained via either the **ReposConnection** property or the **GetNewConnection** method before releasing your open repository instance.

Repository GetNewConnection Method

See Also

This method obtains a new ODBC connection handle using the same ODBC connection string that the repository engine is using to access the repository database. Using a new ODBC connection handle isolates you from changes made by the repository engine.

This property is not attached to the default interface for the Repository Automation object; it is attached to the **IRepositoryODBC** interface. For details on how to access a member of an interface that is not the default interface, see Accessing Automation Object Members.

Syntax

variable = *object*.**GetNewConnection**

The **GetNewConnection** method syntax has these parts:

Part	Description
variable	Receives the new connection handle.
object	The object that represents the open repository instance through which this program is interacting with the repository.

Remarks

Be sure to free the handle obtained via this method before releasing your open repository instance. To free the connection handle, use the **FreeConnection** method.

Repository InternalIDToObjectID Method

See Also

This method translates an internal identifier into an object identifier. Internal identifiers are used by the repository engine to identify repository objects.

Syntax

variable = *object*.**InternalIDToObjectID(** *internalId* **)**

The **InternalIDToObjectID** method syntax has these parts:

Part	Description
variable	Receives the object identifier.
object	The object that represents the open repository instance through which this program is interacting with the repository.
internalId	The internal identifier to be converted.

Remarks

Repository object identifiers are globally unique, and are the same across repositories for the same object. Internal identifiers are unique only within the scope of a single repository.

The translation performed by this method is performed without loading the object in question. This enables database queries involving an object or relationship type identifier to be constructed without having to load the definition object.

Repository Object Property

See Also

Use this property to retrieve a particular repository object. This is a read-only property.

Syntax

Set *variable* **=** *object***.Object(** *objectId* **)**

The **Object** property syntax has these parts:

Part	Description
variable	Declared as a RepositoryObject. Receives the repository object.
object	The object that represents the open repository instance through which this program is interacting with the repository.
objectId	The object identifier for the repository object to be retrieved.

Repository ObjectIDToInternalID Method

See Also

This method translates an object identifier into an internal identifier. Internal identifiers are used by the repository engine to identify repository objects.

Syntax

variable = *object*.**ObjectIDToInternalID(** *objectId* **)**

The **ObjectIDToInternalID** method syntax has these parts:

Part	Description
variable	Receives the internal identifier.
object	The object that represents the open repository instance through which this program is interacting with the repository.
objectId	The object identifier to be converted.

Remarks

Repository object identifiers are globally unique, and are the same across repositories for the same object. Internal identifiers are unique only within the scope of a single repository.

The translation performed by this method is performed without loading the object in question. This enables database queries involving an object or relationship type identifier to be constructed without having to load the definition object.

Repository Open Method

See Also

Use this method to open (connect to) a repository. The root repository object is passed back to the caller.

Syntax

Set *variable* **=** *object*.**Open(** *connect, user, password, flags* **)**

The **Open** method syntax has these parts:

Part	Description
variable	A variable declared as a RepositoryObject. Receives the root repository object for the repository.
object	The instance of the Repository class that you are using to connect to the repository.
connect	The ODBC connection string to be used for accessing the database server that hosts your repository.
user	The user name to use for identification to the database server.
password	The password that matches the *user* input parameter.
flags	Flags that determine database access and caching behavior for the open repository. For details, see the ConnectionFlags Enumeration.

Repository Refresh Method

See Also

This method refreshes all of the cached data for this open repository instance. Only cached data that has not been changed *by the current process* is refreshed.

Syntax

Call *object*.**Refresh(** *milliSeconds* **)**

The **Refresh** method syntax has these parts:

Part	Description
object	The object that represents the open repository instance through which this program is interacting with the repository.
milliSecs	All unchanged repository data that has been in the cache for longer than *millisecs* milliseconds is refreshed. Set to zero to refresh all unchanged data for the repository.

Repository ReposConnection Property

See Also

This property contains the ODBC connection handle that the repository engine is using to access the repository database. This is a read-only property.

This property is not attached to the default interface for the Repository Automation object; it is attached to the **IRepositoryODBC** interface. For details on how to access a member of an interface that is not the default interface, see Accessing Automation Object Members.

Syntax

object.**ReposConnection**

The **ReposConnection** property syntax has these parts:

Part	Description
object	The object that represents the open repository instance through which this program is interacting with the repository.

Remarks

If you use the repository engine's ODBC connection handle, you are not isolated from changes made by the repository engine. For example, uncommitted changes made by the repository engine will be visible to your application.

When using the repository engine's ODBC connection handle, you must not change the state of the handle in a way that is incompatible with the repository engine. Specifically, do not:

- Change any ODBC connection options.
- Perform any accesses concurrent with repository method invocations.
- Directly commit or rollback a database transaction. The IRepositoryTransaction interface must always be used to manage transactions.

Be sure to free the handle obtained via this method before releasing your open repository instance. To free the connection handle, use the **FreeConnection** method.

Repository RootObject Property

See Also

This property is the root repository object for the currently open repository. This is a read-only property.

Syntax

Set *variable* **=** *object***.RootObject**

The **RootObject** property syntax has these parts:

Part	Description
variable	A variable declared as a RepositoryObject. Receives the root repository object.
object	The object that represents the open repository instance through which this program is interacting with the repository.

Repository Transaction Property

See Also

This property is the RepositoryTransaction object for the open repository instance. This is a read-only property.

Syntax

Set *variable* **=** *object***.Transaction**

The **Transaction** property syntax has these parts:

Part	Description
variable	A variable declared as an Object. Receives the RepositoryTransaction object for this repository instance.
object	The object that represents the open repository instance through which this program is interacting with the repository.

Remarks

You can gain access to the RepositoryTransaction object by using the syntax shown above. Then you can access the properties and methods of the RepositoryTransaction object via *variable.method* and *variable.property* syntax. You can also access the properties and methods of the RepositoryTransaction object directly, using syntax like:

Call *object***.Transaction.***method*

Or:

variable **=** *object***.Transaction.***property*

See the **RepositoryTransaction** object for details on the methods and properties that it provides.

RepositoryObject Object

See Also Properties Methods Collections

A repository object is an object that is stored in the repository database, and is managed by the repository engine. A repository object is also an Automation object.

When to Use

Use the **RepositoryObject** object to manipulate the properties of a repository object, to delete a repository object, or to refresh the cached image of a repository object.

Properties

Property	Description
Interface	The specified object interface.
InternalID	The internal identifier that the repository uses to refer to the repository object.
Name	The name of the repository object.
ObjectID	The object identifier for the repository object.
Repository	The open repository instance through which this repository object was instantiated.
Type	The type of the repository object.

Methods

Method	Description
Delete	Deletes a repository object.
Lock	Locks the repository object.
Refresh	Refreshes the cached image of a repository object.

Collections

Collection	Description
Properties	The collection of all of the properties that are attached to the repository object.

RepositoryObject Delete Method

See Also

This method deletes the repository object from the repository. Any relationships that connect the object to other objects are deleted. If the repository object is an origin object of a relationship collection, and the relationship type indicates that deletes are to be propagated, then all of the destination objects are also deleted.

Syntax

Call *object*.**Delete**

The **Delete** method syntax has these parts:

Part	Description
object	An object expression that evaluates to a RepositoryObject object.

RepositoryObject Interface Property

See Also

Use this property to obtain a view of the repository object that uses an alternate interface as the default interface. This is a read-only property. There are two variations of this property.

Syntax

Set *variable* **=** *object***.Interface(** *interfaceId* **)**

Set *variable* **=** *object***.Interface(** *objectId* **)**

The **Interface** property syntax has these parts:

Part	Description
variable	An object variable. Receives the repository object with the specified interface as the default interface.
object	An object expression that evaluates to a RepositoryObject object.
interfaceId	The interface identifier for the interface to be retrieved.
objectId	The object identifier for the interface definition to which the interface to be retrieved conforms.

RepositoryObject InternalID Property

See Also

This property is the internal identifier that the repository engine uses to refer to this object. The internal identifier is unique within the repository, but not unique across repositories. This is a read-only property. To copy this property to another variable, use a variable declared as a Variant.

Syntax

object.**InternalID**

The **InternalID** property syntax has these parts:

Part	Description
object	An object expression that evaluates to a RepositoryObject object.

RepositoryObject Lock Method

See Also

Use this method to lock the repository object. Locking the object prevents other processes from updating the object while you are working with it. The lock is released when you end the current transaction.

Call *object*.**Lock**

The **Lock** method syntax has these parts:

Part	Description
object	An object expression that evaluates to a RepositoryObject object.

RepositoryObject Name Property

See Also

A character string that contains the name of the repository object.

The name property is normally a property of the relationship for which this repository object is the destination object. When the name is retrieved, the name from the first naming relationship found is returned. If the object is not the destination of any naming relationship, a null name is returned. When the name is set, the new name is set for all naming relationships for which the object is the destination.

Syntax

object.**Name**

The **Name** property syntax has these parts:

Part	Description
object	An object expression that evaluates to a RepositoryObject object.

Remarks

If the repository object exposes the INamedObject interface, then the name that is retrieved is always the Name property of the INamedObject interface. Likewise, when this property is set, the Name property of the INamedObject interface *and* the name associated with all naming relationships are set to the new value.

RepositoryObject ObjectID Property

See Also

This property is the object identifier for the repository object. The object identifier is unique across all repositories. This is a read-only property. To copy this property to another variable, use a variable declared as a Variant.

Syntax

object.**ObjectID**

The **ObjectID** property syntax has these parts:

Part	Description
object	An object expression that evaluates to a RepositoryObject object.

RepositoryObject Refresh Method

See Also

This method refreshes the cached image of the repository object. Only cached data that has not been changed by the current process is refreshed.

Syntax

Call *object*.**Refresh(** *milliSecs* **)**

The **Refresh** method syntax has these parts:

Part	Description
object	An object expression that evaluates to a RepositoryObject object.
milliSecs	All unchanged data relating to the item that has been in the cache for longer than *millisecs* milliseconds is refreshed. Set this parameter to zero to refresh all unchanged data.

RepositoryObject Repository Property

See Also

The **Repository** property is the open repository instance through which this repository object was instantiated. This is a read-only property.

Syntax

Set *variable* **=** *object***.Repository**

The **Repository** property syntax has these parts:

Part	Description
variable	A variable declared as an instance of the Repository class. Receives the object that represents the open repository instance.
object	An object expression that evaluates to a RepositoryObject object.

RepositoryObject Properties Collection

See Also

The **Properties** collection contains all of the persistent properties that are attached to the repository object via the IRepositoryObject interface.

Syntax

Set *variable* = *object*.**Properties(** *index* **)**

The **Properties** collection syntax has these parts:

Part	Description
variable	A variable declared as a ReposProperty object. Receives the specified property.
object	An object expression that evaluates to a RepositoryObject object.
index	An integer index that identifies which property in the collection is to be addressed. The valid range is from one to the number of elements in the collection. The number of elements in the collection is specified by *object*.Properties.Count.

RepositoryObject Type Property

See Also

This property specifies the type of the repository object. More specifically, it is the object identifier of the object definition object for the repository object. **Type** is a read-only property. To copy this property to another variable, use a variable declared as a Variant.

Syntax

object.**Type**

The **Type** property syntax has these parts:

Part	Description
object	An object expression that evaluates to a RepositoryObject object.

Remarks

An object in the repository is simultaneously a repository object, an Automation object, *and* an object of a model-specific type, as defined by a tool information model.

The model-specific type of a repository object is defined by its object definition. Each object definition is itself a repository object, and consequently has an object identifier assigned to it. In other words, the value of an object definition object's **ObjectID** property is used as the value of the **Type** property for all repository objects that conform to that object definition.

RepositoryTransaction Object

The repository supports the bracketing of multiple changes within the scope of a transaction. Changes to a repository that are bracketed within a transaction are either all committed or all undone, depending upon the way that the transaction is completed. Repository methods that are reading data from the repository may be executed outside of a transaction, but methods that write data must be bracketed within a transaction.

You cannot directly instantiate a RepositoryTransaction object. When you connect to a repository, a RepositoryTransaction object is created for you. It is accessible via the **Repository.Transaction** property.

When to Use

Use the RepositoryTransaction object to manage repository transactions.

Properties

Property	Description
Status	The transaction status of the repository.

Methods

Method	Description
Abort	Cancels the current transaction.
Begin	Begins a new transaction.
Commit	Commits the current transaction.
Flush	Flushes uncommitted changes to the repository database.
GetOption	Retrieves various transaction options.
SetOption	Sets various transaction options.

Remarks

Only one transaction may be active at a time for each opened repository instance. Nesting of Begin/Commit method invocations is permitted, but no actual nesting of transactions occurs.

RepositoryTransaction Abort Method

See Also

This method cancels the currently active transaction for an open repository. All updates made during the transaction are undone. The nested transaction count is set to zero.

Syntax

Call *object*.**Abort**

The **Abort** method syntax has these parts:

Part	Description
object	The RepositoryTransaction object for the currently open repository instance.

RepositoryTransaction Begin Method

See Also

This method increments the nested transaction count by one. If there is no active transaction, this method begins a transaction for the open repository instance.

Syntax

Call *object*.**Begin**

The **Begin** method syntax has these parts:

Part	Description
object	The RepositoryTransaction object for the currently open repository instance.

RepositoryTransaction Commit Method

See Also

This method decrements the nested transaction count for an open repository instance. If the currently active transaction is not nested, all changes made to repository data within the transaction are committed to the repository database. A transaction is not nested if the nested transaction count equals one.

Syntax

Call *object*.**Commit**

The **Commit** method syntax has these parts:

Part	Description
object	The RepositoryTransaction object for the currently open repository instance.

RepositoryTransaction Flush Method

See Also

This method flushes cached changes to the repository database.

Unless you have set the *exclusive-write-through-mode* transaction option, changes that you make within the scope of a transaction are cached, and are not written to the database until the transaction is committed. If a concurrent SQL query is run against the repository database, the result of the query will not reflect the uncommitted changes (this is normally the desired behavior).

If your repository application must be able to see uncommitted changes in SQL queries, you can use the Flush method to write uncommitted changes to the repository database. All changes made within the scope of the current transaction are flushed. Flushing uncommitted changes does not affect your ability to cancel a transaction via the **Abort** method.

Syntax

Call *object*.**Flush**

The **Flush** method syntax has these parts:

Part	Description
object	The RepositoryTransaction object for the currently open repository instance.

RepositoryTransaction GetOption Method

See Also

This method is used to retrieve various transaction options. The transaction options are:

1. Whether or not subsequent transactions will execute in non-exclusive write-back mode. Non-exclusive write-back mode allows transactions for other repository instances to execute concurrently, and caches updates for this repository instance until a transaction is committed.
2. Whether or not subsequent transactions will execute in exclusive write-back mode. Exclusive write-back mode does not allow transactions for other repository instances to execute concurrently, and caches updates for this repository instance until a transaction is committed.
3. Whether or not subsequent transactions will execute in exclusive write-through mode. Exclusive write-through mode does not allow transactions for other repository instances to execute concurrently, and writes updates for this repository instance to persistent storage as soon as possible.
4. What the maximum time is that subsequent transactions will wait to obtain a lock.
5. What the maximum time is that subsequent transactions will wait to begin a transaction.

The first three of these options are mutually exclusive; only one of them can be set at a time. The last option set is the option that is in effect. The fourth and fifth options are independent of the first three, and each other.

Syntax

variable **=** *object*.**GetOption(** *whichOption* **)**

The **GetOption** property syntax has these parts:

Part	Description
object	The RepositoryTransaction object for the currently open repository instance.
whichOption	This parameter specifies which option is to be retrieved or set. For a list of valid values and their meanings, see the TransactionFlags Enumeration.
variable	A variable declared as a Variant. Receives the value of the specified option.

RepositoryTransaction SetOption Method

See Also

This method is used to set various transaction options. The transaction options are:

1. Whether or not subsequent transactions will execute in non-exclusive write-back mode. Non-exclusive write-back mode allows transactions for other repository instances to execute concurrently, and caches updates for this repository instance until a transaction is committed.
2. Whether or not subsequent transactions will execute in exclusive write-back mode. Exclusive write-back mode does not allow transactions for other repository instances to execute concurrently, and caches updates for this repository instance until a transaction is committed.
3. Whether or not subsequent transactions will execute in exclusive write-through mode. Exclusive write-through mode does not allow transactions for other repository instances to execute concurrently, and writes updates for this repository instance to persistent storage as soon as possible.
4. What the maximum time is that subsequent transactions will wait to obtain a lock.
5. What the maximum time is that subsequent transactions will wait to begin a transaction.

The first three of these options are mutually exclusive; only one of them can be set at a time. The last option set is the option that is in effect. The fourth and fifth options are independent of the first three, and each other.

Syntax

Call *object*.**SetOption(** *whichOption, optionValue* **)**

The **SetOption** method syntax has these parts:

Part	Description
object	The RepositoryTransaction object for the currently open repository instance.
whichOption	This parameter specifies which option is to be retrieved or set. For a list of valid values and their meanings, see the TransactionFlags Enumeration.
optionValue	The new value for the option.

RepositoryTransaction Status Property

See Also

This property indicates what the current transaction status is for the repository instance. If the value is zero, no transaction is active. If the value is nonzero, a transaction is active. This is a read-only property. To copy this property to another variable, use a variable that is declared as a Variant.

Syntax

object.**Status**

The **Status** property syntax has these parts:

Part	Description
object	The RepositoryTransaction object for the currently open repository instance.

Remarks

A transaction is considered active until the **Commit** method has successfully executed and the nested transaction count has been decremented to zero. Depending upon the data-flushing capabilities of the underlying database server, the data associated with a committed transaction may or may not be written to the physical storage device when the Commit method returns control to its caller.

ReposProperties Object

A repository properties collection is the set of persistent properties and collections that are attached to a repository object or relationship via a particular interface.

When to Use

Use the **ReposProperties** object to enumerate the collection of repository properties that are attached to a particular repository object or relationship.

Properties

Property	Description
Count	The count of the number of items in the collection.
Item	Retrieves the specified property from the collection.

ReposProperties Count Property

See Also

A long integer that contains the count of the number of items in the collection. This is a read-only property.

Syntax

object.**Count**

The **Count** property syntax has these parts:

Part	Description
object	The repository property collection.

ReposProperties Item Property

See Also

Use **Item** to retrieve a repository property from the collection. This is a read-only property. There are two variations of this property.

Syntax

Set *variable* **=** *object***.Item(** *index* **)**

Set *variable* **=** *object***.Item(** *objName* **)**

The **Item** property syntax has these parts:

Part	Description
variable	An object expression that evaluates to a ReposProperty object. Receives the specified repository property.
object	The repository property collection.
index	The index of the repository property to be retrieved from the collection.
objName	The name associated with the repository property to be retrieved from the collection.

ReposProperty Object

A repository property is a persistent property or collection that is attached to an object.

When to Use

Use the **ReposProperty** object to retrieve the name, type, or value of a repository property, or to set the value of a repository property.

Properties

Property	Description
Name	The name of the property.
Type	The type of the property.
Value	The value of the property.

ReposProperty Name Property

See Also

The name of the repository property. This is a read-only property.

Syntax

object.**Name**

The **Name** property syntax has these parts:

Part	Description
object	An object expression that evaluates to a ReposProperty object.

ReposProperty Type Property

See Also

The type of the repository property; that is, the object identifier of the property definition object to which this repository property conforms. This is a read-only property. Use a Variant variable to receive the **Type** property.

Syntax

object.**Type**

The **Type** property syntax has these parts:

Part	Description
object	An object expression that evaluates to a ReposProperty object.

ReposProperty Value Property

See Also

The value of the repository property. If you do not know the type of the property, use a Variant variable to receive this property value.

Syntax

object.**Value**

object.**Value** = *newValue*

The **Value** property syntax has these parts:

Part	Description
object	An object expression that evaluates to a ReposProperty object.
newValue	An expression that evaluates to a value of the appropriate type for the repository property.

ClassDef Object

See Also Properties Methods Collections

The ClassDef object helps you create tool information models, by adding interfaces to a class. To insert a *new* class definition into a tool information model, use the **ReposTypeLib** object.

To complete a class definition, once you have added all of the interfaces, commit the transaction that brackets your class definition modifications.

A ClassDef object is also a RepositoryObject. In addition to the members described here, ClassDef objects also provide the members that are defined for repository objects.

When to Use

Use the ClassDef object to:

- Add a new or existing interface to a class definition.
- Retrieve the global identifier for the class.
- Access the collection of interfaces that are part of a class definition.

Properties

Property	Description
ClassID	The global identifier of the class.

Methods

Method	Description
AddInterface	Adds an existing interface to the class definition.
CreateInterfaceDef	Creates a new interface and adds it to the class definition.
ObjectInstances	Materializes an object collection containing all of the objects in the repository that conform to this class.

Collections

Collection	Description
Interfaces	The collection of all interfaces that are implemented by the class.
ItemInCollections	This collection is empty for class definitions. It is reserved for future use.
Properties	The collection of all persistent properties that are attached to the ClassDef object.
ReposTypeLibScopes	The collection of all repository type libraries that contain this definition.

ClassDef ClassID Property

See Also

This property contains the global identifier that is assigned to this class. If you copy this property to a variable, declare the variable as a Variant.

Syntax

object.**ClassID**

The **ClassID** property syntax has these parts:

Part	Description
object	An object expression that evaluates to a ClassDef object.

ClassDef AddInterface Method

See Also

The AddInterface method adds an existing interface to the collection of interfaces that are implemented by a particular class.

Syntax

Call *object*.**AddInterface(** *interfaceDef, flag* **)**

The **AddInterface** method syntax has these parts:

Part	Description
object	An object expression that evaluates to a ClassDef object.
interfaceDef	The InterfaceDef definition object for the interface that is to be added to the collection of interfaces that are implemented by this class.
flag	If the interface that you are adding is the default interface for the class, pass in the string "Default". Otherwise, pass in a null string.

ClassDef CreateInterfaceDef Method

See Also

The CreateInterfaceDef method creates a new interface definition, and adds the interface to the collection of interfaces that are implemented by the class.

Syntax

Set *variable* = *object*.**CreateInterfaceDef(** *sObjId, name, interfaceId, ancestor, flag* **)**

The **CreateInterfaceDef** method syntax has these parts:

Part	Description
variable	A variable declared as an InterfaceDef object. Receives the new interface definition.
object	An object expression that evaluates to a ClassDef object.
sObjId	The object identifier to be assigned to the new interface definition object. If this parameter is set to OBJID_NULL, the repository assigns an object identifier for you.
name	The name of the interface that is to be created.
interfaceId	The interface identifier for this interface. If there is none, set this parameter to zero.
ancestor	The InterfaceDef definition object for the interface that is the base interface from which the interface being created is derived.
flag	If the interface that you are creating is the default interface for the class, pass in the string "Default". Otherwise, pass in a null string.

ClassDef ObjectInstances Method

See Also

This method materializes an object collection containing all of the objects in the repository that conform to this class.

Syntax

Set *variable* = *object*.**ObjectInstances**

The **ObjectInstances** method syntax has these parts:

Part	Description
variable	A variable declared as an **ObjectCol** object. Receives the collection of objects that conform to this class.
object	An object expression that evaluates to a ClassDef object.

ClassDef Interfaces Collection

See Also

The collection of all interfaces that are implemented by this class.

Collection Descriptor	Descriptor Value
Relationship Type	Class-Implements-Interface
Source Is Origin	Yes
Minimum Collection Size	Zero
Maximum Collection Size	Many
Sequenced Collection	No
Deletes Propagated	No
Destinations Named	No
Case Sensitive Names	Not applicable
Unique Names	Not applicable

Syntax

Set *variable* **=** *object*.**Interfaces(** *index* **)**

The **Interfaces** collection syntax has these parts:

Part	Description
variable	A variable declared as an InterfaceDef object. Receives the specified interface.
object	An object expression that evaluates to a ClassDef object.
index	An integer index that identifies which element in the collection is to be addressed. The valid range is from one to the number of elements in the collection. The number of elements in the collection is specified by *object*.Interfaces.Count.

ClassDef Properties Collection

See Also

A **Properties** collection contains all of the persistent properties and collections that are attached to an object via a particular interface. The ClassDef object exposes four separate Properties collections. These collections are exposed by:

1. The IClassDef interface (the default).
2. The IReposTypeInfo interface.
3. The IRepositoryObject interface.
4. The IAnnotationalProps interface.

Syntax

Set *variable* = *object*.**Properties(** *index* **)**

The **Properties** collection syntax has these parts:

Part	Description
variable	A variable declared as a ReposProperty object. Receives the specified property.
object	An object expression; evaluates to an object that exposes: • IClassDef, • IReposTypeInfo, • IRepositoryObject, or • IAnnotationalProps as the default interface.
index	An integer index that identifies which property in the collection is to be addressed. The valid range is from one to the number of elements in the collection. The number of elements in the collection is specified by *object*.Properties.Count.

Remarks

For details on how to access a member of an interface that is not the default interface, see Accessing Automation Object Members.

ClassDef ReposTypeLibScopes Collection

See Also

The collection of repository type libraries that contain this definition.

Collection Descriptor	Descriptor Value
Relationship Type	ReposTypeLib-ScopeFor-ReposTypeInfo
Source Is Origin	No
Minimum Collection Size	One
Maximum Collection Size	Many
Sequenced Collection	No
Deletes Propagated	Yes
Destinations Named	Yes
Case Sensitive Names	No
Unique Names	Yes

Syntax

Set *variable* = *object*.**ReposTypeLibScopes(** *index* **)**

The **ReposTypeLibScopes** collection syntax has these parts:

Part	Description
variable	A variable declared as a ReposTypeLib object. Receives the specified repository type library object.
object	An object expression that evaluates to a ClassDef object.
index	An integer index that identifies which element in the collection is to be addressed. The valid range is from one to the number of elements in the collection. The number of elements in the collection is specified by *object*.TypeLibScopes.Count.

CollectionDef Object

A collection type (also referred to as a collection definition) defines how instances of a particular type of collection will behave. The properties of the collection type determine:

- The minimum and maximum number of items in a collection.
- Whether or not the collection type is an origin collection type.
- Whether or not the collection type permits the naming of destination objects, and if so, whether those names are case sensitive, and required to be unique.
- Whether or not the collection type permits the explicit sequencing of items in the collection.
- What happens to related objects when objects or relationships in the collection are deleted.

The kind of relationship that a particular collection type uses to relate objects to each other is determined by its CollectionItem collection. The CollectionItem collection associates a single relationship type to the collection type.

To add a new collection type, use the **InterfaceDef** object.

A CollectionDef object is also a RepositoryObject. In addition to the members described here, CollectionDef objects also provide the members that are defined for repository objects.

When to Use

Use the CollectionDef object to retrieve or modify the properties of a collection type, to determine the kind of relationship that the collection implements, or to determine the interface to which the collection is attached.

Properties

Property	Description
DispatchID	The dispatch identifier to use when accessing an instance of this type of collection.
Flags	Flags that specify details about this collection definition.
IsOrigin	Indicates whether or not collections of this type are origin collections.
MaxCount	The maximum number of target objects that can be contained in a collection of this type.
MinCount	The minimum number of target objects that must be contained in a collection of this type.

Collections

Collection	Description
CollectionItem	The collection of one relationship type that defines the relationship between target objects of this type of collection and a single source object.
Interface	The interface to which this collection definition is attached.
Properties	The collection of all persistent properties

that are attached to the CollectionDef object.

CollectionDef DispatchID Property

See Also

This property contains the dispatch identifier to use when accessing a collection of this type.

This property is not attached to the default interface for the CollectionDef Automation object; it is attached to the **IInterfaceMember** interface. For details on how to access a member of an interface that is not the default interface, see Accessing Automation Object Members.

Syntax

object.**DispatchID**

The **DispatchID** property syntax has these parts:

Part	Description
object	An object expression; evaluates to an object that exposes **IInterfaceMember** as the default interface.

CollectionDef Flags Property

See Also

The CollectionDef object exposes two separate Flags properties. One of these properties is exposed by the default interface, and the other is exposed by the **IInterfaceMember** interface. They are both described here.

The default Flags property determines:

- Whether or not the collection type permits the naming of destination objects, and if so, whether those names are case sensitive, and required to be unique.
- Whether or not the collection type permits the explicit sequencing of items in the collection.
- What happens to related objects when objects or relationships in the collection are deleted.

See the CollectionDefFlags Enumeration for a list of values and their specific purposes.

The IInterfaceDef Flags property is a flag that specifies whether or not the interface member should be visible to Automation queries. See the InterfaceMemberFlags Enumeration for a list of values and their specific purposes.

Syntax

object.**Flags**

The **Flags** property syntax has these parts:

Part	Description
object	An object expression that evaluates to a **CollectionDef** object, for the default Flags property, or: an object expression that evaluates to an object that exposes **IInterfaceMember** as the default interface, for the alternate Flags property.

Remarks

For details on how to access a member of an interface that is not the default interface, see Accessing Automation Object Members.

CollectionDef IsOrigin Property

See Also

This property indicates whether or not collections of this type are origin collections. If you copy this property to a variable, declare the variable as a Boolean.

Syntax

object.**IsOrigin**

The **IsOrigin** property syntax has these parts:

Part	Description
object	An object expression that evaluates to a CollectionDef object.

CollectionDef MaxCount Property

See Also

This property specifies the maximum number of target objects that can be contained in a collection of this type. This property is maintained for informational purposes, and is not enforced by the repository engine.

Syntax

object.**MaxCount**

The **MaxCount** property syntax has these parts:

Part	Description
object	An object expression that evaluates to a CollectionDef object.

CollectionDef MinCount Property

See Also

This property specifies the minimum number of target objects that must be contained in a collection of this type. This property is maintained for informational purposes, and is not enforced by the repository engine.

Syntax

object.**MinCount**

The **MinCount** property syntax has these parts:

Part	Description
object	An object expression that evaluates to a CollectionDef object.

CollectionDef CollectionItem Collection

See Also

The collection of one relationship type that defines the relationship between target objects of this type of collection and a single source object.

Collection Descriptor	Descriptor Value
Relationship Type	Collection-Contains-Items
Source Is Origin	Yes
Minimum Collection Size	Zero
Maximum Collection Size	One
Sequenced Collection	No
Deletes Propagated	No
Destinations Named	No
Case Sensitive Names	Not applicable
Unique Names	Not applicable

Syntax

Set *variable* = *object*.**CollectionItem(** *index* **)**

The **CollectionItem** collection syntax has these parts:

Part	Description
variable	A variable declared as a RelationshipDef object. Receives the specified relationship definition object.
object	An object expression that evaluates to a CollectionDef object.
index	An integer index that identifies which element in the collection is to be addressed. The valid range is from one to the number of elements in the collection. The number of elements in the collection is specified by *object*.CollectionItem.Count.

CollectionDef Interface Collection

See Also

For a particular collection definition, the Interface collection specifies which interface exposes a member of this type.

This collection is not attached to the default interface for the CollectionDef Automation object; it is attached to the **IInterfaceMember** interface. For details on how to access a member of an interface that is not the default interface, see Accessing Automation Object Members.

Collection Descriptor	Descriptor Value
Relationship Type	Interface-Has-Members
Source Is Origin	No
Minimum Collection Size	One
Maximum Collection Size	One
Sequenced Collection	Yes
Deletes Propagated	Yes
Destinations Named	Yes
Case Sensitive Names	No
Unique Names	Yes

Syntax

Set *variable* = *object*.**Interface(** *index* **)**

The **Interface** collection syntax has these parts:

Part	Description
variable	A variable declared as an InterfaceDef object. Receives the specified interface definition.
object	An object expression; evaluates to an object that implements **IInterfaceMember** as the default interface.
index	An integer index that identifies which element in the collection is to be addressed. The valid range is from one to the number of elements in the collection. The number of elements in the collection is specified by *object*.Interface.Count.

CollectionDef Properties Collection

See Also

A **Properties** collection contains all of the persistent properties and collections that are attached to an object via a particular interface. The CollectionDef object exposes four separate Properties collections. These collections are exposed by:

1. The ICollectionDef interface (the default).
2. The IInterfaceMember interface.
3. The IRepositoryObject interface.
4. The IAnnotationalProps interface.

Syntax

Set *variable* = *object*.**Properties(** *index* **)**

The **Properties** collection syntax has these parts:

Part	Description
variable	A variable declared as a ReposProperty object. Receives the specified property.
object	An object expression; evaluates to an object that exposes: • ICollectionDef, • IInterfaceMember, • IRepositoryObject, or • IAnnotationalProps as the default interface.
index	An integer index that identifies which property in the collection is to be addressed. The valid range is from one to the number of elements in the collection. The number of elements in the collection is specified by *object*.Properties.Count.

Remarks

For details on how to access a member of an interface that is not the default interface, see Accessing Automation Object Members.

InterfaceDef Object

See Also Properties Methods Collections

The properties methods, and collections that a class implements are organized into functionally related groups. Each group is implemented as a repository interface. The properties, methods, and collections of each interface are *members* of the interface. An interface definition is the template to which an interface conforms.

To add a new interface to the repository, use the **ClassDef** object or the **ReposTypeLib** object.

An InterfaceDef object is also a RepositoryObject. In addition to the members described here, InterfaceDef objects also provide the members that are defined for repository objects.

When to Use

Use the InterfaceDef class to:

- Retrieve or modify properties of an interface definition.
- Determine which members are attached to an interface definition.
- Determine which classes implement an interface.
- Determine the base interface from which an interface derives.
- Determine what interfaces derive from a particular interface.
- Determine what repository objects expose a particular interface.
- Add a new property, method or collection type to an interface definition.

Properties

Property	Description
Flags	Flags that specify whether the interface is extensible, and whether the interface should be visible to Automation interface queries.
InterfaceID	The global interface identifier for the interface.
TableName	The name of the SQL table that is used to store instance information for the properties of the interface.

Methods

Method	Description
CreateMethodDef	Creates a new method definition, and attaches it to the interface definition.
CreatePropertyDef	Creates a new property definition, and attaches it to the interface definition.
CreateRelationshipColDef	Creates a relationship collection type. The collection type is attached to the interface definition.
ObjectInstances	Materializes an **ObjectCol** collection of all objects in the repository that expose this interface.

Collections

Collection	Description
Ancestor	The collection of one base interface from which this interface derives.
Classes	The collection of classes that implement the interface.
Descendants	The collection of other interfaces that derive from this interface.
Members	The collection of members that are exposed by the interface.
ItemInCollections	This collection is empty for interface definitions.
Properties	The collection of all persistent properties that are attached to the InterfaceDef object.
ReposTypeLibScopes	The collection of all repository type libraries that contain this definition.

InterfaceDef Flags Property

See Also

This property contains flags that specify whether the interface is extensible, and whether the interface should be visible to Automation interface queries. See the InterfaceDefFlags Enumeration for a list of values and their specific purposes.

Syntax

object.**Flags**

The **Flags** property syntax has these parts:

Part	Description
object	An object expression that evaluates to an InterfaceDef object.

InterfaceDef InterfaceID Property

See Also

This property is the global interface identifier for the interface. If you copy this property to a variable, declare the variable as a Variant.

Syntax

object.**InterfaceID**

The **InterfaceID** property syntax has these parts:

Part	Description
object	An object expression that evaluates to an InterfaceDef object.

InterfaceDef TableName Property

See Also

This character string property contains the name of the SQL table that is used to store instance information for the properties of the interface. The length of the name must be 30 characters or less.

Syntax

object.**TableName**

The **TableName** property syntax has these parts:

Part	Description
object	An object expression that evaluates to an InterfaceDef object.

InterfaceDef CreateMethodDef Method

See Also

This method creates a new method definition and attaches it to the interface definition.

Syntax

Set *variable* **=** *object***.CreateMethodDef(** *sObjId, name, dispId* **)**

The **CreateMethodDef** method syntax has these parts:

Part	Description
object	An object expression that evaluates to an InterfaceDef object.
variable	A variable declared as a MethodDef object. Receives the new method definition.
sObjId	The object identifier to be used for the new method definition object. The repository engine will assign an object identifier if you set this parameter to OBJID_NULL.
name	The name of the new method.
dispId	The dispatch identifier to be used for accessing the new method.

InterfaceDef CreatePropertyDef Method

See Also

This method creates a new property definition and attaches it to the interface definition.

Syntax

Set *variable* = *object*.**CreatePropertyDef(** *sObjId, name, dispId, CType* **)**

The **CreatePropertyDef** method syntax has these parts:

Part	Description
object	An object expression that evaluates to an InterfaceDef object.
variable	A variable declared as a PropertyDef object. Receives the new property definition.
sObjId	The object identifier to be used for the new property definition object. The repository engine will assign an object identifier if you set this parameter to OBJID_NULL.
name	The name of the new property.
dispId	The dispatch identifier to be used for accessing the new property.
CType	The C data type of the property. For a definition of valid values, see the *ODBC Programmer's Reference*.

InterfaceDef CreateRelationshipColDef Method

See Also

This method creates a new collection type, attaches it to this interface, and associates it with the specified relationship type.

Syntax

Set *variable* = *object*.**CreateRelationshipColDef(** *sObjId, name, dispId, isOrigin, flags, relshipDef* **)**

The **CreateRelationshipColDef** method syntax has these parts:

Part	Description
object	An object expression that evaluates to an InterfaceDef object.
variable	A variable declared as a CollectionDef object. Receives the new collection definition.
sObjId	The object identifier for the collection type. The repository engine will assign an object identifier if you set this parameter to OBJID_NULL.
name	The name of the new collection type.
dispId	The dispatch identifier to be used for Automation access to collections of this type.
isOrigin	Specifies whether collections of this type are origin collections. This is a Boolean parameter.
flags	Flags that specify naming, sequencing, and delete propagation behavior for the collection type. See the CollectionDefFlags Enumeration for a list of values and their specific purposes.
relshipDef	The relationship definition object to which this collection type is connected.

Remarks

By default, the collection definition specifies that zero to many items are permitted in collections of this type. To specify a different minimum and maximum item count for the new collection type, change the MinCount and MaxCount properties *before committing the transaction* that contains this method invocation.

InterfaceDef ObjectInstances Method

See Also

This method materializes an ObjectCol collection of all objects in the repository that expose this interface.

Syntax

Set *variable* = *object*.**ObjectInstances**

The **ObjectInstances** method syntax has these parts:

Part	Description
object	An object expression that evaluates to an InterfaceDef object.
variable	A variable declared as an **ObjectCol** object. Receives the collection of objects that expose this interface.

InterfaceDef Ancestor Collection

See Also

This collection specifies the one base interface from which this interface derives.

Collection Descriptor	Descriptor Value
Relationship Type	Interface-InheritsFrom-Interface
Source Is Origin	Yes
Minimum Collection Size	One
Maximum Collection Size	One
Sequenced Collection	No
Deletes Propagated	No
Destinations Named	No
Case Sensitive Names	Not applicable
Unique Names	Not applicable

Syntax

Set *variable* = *object*.**Ancestor(** *index* **)**

The **Ancestor** collection syntax has these parts:

Part	Description
variable	A variable declared as an InterfaceDef object. Receives the specified base interface definition.
object	An object expression that evaluates to an InterfaceDef object.
index	An integer index that identifies which element in the collection is to be addressed. The valid range is from one to the number of elements in the collection. The number of elements in the collection is specified by *object*.Ancestor.Count.

InterfaceDef Classes Collection

See Also

This collection specifies which classes implement the interface.

Collection Descriptor	Descriptor Value
Relationship Type	Class-Implements-Interface
Source Is Origin	No
Minimum Collection Size	Zero
Maximum Collection Size	Many
Sequenced Collection	No
Deletes Propagated	No
Destinations Named	No
Case Sensitive Names	Not Applicable
Unique Names	Not Applicable

Syntax

Set *variable* **=** *object***.Classes(** *index* **)**

The **Classes** collection syntax has these parts:

Part	Description
variable	A variable declared as a ClassDef object. Receives the specified class definition.
object	An object expression that evaluates to an InterfaceDef object.
index	An integer index that identifies which element in the collection is to be addressed. The valid range is from one to the number of elements in the collection. The number of elements in the collection is specified by *object*.Classes.Count.

InterfaceDef Descendants Collection

See Also

This collection specifies other interfaces that derive from this interface..

Collection Descriptor	Descriptor Value
Relationship Type	Interface-InheritsFrom-Interface
Source Is Origin	No
Minimum Collection Size	Zero
Maximum Collection Size	Many
Sequenced Collection	No
Deletes Propagated	No
Destinations Named	No
Case Sensitive Names	Not applicable
Unique Names	Not applicable

Syntax

Set *variable* **=** *object*.**Descendants(** *index* **)**

The **Descendants** collection syntax has these parts:

Part	Description
variable	A variable declared as an InterfaceDef object. Receives the specified interface definition.
object	An object expression that evaluates to an InterfaceDef object.
index	An integer index that identifies which element in the collection is to be addressed. The valid range is from one to the number of elements in the collection. The number of elements in the collection is specified by *object*.Descendants.Count.

InterfaceDef Members Collection

See Also

This collection specifies which members are attached to the interface.

Collection Descriptor	Descriptor Value
Relationship Type	Interface-Has-Members
Source Is Origin	Yes
Minimum Collection Size	Zero
Maximum Collection Size	Many
Sequenced Collection	Yes
Deletes Propagated	Yes
Destinations Named	Yes
Case Sensitive Names	No
Unique Names	Yes

Syntax

Set *variable* = *object*.**Members(** *index* **)**

The **Members** collection syntax has these parts:

Part	Description
variable	A variable declared as an object. Receives the specified property definition, method definition, or collection definition.
object	An object expression that evaluates to an InterfaceDef object.
index	An integer index that identifies which element in the collection is to be addressed. The valid range is from one to the number of elements in the collection. The number of elements in the collection is specified by *object*.Members.Count.

InterfaceDef Properties Collection

See Also

A **Properties** collection contains all of the persistent properties and collections that are attached to an object via a particular interface. The InterfaceDef object exposes four separate Properties collections. These collections are exposed by:

1. The IInterfaceDef interface (the default).
2. The IReposTypeInfo interface.
3. The IRepositoryObject interface.
4. The IAnnotationalProps interface.

Syntax

Set *variable* = *object*.**Properties(** *index* **)**

The **Properties** collection syntax has these parts:

Part	Description
variable	A variable declared as a ReposProperty object. Receives the specified property.
object	An object expression; evaluates to an object that exposes: • IInterfaceDef, • IReposTypeInfo, • IRepositoryObject, or • IAnnotationalProps as the default interface.
index	An integer index that identifies which property in the collection is to be addressed. The valid range is from one to the number of elements in the collection. The number of elements in the collection is specified by *object*.Properties.Count.

Remarks

For details on how to access a member of an interface that is not the default interface, see Accessing Automation Object Members.

InterfaceDef ReposTypeLibScopes Collection

See Also

The collection of repository type libraries that contain this definition.

Collection Descriptor	Descriptor Value
Relationship Type	ReposTypeLib-ScopeFor-ReposTypeInfo
Source Is Origin	No
Minimum Collection Size	One
Maximum Collection Size	Many
Sequenced Collection	No
Deletes Propagated	Yes
Destinations Named	Yes
Case Sensitive Names	No
Unique Names	Yes

Syntax

Set *variable* = *object*.**ReposTypeLibScopes(** *index* **)**

The **ReposTypeLibScopes** collection syntax has these parts:

Part	Description
variable	A variable declared as a ReposTypeLib object. Receives the specified repository type library object.
object	An object expression that evaluates to an InterfaceDef object.
index	An integer index that identifies which element in the collection is to be addressed. The valid range is from one to the number of elements in the collection. The number of elements in the collection is specified by *object*.TypeLibScopes.Count.

MethodDef Object

See Also Properties Methods Collections

When you define a class for a tool information model, you specify the interfaces that the class implements. For each of those interfaces, you specify the *members* (properties, methods, and collections) that are attached to the interface.

The definition of a method as a member of an interface does not result in the method's implementation logic being stored in the repository. However, it does add the method name to the set of defined member names for that interface. It also reserves the method's dispatch identifier in the set of defined dispatch identifier values for the interface.

A method definition (a **MethodDef** object) is also a RepositoryObject. In addition to the members described here, MethodDef objects also provide the members that are defined for repository objects.

To attach a new method to an interface, use the **CreateMethodDef** method of the **InterfaceDef** object.

When to Use

Use the MethodDef object to access or modify the characteristics of a method definition, or to determine the interface definition to which a particular method is attached.

Properties

Property	Description
DispatchID	The dispatch identifier to use when invoking a method that conforms to this method definition.
Flags	Flags that specify details about this method definition.

Collections

Collection	Description
Interface	The interface to which this method definition is attached.
Properties	The collection of all persistent properties that are attached to the MethodDef object.

MethodDef DispatchID Property

See Also

This property contains the dispatch identifier that is used to invoke a method that conforms to this method definition.

Syntax

object.**DispatchID**

The **DispatchID** property syntax has these parts:

Part	Description
object	An object expression that evaluates to a MethodDef object.

MethodDef Flags Property

See Also

This property is a flag that specifies whether or not the interface member should be visible to Automation queries. See the InterfaceMemberFlags Enumeration for a list of values and their specific purposes.

Syntax

object.**Flags**

The **Flags** property syntax has these parts:

Part	Description
object	An object expression that evaluates to a MethodDef object.

MethodDef Interface Collection

See Also

For a particular method definition, the Interface collection specifies which interface exposes a member of this type.

Collection Descriptor	Descriptor Value
Relationship Type	Interface-Has-Members
Source Is Origin	No
Minimum Collection Size	One
Maximum Collection Size	One
Sequenced Collection	Yes
Deletes Propagated	Yes
Destinations Named	Yes
Case Sensitive Names	No
Unique Names	Yes

Syntax

Set *variable* = *object*.**Interface(** *index* **)**

The **Interface** collection syntax has these parts:

Part	Description
variable	A variable declared as an InterfaceDef object. Receives the specified interface definition.
object	An object expression that evaluates to a MethodDef object.
index	An integer index that identifies which element in the collection is to be addressed. The valid range is from one to the number of elements in the collection. The number of elements in the collection is specified by *object*.Interface.Count.

MethodDef Properties Collection

See Also

A **Properties** collection contains all of the persistent properties and collections that are attached to an object via a particular interface. The MethodDef object exposes three separate Properties collections. These collections are exposed by:

1. The IInterfaceMember interface (the default).
2. The IRepositoryObject interface.
3. The IAnnotationalProps interface.

Syntax

Set *variable* = *object*.**Properties(** *index* **)**

The **Properties** collection syntax has these parts:

Part	Description
variable	A variable declared as a ReposProperty object. Receives the specified property.
object	An object expression; evaluates to an object that exposes: • IInterfaceMember, • IRepositoryObject, or • IAnnotationalProps as the default interface.
index	An integer index that identifies which property in the collection is to be addressed. The valid range is from one to the number of elements in the collection. The number of elements in the collection is specified by *object*.Properties.Count.

Remarks

For details on how to access a member of an interface that is not the default interface, see Accessing Automation Object Members.

PropertyDef Object

See Also Properties Methods Collections

When you define a class for a tool information model, you specify the interfaces that the class implements. For each of those interfaces, you specify the *members* (properties, methods, and collections) that are attached to the interface.

In order to attach a property to an interface, a property definition must exist for the property. The characteristics of the property (it's name, dispatch identifier, data type, and various storage details) are stored in the property definition object. These characteristics are defined by the properties *of the property definition object*.

To create a new property definition:

1. Use the **CreatePropertyDef** method of the **InterfaceDef** object.
2. Define any non-default characteristics of your new property definition by manipulating the properties of the property definition object.
3. Commit your changes to the repository database.

A PropertyDef object is also a RepositoryObject. In addition to the members described here, PropertyDef objects also provide the members that are defined for repository objects.

When to Use

Use the PropertyDef object to retrieve or modify the characteristics of a property definition, or to determine which interface exposes a particular property.

Properties

Property	Description
APIType	The C data type of the property.
ColumnName	The name of the column in the SQL table for this property.
DispatchID	The dispatch identifier to use when accessing an instance of this type of property.
Flags	Flags that specify details about this property definition.
SQLScale	The number of digits to the right of the decimal point for a numeric property.
SQLSize	The size in bytes of the property.
SQLType	The SQL data type of the property.

Collections

Collection	Description
Interface	The interface to which this property definition is attached.
Properties	The collection of all persistent properties that are attached to the PropertyDef object.

PropertyDef APIType Property

See Also

The C data type of the property. For a definition of valid values, see the *ODBC Programmer's Reference*.

Syntax

object.**APIType**

The **APIType** property syntax has these parts:

Part	Description
object	An object expression that evaluates to a PropertyDef object.

PropertyDef ColumnName Property

See Also

An SQL table is used to store instance information for the properties of an interface. By default, there is a column in this table for each property that is defined as a member of the interface. The **ColumnName** string property specifies the name of the column in the SQL table for the property definition. The length of the column name must be 30 bytes or less.

Syntax

object.**ColumnName**

The **ColumnName** property syntax has these parts:

Part	Description
object	An object expression that evaluates to a PropertyDef object.

PropertyDef DispatchID Property

See Also

This property contains the dispatch identifier to use when accessing an instance of this type of member.

This property is not attached to the default interface for the PropertyDef Automation object; it is attached to the **IInterfaceMember** interface. For details on how to access a member of an interface that is not the default interface, see Accessing Automation Object Members.

Syntax

object.**DispatchID**

The **DispatchID** property syntax has these parts:

Part	Description
object	An object expression; evaluates to an object that exposes **IInterfaceMember** as the default interface.

PropertyDef Flags Property

See Also

The PropertyDef object exposes two separate Flags properties. One of these properties is exposed by the default interface, and the other is exposed by the **IInterfaceMember** interface. They are both described here.

1. **The default Flags property:** a flag that specifies whether or not a column is created in the SQL table for the interface to which this property is attached. If no column is created, then instances of this property are only attached to individual objects, when the property value is set for that particular object. By default, a column is created for each property. See the PropertyDefFlags Enumeration for the symbolic and numeric values of this flag.
2. **The IInterfaceDef Flags property:** a flag that specifies whether or not the interface member should be visible to Automation queries. See the InterfaceMemberFlags Enumeration for a list of values and their specific purposes.

Syntax

object.**Flags**

The **Flags** property syntax has these parts:

Part	Description
object	An object expression that evaluates to a **PropertyDef** object, for the default Flags property, or: an object expression that evaluates to an object that exposes **IInterfaceMember** as the default interface, for the alternate Flags property.

Remarks

For details on how to access a member of an interface that is not the default interface, see Accessing Automation Object Members.

PropertyDef SQLScale Property

See Also

The number of digits to the right of the decimal point for a numeric property. This parameter is ignored unless the **SQLType** property specifies an SQL_NUMERIC, SQL_DECIMAL, or SQL_TIME data type.

Syntax

object.**SQLScale**

The **SQLScale** property syntax has these parts:

Part	Description
object	An object expression that evaluates to a PropertyDef object.

PropertyDef SQLSize Property

See Also

The size in bytes of the property. This parameter is ignored when the data type of the property inherently specifies the size of the property.

Syntax

object.**SQLSize**

The **SQLSize** property syntax has these parts:

Part	Description
object	An object expression that evaluates to a PropertyDef object.

PropertyDef SQLType Property

See Also

The SQL data type of the property. For a definition of valid values, see the *ODBC Programmer's Reference*.

Syntax

object.**SQLType**

The **SQLType** property syntax has these parts:

Part	Description
object	An object expression that evaluates to a PropertyDef object.

PropertyDef Interface Collection

See Also

For a particular property definition, the Interface collection specifies which interface exposes a member of this type.

This collection is not attached to the default interface for the PropertyDef Automation object; it is attached to the **IInterfaceMember** interface. For details on how to access a member of an interface that is not the default interface, see Accessing Automation Object Members.

Collection Descriptor	Descriptor Value
Relationship Type	Interface-Has-Members
Source Is Origin	No
Minimum Collection Size	One
Maximum Collection Size	One
Sequenced Collection	Yes
Deletes Propagated	Yes
Destinations Named	Yes
Case Sensitive Names	No
Unique Names	Yes

Syntax

Set *variable* = *object*.**Interface(** *index* **)**

The **Interface** collection syntax has these parts:

Part	Description
variable	A variable declared as an InterfaceDef object. Receives the specified interface definition.
object	An object expression; evaluates to an object that implements **IInterfaceMember** as the default interface.
index	An integer index that identifies which element in the collection is to be addressed. The valid range is from one to the number of elements in the collection. The number of elements in the collection is specified by *object*.Interface.Count.

PropertyDef Properties Collection

See Also

A **Properties** collection contains all of the persistent properties and collections that are attached to an object via a particular interface. The PropertyDef object exposes four separate Properties collections. These collections are exposed by:

1. The IPropertyDef interface (the default).
2. The IRepositoryObject interface.
3. The IInterfaceMember interface.
4. The IAnnotationalProps interface.

Syntax

Set *variable* = *object*.**Properties(** *index* **)**

The **Properties** collection syntax has these parts:

Part	Description
variable	A variable declared as a ReposProperty object. Receives the specified property.
object	An object expression; evaluates to an object that exposes: • IPropertyDef, • IRepositoryObject, • IInterfaceMember, or • IAnnotationalProps as the default interface.
index	An integer index that identifies which property in the collection is to be addressed. The valid range is from one to the number of elements in the collection. The number of elements in the collection is specified by *object*.Properties.Count.

Remarks

For details on how to access a member of an interface that is not the default interface, see Accessing Automation Object Members.

ReposTypeLib Object

See Also Properties Methods Collections

There is one repository type library for every tool information model contained in the repository database. Each tool information model provides a logical grouping of all of the type definitions related to a particular tool (or tool set). Repository type libraries are represented by **ReposTypeLib** objects.

A ReposTypeLib object is also a RepositoryObject. In addition to the members described here, ReposTypeLib objects also provide the members that are defined for repository objects.

To insert a new tool information model into the repository database, use the **ReposRoot** object.

When to Use

Use a ReposTypeLib object to:

- Define new classes, relationship types, and interfaces for a tool information model.
- Retrieve or modify the global identifier associated with a repository type library.
- Determine which type definitions are associated with a particular repository type library.

Properties

Property	Description
TypeLibID	The global identifier for the repository type library.

Methods

Method	Description
CreateClassDef	Creates a new class definition object.
CreateInterfaceDef	Creates a new interface definition object.
CreateRelationshipDef	Creates a new relationship definition object.

Collections

Collection	Description
ReposTypeInfos	The collection of all classes, interfaces, and relationship types that are defined in the repository type library.
ReposTypeLibContexts	The collection of one repository root object that is the context for the repository type library.
Properties	The collection of all persistent properties that are attached to the ReposTypeLib object.

ReposTypeLib TypeLibID Property

See Also

This property is the global identifier for the repository type library. If you copy this property to a variable, declare the variable as a Variant.

Syntax

object.**TypeLibID**

The **TypeLibID** property syntax has these parts:

Part	Description
object	An object expression that evaluates to a ReposTypeLib object.

ReposTypeLib CreateClassDef Method

See Also

This method creates a new class definition object. No interfaces are attached to the class.

Syntax

Set *variable* **=** *object***.CreateClassDef(** *sObjId, Name, sClsId* **)**

The **CreateClassDef** method syntax has these parts:

Part	Description
variable	A variable declared as a ClassDef object. Receives the new class definition.
object	An object expression that evaluates to a ReposTypeLib object.
sObjId	The object identifier to be used for the new class definition object. The repository engine will assign an object identifier if you set this parameter to OBJID_NULL.
Name	The name of the new class.
sClsId	The global identifier by which this class is referenced.

ReposTypeLib CreateInterfaceDef Method

See Also

The CreateInterfaceDef method creates a new interface definition object. Use the **AddInterface** method of the **ClassDef** object to attach the interface to a class definition object.

Syntax

Set *variable* = *object*.**CreateInterfaceDef(** *sObjId, Name, sIId, Ancestor* **)**

The **CreateInterfaceDef** method syntax has these parts:

Part	Description
variable	A variable declared as an InterfaceDef object. Receives the new interface definition.
object	An object expression that evaluates to a ReposTypeLib object.
sObjId	The object identifier to be assigned to the new interface definition object. If this parameter is set to OBJID_NULL, the repository assigns an object identifier for you.
Name	The name of the interface that is to be created.
sIId	The interface identifier associated with the signature for this interface. If there is none, set this parameter to zero.
Ancestor	The base interface from which the new interface is derived.

ReposTypeLib CreateRelationshipDef Method

See Also

This method creates a relationship definition object for a new relationship type. Once the relationship definition is created, use the **CreateRelationshipColDef** method of the **InterfaceDef** object to create origin and destination collection definitions for the new relationship type.

Syntax

Set *variable* = *object*.**CreateRelationshipDef(** *sObjId, Name* **)**

The **CreateRelationshipDef** method syntax has these parts:

Part	Description
variable	A variable declared as a RelationshipDef object. Receives the new relationship definition.
object	An object expression that evaluates to a ReposTypeLib object.
sObjId	The object identifier for the new relationship type. The repository engine will assign an object identifier if you set this parameter to OBJID_NULL.
Name	The name of the new relationship type.

ReposTypeLib ReposTypeInfos Collection

See Also

The collection of all classes, interfaces, and relationship types that are associated with a repository type library. The repository engine uses this collection to enforce the unique naming of all classes, interfaces, and relationship types for a repository type library.

Collection Descriptor	Descriptor Value
Relationship Type	ReposTypeLib-ScopeFor-ReposTypeInfo
Source Is Origin	Yes
Minimum Collection Size	Zero
Maximum Collection Size	Many
Sequenced Collection	No
Deletes Propagated	Yes
Destinations Named	Yes
Case Sensitive Names	No
Unique Names	Yes

Syntax

Set *variable* **=** *object*.**ReposTypeInfos(** *index* **)**

The **ReposTypeInfos** collection syntax has these parts:

Part	Description
variable	A variable declared as an Object. Receives the specified class definition, interface definition, or relationship definition.
object	An object expression that evaluates to a ReposTypeLib object.
index	An integer index that identifies which element in the collection is to be addressed. The valid range is from one to the number of elements in the collection. The number of elements in the collection is specified by *object*.ReposTypeInfos.Count.

ReposTypeLib ReposTypeLibContexts Collection

See Also

The collection of one repository root that is the context for a repository type library.

Collection Descriptor	Descriptor Value
Relationship Type	TlbManager-ContextFor-ReposTypeLibs
Source Is Origin	No
Minimum Collection Size	One
Maximum Collection Size	Many
Sequenced Collection	No
Deletes Propagated	Yes
Destinations Named	Yes
Case Sensitive Names	No
Unique Names	Yes

Syntax

Set *variable* = *object*.**ReposTypeLibContexts(** *index* **)**

The **ReposTypeLibContexts** collection syntax has these parts:

Part	Description
variable	A variable declared as a ReposRoot object. Receives the repository root object.
object	An object expression that evaluates to a ReposTypeLib object.
index	An integer index that identifies which element in the collection is to be addressed. The valid range is from one to the number of elements in the collection. The number of elements in the collection is specified by *object*.ReposTypeLibContexts.Count.

ReposTypeLib Properties Collection

See Also

A **Properties** collection contains all of the persistent properties and collections that are attached to an object via a particular interface. The ReposTypeLib object exposes three separate Properties collections. These collections are exposed by:

1. The IReposTypeLib interface (the default).
2. The IRepositoryObject interface.
3. The IAnnotationalProps interface.

Syntax

Set *variable* = *object*.**Properties(** *index* **)**

The **Properties** collection syntax has these parts:

Part	Description
variable	A variable declared as a ReposProperty object. Receives the specified property.
object	An object expression; evaluates to an object that exposes: • IReposTypeLib, • IRepositoryObject, or • IAnnotationalProps as the default interface.
index	An integer index that identifies which property in the collection is to be addressed. The valid range is from one to the number of elements in the collection. The number of elements in the collection is specified by *object*.Properties.Count.

Remarks

For details on how to access a member of an interface that is not the default interface, see Accessing Automation Object Members.

ReposRoot Object

See Also Properties Methods Collections

There is one root object in each repository. The root object is the starting point for navigating to other objects in the repository. The root object serves as the starting point for both *type* and *instance* data navigations.

- **Type data navigation:**

  When you create a tool information model, the corresponding repository type library is attached to the root object via the ReposTypeLibs collection. This collection can be used to enumerate all of the tool information models (type data) that are contained in the repository.

- **Instance data navigation:**

  Once a tool information model is defined, the repository can be populated with *instance data.* This instance data consists of objects and relationships that conform to the classes and relationship types of the tool information model.

  Because the objects are connected via relationships, you can navigate through this data. However, to enable general purpose repository browsers to navigate this data, the first navigational step must be from the root object of the repository through a root relationship collection to the *primary objects* of your tool information model. Primary objects are objects that make a good starting point for navigating to other objects of your tool information model.

  Because this root relationship collection is different for each tool information model, it must be defined *by the tool information model*. There are two options for attaching this relationship collection to the root object:

  1. The ReposRoot class implements the IReposRoot interface. This interface is provided to tool information model creators as a connection point. You can add your connecting relationship collection to this interface.
  2. You can extend the ReposRoot class to implement a new interface that is defined in your tool information model. This interface implements a relationship collection that attaches the root object to the primary objects in your tool information model.

To facilitate navigation, the root object in all repositories always has *the same object identifier.* The symbolic name for this object identifier is **OBJID_ReposRootObj**.

A ReposRoot object is also a RepositoryObject. In addition to the members described here, ReposRoot objects also provide the members that are defined for repository objects.

When to Use

Use the **ReposRoot** object to:

- Obtain a starting point for navigating to objects in the repository.
- Create a new tool information model.
- Attach a relationship collection to the root object of the repository that connects to the primary objects of your tool information model.
- Determine what tool information models are currently stored in the repository.

Methods

Method	Description
CreateTypeLib	Creates a repository type library for a new tool information model.

Collections

Collection	Description

ReposTypeLibs	The collection of repository type libraries that are currently stored in the repository.
Properties	The collection of all persistent properties that are attached to the ReposRoot object.

ReposRoot CreateTypeLib Method

See Also

This method creates a new repository type library and attaches it to the root of the repository. Each repository type library represents a tool information model.

Syntax

Set *variable* = *object*.**CreateTypeLib(** *sObjId, Name, TypeLibId* **)**

The **CreateTypeLib** method syntax has these parts:

Part	Description
variable	A variable declared as a ReposTypeLib object. Receives the new repository type library.
object	An object expression that evaluates to a ReposRoot object.
sObjId	The object identifier to be used for the new repository type library object. The repository engine will assign an object identifier if you set this parameter to OBJID_NULL.
Name	The name of the new repository type library.
TypeLibId	The global identifier by which this repository type library is referenced.

Remarks

This method *does not* create an external type library; it creates a ReposTypeLib object in the repository database.

ReposRoot ReposTypeLibs Collection

See Also

The collection of repository type libraries that are currently stored in the repository. Each repository type library represents a tool information model.

Collection Descriptor	Descriptor Value
Relationship Type	TlbManager-ContextFor-ReposTypeLibs
Source Is Origin	Yes
Minimum Collection Size	Zero
Maximum Collection Size	Many
Sequenced Collection	No
Deletes Propagated	Yes
Destinations Named	Yes
Case Sensitive Names	No
Unique Names	Yes

Syntax

Set *variable* = *object*.**ReposTypeLibs(** *index* **)**

The **ReposTypeLibs** collection syntax has these parts:

Part	Description
variable	A variable declared as a ReposTypeLib object. Receives the specified repository type library.
object	An object expression that evaluates to a ReposRoot object.
index	An integer index that identifies which element in the collection is to be addressed. The valid range is from one to the number of elements in the collection. The number of elements in the collection is specified by *object*.ReposTypeLibs.Count.

ReposRoot Properties Collection

See Also

A **Properties** collection contains all of the persistent properties and collections that are attached to an object via a particular interface. The ReposRoot object exposes four separate Properties collections. These collections are exposed by:

1. The IManageReposTypeLib interface (the default).
2. The IReposRoot interface.
3. The IRepositoryObject interface.
4. The IAnnotationalProps interface.

Syntax

Set *variable* = *object*.**Properties(** *index* **)**

The **Properties** collection syntax has these parts:

Part	Description
variable	A variable declared as a ReposProperty object. Receives the specified property.
object	An object expression; evaluates to an object that exposes: • IManageReposTypeLib, • IReposRoot, • IRepositoryObject, or • IAnnotationalProps as the default interface.
index	An integer index that identifies which property in the collection is to be addressed. The valid range is from one to the number of elements in the collection. The number of elements in the collection is specified by *object*.Properties.Count.

Remarks

For details on how to access a member of an interface that is not the default interface, see Accessing Automation Object Members.

RelationshipDef Object

When you define a tool information model in the repostory, you define classes of objects, types of relationships that can exist between objects, and various properties that are attached to these object classes and relationship types. The relationship types that you define in your tool information model are represented by instances of the **RelationshipDef** class.

To add a new relationship type (also referred to as a relationship definition) to a tool information model, use the **CreateRelationshipDef** method of the **ReposTypeLib** object.

A RelationshipDef object is also a RepositoryObject. In addition to the members described here, RelationshipDef objects also provide the members that are defined for repository objects.

When to Use

Use the **RelationshipDef** object to:

- Access any persistent properties that are attached to a relationship definition.
- Determine which collection types are associated with a relationship definition.
- Determine which repository type libraries contain a relationship definition.

Properties

Property	Description
ClassID	This property is reserved for future use.

Collections

Collection	Description
Interfaces	This collection is empty for relationship definitions. It is reserved for future use.
ItemInCollections	The collection of two collection types that are associated with this relationship definition.
Properties	The collection of all persistent properties that are attached to the RelationshipDef object.
ReposTypeLibScopes	The collection of all repository type libraries that contain this definition.

RelationshipDef ItemInCollections Collection

See Also

A relationship type is associated with two collection types. Origin collections conform to one collection type (the origin collection type), and destination collections conform to the other collection type (the destination collection type). The ItemInCollections collection contains the two collection definition objects that represent the origin and destination collection types.

Collection Descriptor	Descriptor Value
Relationship Type	Collection-Contains-Items
Source Is Origin	No
Minimum Collection Size	Zero
Maximum Collection Size	Two
Sequenced Collection	No
Deletes Propagated	No
Destinations Named	No
Case Sensitive Names	Not applicable
Unique Names	Not applicable

Syntax

Set *variable* = *object*.**ItemInCollections(** *index* **)**

The **ItemInCollections** collection syntax has these parts:

Part	Description
variable	A variable declared as a CollectionDef object. Receives the specified collection definition.
object	An object expression that evaluates to a RelationshipDef object.
index	An integer index that identifies which element in the collection is to be addressed. The valid range is from one to the number of elements in the collection. The number of elements in the collection is specified by *object*.ItemInCollections.Count.

Remarks

If the relationship type has not yet been connected to its origin and destination collection types, then this collection can contain less than two collection types.

RelationshipDef Properties Collection

See Also

A **Properties** collection contains all of the persistent properties and collections that are attached to an object via a particular interface. The RelationshipDef object exposes three separate Properties collections. These collections are exposed by:

1. The IReposTypeInfo interface (the default).
2. The IRepositoryObject interface.
3. The IAnnotationalProps interface.

Syntax

Set *variable* = *object*.**Properties(** *index* **)**

The **Properties** collection syntax has these parts:

Part	Description
variable	A variable declared as a ReposProperty object. Receives the specified property.
object	An object expression; evaluates to an object that exposes: • IReposTypeInfo, • IRepositoryObject, or • IAnnotationalProps as the default interface.
index	An integer index that identifies which property in the collection is to be addressed. The valid range is from one to the number of elements in the collection. The number of elements in the collection is specified by *object*.Properties.Count.

Remarks

For details on how to access a member of an interface that is not the default interface, see Accessing Automation Object Members.

RelationshipDef ReposTypeLibScopes Collection

See Also

The collection of repository type libraries that contain this definition.

Collection Descriptor	Descriptor Value
Relationship Type	ReposTypeLib-ScopeFor-ReposTypeInfo
Source Is Origin	No
Minimum Collection Size	One
Maximum Collection Size	Many
Sequenced Collection	No
Deletes Propagated	Yes
Destinations Named	Yes
Case Sensitive Names	No
Unique Names	Yes

Syntax

Set *variable* = *object*.**ReposTypeLibScopes(** *index* **)**

The **ReposTypeLibScopes** collection syntax has these parts:

Part	Description
variable	A variable declared as a ReposTypeLib object. Receives the specified repository type library object.
object	An object expression that evaluates to a RelationshipDef object.
index	An integer index that identifies which element in the collection is to be addressed. The valid range is from one to the number of elements in the collection. The number of elements in the collection is specified by *object*.TypeLibScopes.Count.

Repository Engine Classes

Repository engine classes are used to add, retrieve, and change tool information model data in the repository. To create a new tool information model, or extend an existing one, use the Type Information Model classes.

All repository classes expose the standard **IUnknown** and **IDispatch** interfaces that provide fundamental COM and Automation support.

Classes

ObjectCol

Relationship

RelationshipCol

Repository

RepositoryObject

ReposProperties

ReposProperty

ObjectCol Class

See Also Interfaces

An object collection is a set of repository objects that can be enumerated. Two kinds of object collections are supported by the repository:

1. The collection of destination objects that correspond to the relationships in a relationship collection. Use the **RelationshipCol** class to manage this kind of collection.
2. The collection of all objects in the repository that implement a particular interface. Use the **ObjectCol** class to enumerate objects in this kind of object collection.

Use the **IInterfaceDef::ObjectInstances** method to materialize an instance of this class.

When to Use

Use the **ObjectCol** class to access the collection of repository objects that expose a particular interface.

Interfaces

Interface	Description
IObjectCol	Manages objects in a collection.
IRepositoryDispatch	Provides enhanced dispatch support.

Relationship Class

See Also Interfaces

A relationship connects two repository objects in the repository database. A relationship has an origin repository object, a destination repository object, and a set of properties.

When to Use

Use the **Relationship** class to manipulate a relationship, or to retrieve the source, target, origin, or destination object for a relationship.

Interfaces

Interface	Description
IAnnotationalProps	Gets and sets annotational properties.
IRelationship	Retrieves information about a relationship.
IRepositoryDispatch	Provides enhanced dispatch support.
IRepositoryItem	Manages repository objects and relationships.

In each repository relationship, there is an *origin* and a *destination*. Consider this relationship example: *Fred drives a black truck*. "Person drives vehicle" is the relationship type, Fred is the origin of the relationship, and the black truck is the destination of the relationship.

Don't confuse the terms origin and destination with the terms source and target. For any particular relationship, the source and target are dependent upon whether you are navigating via the collection of origin objects or the collection of destination objects. The origin and destination for the *relationship* are not dependent upon which collection is being navigated.

When navigating into or out of a repository collection, the *source* object is the one object to which all objects in the collection are related. The *target* object can be any one of the objects in the collection.

Don't confuse the terms source and target with the terms origin and destination. For any particular relationship, the source and target are dependent upon whether you are navigating via the collection of origin objects or the collection of destination objects. The origin and destination for the *relationship* are not dependent which collection is being navigated.

RelationshipCol Class

See Also Interfaces

A relationship collection is the set of repository relationships that connect a particular source repository object to a set of one or more target objects. All of the relationships in the collection must conform to the same relationship type.

When to Use

Use the **RelationshipCol** class to manage a collection of relationships in the repository database.

Interfaces

Interface	Description
IRelationshipCol	Manages a collection of relationships.
IRepositoryDispatch	Provides enhanced dispatch support.
ITargetObjectCol	Manages objects in a target object collection.

Repository Class

See Also Interfaces

When you populate a tool information model, the objects and relationships that conform to the model are stored in a repository. Multiple tool information models may be stored in the same repository. The **Repository** class represents your connection to a particular repository.

When to Use

You can use the Repository class to connect to a repository, retrieve the root object of the repository, create new repository objects, and manage repository transactions and error handling.

Interfaces

Interface	Description
IRepository	Creates and populates a repository.
IRepositoryDispatch	Provides enhanced dispatch support.
IRepositoryErrorQueueHandler	Creates and assigns error queues.
IRepositoryODBC	Provides access to repository database connection information.
IRepositoryTransaction	Controls repository transactions.

RepositoryObject Class

See Also Interfaces

When you define a tool information model in the repostory, you define classes of objects, types of relationships that can exist between objects, and various properties that are attached to these object classes and relationship types. The object classes that you define in your tool information model derive their fundamental characteristics from the **RepositoryObject** class.

When to Use

Use the RepositoryObject class to access, modify, or delete objects in the repository.

Interfaces

Interface	Description
IRepositoryDispatch	Provides enhanced dispatch support.
IRepositoryItem	Manages repository objects and relationships.
IRepositoryObject	Retrieves repository object identifiers.
IRepositoryObjectStorage	Creates and loads repository objects.

ReposProperties Class

See Also Interfaces

The **ReposProperties** class provides access to the Properties collection. The Properties collection gives you a convenient mechanism to enumerate through all of the persistent properties and collections of an interface. The **ReposProperty** class can be used to access the individual members in the Properties collection.

When to Use

Use the ReposProperties class to access the properties and collections of a repository object, when no custom implementation is available, and you do not already know what members are exposed by the object's interface.

Interfaces

Interface	Description
IReposProperties	Provides access to the members that are attached to an interface.
IRepositoryDispatch	Provides enhanced dispatch support.

ReposProperty Class

See Also Interfaces

The **ReposProperty** class provides access to a persistent member (a property or collection) of a tool information model interface.

When to Use

Use the ReposProperty class to access a persistent interface member, when no custom implementation is available, and you do not already know the type or name of the member.

Interfaces

Interface	Description
IReposProperty	Provides access to the members that are attached to an interface.
IRepositoryDispatch	Provides enhanced dispatch support.

Type Information Model Classes

The Type Information Model is the object model the repository uses to store tool information models. Use the Type Information Model classes to create or extend a tool information model. These classes build upon the fundamental Repository Engine Classes.

All repository classes expose the standard **IUnknown** and **IDispatch** interfaces that provide fundamental COM and Automation support.

Classes

ClassDef

CollectionDef

InterfaceDef

MethodDef

PropertyDef

RelationshipDef

ReposRoot

ReposTypeLib

ClassDef Class

See Also Interfaces

When you define a tool information model in the repository, you define classes of objects, types of relationships that can exist between objects, and various properties that are attached to these object classes and relationship types. The object classes that you define in your tool information model are represented by instances of the **ClassDef** class.

To insert a new class definition into a tool information model, use the **ReposTypeLib** class.

When to Use

Use the ClassDef class to complete the definition of a new repository class. You can define new interfaces and attach them to the class definition. You can also attach existing interfaces to the class definition.

Interfaces

Interface	Description
IAnnotationalProps	Gets and sets annotational properties.
IClassDef	Manages class definitions.
IRepositoryDispatch	Provides enhanced dispatch support.
IRepositoryItem	Manages repository objects and relationships.
IRepositoryObject	Retrieves repository object identifiers.
IRepositoryObjectStorage	Creates and loads repository objects.
IReposTypeInfo	Contains the collection of definition objects that are associated with a tool information model's repository type library.

CollectionDef Class

See Also Interfaces

Repository objects are related to each other via relationships. The set of relationships, all of the same type, that relate one object to zero or more other objects, is a relationship collection.

A collection type (also referred to as a collection definition) defines how instances of a particular collection type will behave. The characteristics of the collection type determine:

- The minimum and maximum number of items in a collection.
- Whether or not the collection type is an origin collection type.
- Whether or not the collection type permits the naming of destination objects, and if so, whether those names are case sensitive, and required to be unique.
- Whether or not the collection type permits the explicit sequencing of items in the collection.
- What happens to related objects when objects or relationships in the collection are deleted.
- The kind of relationship that a particular collection type uses to relate objects to each other.

A collection is attached to an interface as a *member* of the interface. To add a new collection type to an interface definition, use the **InterfaceDef** class.

When to Use

Use the **CollectionDef** class to retrieve or modify the properties of a collection type, or to determine the kind of relationship that the collection implements.

Interfaces

Interface	Description
IAnnotationalProps	Gets and sets annotational properties.
ICollectionDef	Manages collection definitions.
IInterfaceMember	Relates a member to an interface.
IRepositoryDispatch	Provides enhanced dispatch support.
IRepositoryItem	Manages repository objects and relationships.
IRepositoryObject	Retrieves repository object identifiers.
IRepositoryObjectStorage	Creates and loads repository objects.

InterfaceDef Class

See Also Interfaces

The properties methods, and collections that a class implements are organized into functionally related groups. Each group is implemented as a COM interface. The properties, methods, and collections of each interface are *members* of the interface. An interface definition is the template to which an interface conforms. Interface definitions are instances of the **InterfaceDef** class.

To create a new interface definition, use the **ClassDef** class or the **ReposTypeLib** class.

When to Use

Use the InterfaceDef class to:

- Retrieve or modify properties of an interface definition.
- Determine which members are attached to an interface definition.
- Determine which classes implement an interface.
- Determine the base interface from which an interface derives.
- Determine what interfaces derive from a particular interface.
- Determine what repository objects expose a particular interface.
- Add a new property, method or collection type to an interface definition.

Interfaces

Interface	Description
IAnnotationalProps	Gets and sets annotational properties.
IInterfaceDef	Manages interface definitions.
IRepositoryDispatch	Provides enhanced dispatch support.
IRepositoryItem	Manages repository objects and relationships.
IRepositoryObject	Retrieves repository object identifiers.
IRepositoryObjectStorage	Creates and loads repository objects.
IReposTypeInfo	Contains the collection of definition objects that are associated with a tool information model's repository type library.

MethodDef Class

See Also Interfaces

When you define a class for a tool information model, you specify the interfaces that the class implements. For each of those interfaces, you specify the *members* (properties, methods, and collections) that are attached to the interface.

The definition of a method as a member of an interface does not result in the method's implementation logic being stored in the repository. However, it does add the method name to the set of defined member names for that interface. It also reserves the method's dispatch identifier in the set of defined dispatch identifier values for the interface.

Instances of the **MethodDef** class represent method definitions.

To attach a new method to an interface, use the **InterfaceDef** class.

When to Use

Use the MethodDef class to access or modify the characteristics of a method definition, or to determine the interface definition to which a particular method is attached.

Interfaces

Interface	Description
IAnnotationalProps	Gets and sets annotational properties.
IInterfaceMember	Relates a member to an interface.
IRepositoryDispatch	Provides enhanced dispatch support.
IRepositoryItem	Manages repository objects and relationships.
IRepositoryObject	Retrieves repository object identifiers.
IRepositoryObjectStorage	Creates and loads repository objects.

PropertyDef Class

See Also Interfaces

When you define a class for a tool information model, you specify the interfaces that the class implements. For each of those interfaces, you specify the *members* (properties, methods, and collections) that are attached to the interface.

In order to attach a property to an interface, a property definition must exist for the property. The characteristics of the property (it's name, dispatch identifier, data type, and various storage details) are stored in the property definition. Property definitions are instances of the **PropertyDef** class.

To attach a new property to an interface, use the **InterfaceDef** class.

When to Use

Use the PropertyDef class to access or modify the characteristics of a property definition, or to determine the interface definition to which a particular property is attached.

Interfaces

Interface	Description
IAnnotationalProps	Gets and sets annotational properties.
IInterfaceMember	Relates a member to an interface.
IPropertyDef	Retains property characteristics.
IRepositoryDispatch	Provides enhanced dispatch support.
IRepositoryItem	Manages repository objects and relationships.
IRepositoryObject	Retrieves repository object identifiers.
IRepositoryObjectStorage	Creates and loads repository objects.

RelationshipDef Class

See Also Interfaces

When you define a tool information model in the repository, you define classes of objects, types of relationships that can exist between objects, and various properties that are attached to these object classes and relationship types. The relationship types that you define in your tool information model are represented by instances of the **RelationshipDef** class.

When to Use

Use the **RelationshipDef** Class to access the properties of a relationship definition (also referred to as a relationship type).

To insert a new relationship type into a tool information model, use the **ReposTypeLib** class.

Interfaces

Interface	Description
IAnnotationalProps	Gets and sets annotational properties.
IRepositoryDispatch	Provides enhanced dispatch support.
IRepositoryItem	Manages repository objects and relationships.
IRepositoryObject	Retrieves repository object identifiers.
IRepositoryObjectStorage	Creates and loads repository objects.
IReposTypeInfo	Contains the collection of definition objects that are associated with a tool information model's repository type library.

ReposRoot Class

See Also Interfaces

There is one root object in each repository. The root object is the starting point for navigating to other objects in the repository. The root object serves as the starting point for both *type* and *instance* data navigations.

- **Type data navigation:**

  When you create a tool information model, the corresponding repository type library is attached to the root object via the ReposTypeLibs collection. This collection can be used to enumerate all of the tool information models (type data) that are contained in the repository.
- **Instance data navigation:**

  Once a tool information model is defined, the repository can be populated with *instance data.* This instance data consists of objects and relationships that conform to the classes and relationship types of the tool information model.

  Because the objects are connected via relationships, you can navigate through this data. However, to enable general purpose repository browsers to navigate this data, the first navigational step must be from the root object of the repository through a root relationship collection to the *primary objects* of your tool information model. Primary objects are objects that make a good starting point for navigating to other objects of your tool information model.

  Because this root relationship collection is different for each tool information model, it must be defined *by the tool information model*. There are two options for attaching this relationship collection to the root object:

  1. The ReposRoot class implements the IReposRoot interface. This interface is provided to tool information model creators as a connection point. You can add your connecting relationship collection to this interface.
  2. You can extend the ReposRoot class to implement a new interface that is defined in your tool information model. This interface implements a relationship collection that attaches the root object to the primary objects in your tool information model.

To facilitate navigation, the root object in all repositories always has *the same object identifier.* The symbolic name for this object identifier is **OBJID_ReposRootObj**.

When to Use

Use the **ReposRoot** class to:

- Obtain a starting point for navigating to objects in the repository.
- Create a new tool information model.
- Attach a relationship collection to the root object of the repository that connects to the primary objects of your tool information model.

Interfaces

Interface	Description
IAnnotationalProps	Gets and sets annotational properties.
IManageReposTypeLib	Adds tool information models (repository type libraries) to a repository.
IRepositoryDispatch	Provides enhanced dispatch support.
IRepositoryItem	Manages repository objects and relationships.
IRepositoryObject	Retrieves repository object identifiers.
IRepositoryObjectStora	Creates and loads repository objects.

ge	
IReposRoot	Provides an attachment point for tool information model instance data.

ReposTypeLib Class

See Also Interfaces

There is one repository type library for every tool information model contained in the repository database. Each tool information model provides a logical grouping of all of the type definitions related to a particular tool (or tool set). Repository type libraries are instances of the **ReposTypeLib** class.

To insert a new tool information model into the repository database, use the **ReposRoot** class.

When to Use

Use the ReposTypeLib class to:

- Define new classes, relationship types, and interfaces for a tool information model.
- Retrieve or modify the global identifier associated with a repository type library.
- Determine which type definitions are associated with a particular repository type library.

Interfaces

Interface	Description
IAnnotationalProps	Gets and sets annotational properties.
IRepositoryDispatch	Provides enhanced dispatch support.
IRepositoryItem	Manages repository objects and relationships.
IRepositoryObject	Retrieves repository object identifiers.
IRepositoryObjectStorage	Creates and loads repository objects.
IReposTypeLib	Creates class, interface, and relationship definitions for a repository type library.

IAnnotationalProps Interface

Annotational properties are repository properties that can be associated with *individual* repository objects *or relationships*. When a normal property is defined as a member of a repository interface, it is defined for *all* objects that implement that interface. Normal properties cannot be associated with repository relationships.

In order to be able to attach an annotational property value to a particular repository object, two requirements must be met:

1. The object must conform to an object class that exposes the IAnnotationalProps interface.
2. A property definition object must exist for an IAnnotationalProps interface property with a name that matches the name of your annotational property.

If these two requirements are met, then you can attach an annotational property value to an object by using the **IReposProperty::put_Value** method to set the value of the annotational property for that particular object.

When to Use

Use the IAnnotationalProps interface to access the annotational properties of a repository object or relationship.

Methods

IUnknown Method	Description
QueryInterface	Returns pointers to supported interfaces.
AddRef	Increments the reference count.
Release	Decrements the reference count.

IDispatch Method	Description
GetIDsOfNames	Maps a single member and a set of argument names to a corresponding set of dispatch identifiers.
GetTypeInfo	Retrieves a type information object, which can be used to get the type information for an interface.
GetTypeInfoCount	Retrieves the number of type information interfaces that an object provides (either 0 or 1).
Invoke	Provides access to properties and methods exposed by an Automation object.

IRepositoryDispatch Method	Description
get_Properties	Retrieves the IReposProperties interface pointer. The IReposProperties interface provides access to the Properties collection.

Remarks

Annotational properties are maintained by the repository as string data. The creator and users of the annotational property must get and set the property value using the appropriate data type via the **VARIANT** structure. If a data type other than *string* is used, the repository will perform the appropriate data conversion.

Since *all* annotational properties in the repository must be defined as interface members of the IAnnotationalProps interface, all annotational property names *share the same name space*. When you choose a name for an annotational property, make the name as specific and unique as possible.

INamedObject Interface

See Also Properties Methods Collections

Normally, a name is associated with a repository object via a naming relationship. The collection for such a relationship provides the scope for the name, and can require that all names in the collection be unique. This is the preferred method for naming objects, when a given object will be the destination of only one naming relationship.

If your tool information model contains a class that is not the destination of a naming relationship type, or is the destination of multiple relationship types, but no single relationship type is the obvious choice to be the *naming* relationship type, then you can attach the name property to the class. This is accomplished by defining your class to implement the **INamedObject** interface. If your class implements the INamedObject interface, the repository engine will use that interface when asked to retrieve or set an object name.

When to Use

Use the INamedObject interface to access the **Name** property of a repository object that exposes this interface.

Properties

Property	Description
Name	The name of the object.

Methods

IUnknown Method	Description
QueryInterface	Returns pointers to supported interfaces.
AddRef	Increments the reference count.
Release	Decrements the reference count.

IDispatch Method	Description
GetIDsOfNames	Maps a single member and a set of argument names to a corresponding set of dispatch identifiers.
GetTypeInfo	Retrieves a type information object, which can be used to get the type information for an interface.
GetTypeInfoCount	Retrieves the number of type information interfaces that an object provides (either 0 or 1).
Invoke	Provides access to properties and methods exposed by an Automation object.

IRepositoryDispatch Method	Description
get_Properties	Retrieves the IReposProperties interface pointer. The IReposProperties interface provides access to the persistent members exposed by the INamedObject interface.

Remarks

None of the standard repository engine or Type Information Model classes implement the INamedObject interface. However, the repository engine does use the INamedObject interface, if it is exposed by a repository object.

When the **IRepositoryItem::get_Name** method is invoked for a repository object, the repository engine will perform these steps to retrieve the name:

1. If the object exposes the INamedObject interface, the repository engine returns the value of the Name property.
2. Otherwise, the repository engine searches for a naming relationship for which the current object is the destination object.
3. If such a relationship is found, the repository engine returns the name associated with that relationship.
4. If the object is not the destination of a naming relationship, then the repository engine returns a null name.

When the **IRepositoryItem::put_Name** method is invoked for a repository object, the repository engine will perform these steps to set the name:

1. The repository engine sets the value of the Name property of all naming relationships for which the object is the destination.
2. If the object exposes the INamedObject interface, the repository engine also sets the value of the Name property attached to that interface.

INamedObject Name Property

See Also

This property contains the name of an object that exposes the INamedObject interface. The name can be up to 200 bytes in length.

Dispatch Identifier: DISPID_ObjName (68)

Property Data Type: String

ISummaryInformation Interface

See Also Properties Methods Collections

The ISummaryInformation interface maintains **Comments** and **ShortDescription** properties for objects that expose this interface.

When to Use

Use the ISummaryInformation interface to access the **Comments** and **ShortDescription** properties of a repository object.

Properties

Property	Description
Comments	General comments about the object.
ShortDescription	Brief description of the object.

Methods

IUnknown Method	Description
QueryInterface	Returns pointers to supported interfaces.
AddRef	Increments the reference count.
Release	Decrements the reference count.

IDispatch Method	Description
GetIDsOfNames	Maps a single member and a set of argument names to a corresponding set of dispatch identifiers.
GetTypeInfo	Retrieves a type information object, which can be used to get the type information for an interface.
GetTypeInfoCount	Retrieves the number of type information interfaces that an object provides (either 0 or 1).
Invoke	Provides access to properties and methods exposed by an Automation object.

IRepositoryDispatch Method	Description
get_Properties	Retrieves the IReposProperties interface pointer. The IReposProperties interface provides access to the properties exposed by the ISummaryInformation interface.

ISummaryInformation Comments Property

See Also

This property contains general comments about an object. Up to 65,536 bytes of information may be stored in this property.

Dispatch Identifier: DISPID_Comments (66)

Property Data Type: Long varchar

ISummaryInformation ShortDescription Property

See Also

This property contains a short description of an object. Up to 255 bytes of information may be stored in this property.

Dispatch Identifier: DISPID_ShortDesc (67)

Property Data Type: varchar

IEnumRepositoryErrors Interface

This interface provides enumeration capabilities for the set of errors that have been placed on the repository error queue.

When to Use

Use the IEnumRepositoryErrors interface to access the queue of repository errors.

Methods

IUnknown Method	Description
QueryInterface	Returns pointers to supported interfaces.
AddRef	Increments the reference count.
Release	Decrements the reference count.

IEnumRepositoryErrors Method	Description
Clone	Clones the current enumerator.
Next	Returns the next one or more elements.
Reset	Resets the enumerator to the beginning.
Skip	Skips over the next one or more elements.

IEnumRepositoryErrors::Clone

See Also

Use this method to create a clone of the COM enumerator object. After cloning, the two enumerators operate independently of each other.

HRESULT Clone(IEnumRepositoryErrors ***ppIEnum* **);**

Parameters

**ppIEnum* [out]
The interface pointer for the new enumerator object.

Return Value

S_OK
The method completed successfully.

Error Values
This method failed to complete successfully.

IEnumRepositoryErrors::Next

See Also

Use this method to retrieve the next one or more elements from the enumeration. There are two variations of this method.

```
HRESULT Next(
        ULONG           iCount,
        REPOSERR    *psErrors,
        ULONG           *piFetched
    );

HRESULT Next( IErrorInfo     **ppIErrorInfo );
```

Parameters

iCount [in]
The number of elements the caller is requesting.

**psErrors* [out]
The array of repository error information structures for the retrieved items.

**ppIErrorInfo* [out]
The interface pointer to the error information object for the first element in the error queue.

**piFetched* [out]
The number of elements actually fetched for the caller.

Return Value

S_OK
The method completed successfully.

Error Values
This method failed to complete successfully.

IEnumRepositoryErrors::Reset

See Also

Use this method to reset the enumerator to the beginning of the enumeration sequence.

HRESULT Reset(*void*);

Return Value

S_OK
  The method completed successfully.

Error Values
  This method failed to complete successfully.

IEnumRepositoryErrors::Skip

See Also

Use this method to skip over the next one or more elements in the enumeration.

HRESULT Skip(ULONG *iCount* **);**

Parameters

iCount [in]
The number of elements to be skipped.

Return Value

S_OK
The method completed successfully.

Error Values
This method failed to complete successfully.

IObjectCol Interface

An object collection is a set of repository objects that can be enumerated. Two kinds of object collections are supported by the repository:

1. The collection of destination objects that correspond to the relationships in a relationship collection. Use the **ITargetObjectCol** interface to manage this kind of collection.
2. The collection of all objects in the repository that conform to a particular class or expose a particular interface.

When to Use

Use the **IObjectCol** interface to enumerate the collection of repository objects that conform to a particular class or expose a particular interface. With this interface, you can:

- Get a count of the number of objects in the collection.
- Enumerate the objects in the collection.
- Retrieve an **IRepositoryObject** pointer to one of the objects in the collection.
- Refresh the cached image of the object collection.

Methods

IUnknown Method	Description
QueryInterface	Returns pointers to supported interfaces.
AddRef	Increments the reference count.
Release	Decrements the reference count.

IDispatch Method	Description
GetIDsOfNames	Maps a single member and a set of argument names to a corresponding set of dispatch identifiers.
GetTypeInfo	Retrieves a type information object, which can be used to get the type information for an interface.
GetTypeInfoCount	Retrieves the number of type information interfaces that an object provides (either 0 or 1).
Invoke	Provides access to properties and methods exposed by an Automation object.

IObjectCol Method	Description
get_Count	Retrieves a count of the number of objects in the collection.
get_Item	Retrieves an IRepositoryObject interface pointer for the specified collection object.
_NewEnum	Retrieves an enumeration interface pointer for the collection.
Refresh	Refreshes the cached image of the

object collection.

IObjectCol::get_Count

See Also

This method is used to retrieve a count of the number of repository objects that are in the object collection.

HRESULT get_Count(long *piCount);

Parameters

**piCount* [out]
The number of objects in the collection.

Return Value

S_OK
The method completed successfully.

Error Values
This method failed to complete successfully.

IObjectCol::_NewEnum

See Also

This method retrieves an enumeration interface pointer for the object collection. This interface is a standard Automation enumeration interface. It supports the Clone, Next, Reset, and Skip methods. You can use the enumeration interface to step through the objects in the collection.

HRESULT _NewEnum(
IUnknown ***ppIEnumObjects*
);

Parameters

**ppIEnumObjects* [out]
The enumeration interface pointer.

Return Value

S_OK
The method completed successfully.

Error Values
This method failed to complete successfully.

IObjectCol::get_Item

See Also

This method retrieves the specified object from the collection.

```
HRESULT get_Item(
    VARIANT              sItem,
    IRepositoryObject    **ppIReposObj
);
```

Parameters

sItem [in]
Identifies the item to be retrieved from the collection. This parameter can be either the index, the name, or the object identifier of the item.

**ppIReposObj* [out]
The IRepositoryObject interface pointer for the specified object from the collection.

Return Value

S_OK
The method completed successfully.

Error Values
This method failed to complete successfully.

Remarks

An object can only be retrieved by name if it is the destination object of a naming relationship.

IObjectCol::Refresh

See Also

This method refreshes the cached image of the collection. All unchanged data for objects in the collection is flushed from the cache.

HRESULT Refresh(long *iMilliseconds* **);**

Parameters

iMilliseconds [in]
All unchanged data relating to the collection that has been in the cache for longer than *iMilliseconds* milliseconds is refreshed. Set to zero to refresh all unchanged data.

Return Value

S_OK
The method completed successfully.

Error Values
This method failed to complete successfully.

ITargetObjectCol Interface

A target object collection is the set of repository objects that are attached to a particular source repository object via a relationship collection.

When to Use

Use the ITargetObjectCol Interface to manage the repository objects that belong to a particular relationship collection. With this interface, you can:

- Get a count of the number of objects in the collection.
- Enumerate the objects in the collection.
- Add and remove objects to and from the collection.
- If the collection is sequenced, place an object in a specific spot in the collection sequence.
- Retrieve an IRepositoryObject pointer to one of the objects in the collection.
- Obtain the type of the collection.
- Retrieve an interface pointer for the collection's source object.
- Refresh the cached image of the target object collection.

Methods

IUnknown Method	Description
QueryInterface	Returns pointers to supported interfaces.
AddRef	Increments the reference count.
Release	Decrements the reference count.

IDispatch Method	Description
GetIDsOfNames	Maps a single member and a set of argument names to a corresponding set of dispatch identifiers.
GetTypeInfo	Retrieves a type information object, which can be used to get the type information for an interface.
GetTypeInfoCount	Retrieves the number of type information interfaces that an object provides (either 0 or 1).
Invoke	Provides access to properties and methods exposed by an Automation object.

IObjectCol Method	Description
Count	Retrieves a count of the number of objects in the collection.
_NewEnum	Retrieves an enumeration interface pointer for the collection. This interface is a standard Automation enumeration interface. It supports the Clone, Next, Reset, and Skip methods.
Item	Retrieves an IRepositoryObject interface

	pointer for the specified collection object.
Refresh	Refreshes the cached image of the target object collection.

ITargetObjectCol Method	Description
Add	Adds an object to the collection.
get_Source	Retrieves an interface pointer for the collection's source object.
get_Type	Retrieves the object identifier for the collection's definition object.
Insert	Inserts an object into a specific spot in a sequenced collection.
Move	Moves an object from one spot to another in a sequenced collection.
Remove	Removes an object from the collection.

Remarks

The ITargetObjectCol Interface is very similar to the IRelationshipCol Interface. Use the ITargetObjectCol Interface when you are primarily interested in working with objects. Use the IRelationshipCol Interface when you are primarily interested in working with relationships between objects.

ITargetObjectCol::Add

See Also

This method is used to add a new item to a repository object collection, when the sequencing of objects in the collection is not important. An interface pointer for the new relationship is passed back to the caller.

```
HRESULT Add(
    IDispatch       *plReposObj,
    BSTR             Name,
    IRelationship   **pplRelship
);
```

Parameters

**plReposObj* [in]
The repository object to be added to the collection.

Name [in]
The name to be assigned to the object that is being added to the collection.

**pplRelship* [out]
The newly added object's relationship interface pointer.

Return Value

S_OK
The method completed successfully.

Error Values
This method failed to complete successfully.

Remarks

Objects may only be added to a collection when the collection's source object is also the collection's origin object.

ITargetObjectCol::get_Source

See Also

This method retrieves the IRepositoryObject interface pointer for the collection's source object.

HRESULT get_Source(IRepositoryObject ***ppIInterface* **);**

Parameters

**ppIInterface* [out]
The interface pointer of the IRepositoryObject interface for the source object.

Return Value

S_OK
The method completed successfully.

Error Values
This method failed to complete successfully.

ITargetObjectCol::get_Type

See Also

This method retrieves the type of the collection; that is, it returns the object identifier for the collection's definition object.

HRESULT get_Type(VARIANT **pColDefObjId* **);**

Parameters

**pColDefObjId* [out]
The object identifier of the collection's definition object.

Return Value

S_OK
The method completed successfully.

Error Values
This method failed to complete successfully.

ITargetObjectCol::Insert

See Also

This method adds an object to the collection at a specified point in the collection sequence. An interface pointer for the new relationship is passed back to the caller.

```
HRESULT Insert(
    IDispatch       *pIReposObj,
    long             iIndex,
    BSTR             Name,
    IRelationship   **ppIRelship
);
```

Parameters

**pIReposObj* [in]
The repository object to be inserted into the collection sequence.

iIndex [in]
The index of the sequence location where the object is to be inserted. If another object is already present at this sequence location, the new object is inserted before the existing object.

Name [in]
The name of the object. Set this parameter to a null string if the object is not referred to by name.

**ppIRelship* [out]
The IRelationship interface pointer for the new object's relationship with the collection's origin object.

Return Value

S_OK
The method completed successfully.

Error Values
This method failed to complete successfully.

Remarks

Objects may only be inserted into a collection when the collection's source object is also the collection's origin object.

This method can only be used for collections that are sequenced.

ITargetObjectCol::Move

See Also

This method moves an object from one point in the collection sequence to another point.

HRESULT Move(long *iIndexFrom,* **long** *iIndexTo* **);**

Parameters

iIndexFrom [in]
The index of the object to be moved in the collection sequence.

iIndexTo [in]
The index of the sequence location to which the object is to be moved.

Return Value

S_OK
The method completed successfully.

Error Values
This method failed to complete successfully.

Remarks

This method can only be used for collections that are sequenced.

ITargetObjectCol::Remove

See Also

This method removes the specified object from the collection. The relationship instance for the object is deleted.

HRESULT Remove(VARIANT *sItem* **);**

Parameters

sItem [in]
Identifies the item to be retrieved from the collection. This parameter can be either the index, the name, or the object identifier of the item.

Return Value

S_OK
The method completed successfully.

Error Values
This method failed to complete successfully.

Remarks

An object can only be removed by name if it is the destination object of a naming relationship.

IRelationship Interface

See Also Properties Methods Collections

A relationship connects two repository objects in the repository database. A relationship has an origin repository object, a destination repository object, and a set of properties. Each relationship conforms to a particular relationship type.

When to Use

Use the IRelationship Interface to manipulate a relationship, or to retrieve the source, target, origin, or destination object for a relationship.

Methods

IUnknown Method	Description
QueryInterface	Returns pointers to supported interfaces.
AddRef	Increments the reference count.
Release	Decrements the reference count.

IDispatch Method	Description
GetIDsOfNames	Maps a single member and a set of argument names to a corresponding set of dispatch identifiers.
GetTypeInfo	Retrieves a type information object, which can be used to get the type information for an interface.
GetTypeInfoCount	Retrieves the number of type information interfaces that an object provides (either 0 or 1).
Invoke	Provides access to properties and methods exposed by an Automation object.

IRepositoryDispatch Method	Description
get_Properties	Retrieves the IReposProperties interface pointer. The IReposProperties interface provides access to the Properties collection.

IRepositoryItem Method	Description
Delete	Deletes a repository item.
get_Interface	Retrieves an interface pointer to the specified item interface.
get_Name	Retrieves the name associated with an item.
get_Repository	Retrieves the IRepository interface pointer for an item's open repository instance.
get_Type	Retrieves the type of an item.
Lock	Locks the item.
put_Name	Sets the name associated with an item.

Refresh	Refreshes the cached image of the item.

IRelationship Method	Description
get_Destination	Retrieves an interface pointer to the destination object.
get_Origin	Retrieves an interface pointer to the origin object.
get_Source	Retrieves an interface pointer to the source object.
get_Target	Retrieves an interface pointer to the target object.

IRelationship::get_Destination

See Also

Retrieves an IRepositoryObject interface pointer to the destination object of a relationship.

HRESULT get_Destination(
IRepositoryObject ***ppIReposObj*
);

Parameters

**ppIRepObj* [out]
The IRepositoryObject interface pointer for the destination repository object.

Return Value

S_OK
The method completed successfully.

Error Values
This method failed to complete successfully.

IRelationship::get_Origin

See Also

Retrieves an IRepositoryObject interface pointer to the origin object of a relationship.

HRESULT get_Origin(
IRepositoryObject ***ppIRepObj*
);

Parameters

**ppIRepObj* [out]
The IRepositoryObject interface pointer for the origin repository object.

Return Value

S_OK
The method completed successfully.

Error Values
This method failed to complete successfully.

IRelationship::get_Source

See Also

Retrieves an IRepositoryObject interface pointer to the source object of a relationship.

HRESULT get_Source(
IRepositoryObject ***ppIRepObj*
);

Parameters

**ppIRepObj* [out]
The IRepositoryObject interface pointer for the source repository object.

Return Value

S_OK
The method completed successfully.

Error Values
This method failed to complete successfully.

IRelationship::get_Target

See Also

Retrieves an IRepositoryObject interface pointer to the target object of a relationship.

HRESULT get_Target(
IRepositoryObject ***ppIRepObj*
);

Parameters

**ppIRepObj* [out]
The IRepositoryObject interface pointer for the target repository object.

Return Value

S_OK
The method completed successfully.

Error Values
This method failed to complete successfully.

IRelationshipCol Interface

See Also Properties Methods Collections

A relationship collection is the set of repository relationships that connect a particular source repository object to a set of one or more target objects. All of the relationships in the collection must conform to the same relationship type.

When to Use

Use the IRelationshipCol interface to manage the repository relationships that belong to a particular relationship collection. With this interface, you can:

- Get a count of the number of relationships in the collection.
- Enumerate the relationships in the collection.
- Add and remove relationships to and from the collection.
- If the collection is sequenced, place a relationship in a specific spot in the collection sequence.
- Retrieve an IRelationship pointer to one of the relationships in the collection.
- Obtain the identifier of the collection's definition object.
- Retrieve an interface pointer for the collection's source object.

Methods

IUnknown Method	Description
QueryInterface	Returns pointers to supported interfaces.
AddRef	Increments the reference count.
Release	Decrements the reference count.

IDispatch Method	Description
GetIDsOfNames	Maps a single member and a set of argument names to a corresponding set of dispatch identifiers.
GetTypeInfo	Retrieves a type information object, which can be used to get the type information for an interface.
GetTypeInfoCount	Retrieves the number of type information interfaces that an object provides (either 0 or 1).
Invoke	Provides access to properties and methods exposed by an Automation object.

IRelationshipCol Method	Description
Add	Adds a relationship to the collection.
get_Count	Retrieves a count of the number of relationships in the collection.
_NewEnum	Retrieves an enumeration interface pointer for the collection.
get_Source	Retrieves an interface pointer for the collection's source object.
get_Type	Retrieves the object identifier for the

	collection's definition object.
Insert	Inserts a relationship into a specific spot in a sequenced collection.
get_Item	Retrieves an IRelationship interface pointer for the specified relationship.
Move	Moves a relationship from one spot to another in a sequenced collection.
Refresh	Refreshes the cached image of the relationship collection.
Remove	Removes a relationship from the collection.

Remarks

The IRelationshipCol interface is very similar to the ITargetObjectCol interface. Use the IRelationshipCol interface when you are primarily interested in working with relationships. Use the ITargetObjectCol interface when you are primarily interested in working with objects.

IRelationshipCol::Add

See Also

This method is used to add a new item to a repository relationship collection, when the sequencing of relationships in the collection is not important. An interface pointer for the new relationship is passed back to the caller.

```
HRESULT Add(
    IDispatch       *plReposObj,
    BSTR            Name,
    IRelationship   **pplRelship
);
```

Parameters

**plReposObj* [in]
The object for which a relationship is to be added to the relationship collection.

Name [in]
The name to be attached to the object via the new relationship.

**pplRelship* [out]
The newly added relationship's IRelationship interface pointer.

Return Value

S_OK
The method completed successfully.

Error Values
This method failed to complete successfully.

Remarks

Relationships may only be added to a collection when the collection's source object is also the collection's origin object.

IRelationshipCol::get_Count

See Also

This method is used to retrieve a count of the number of relationships that are in the relationship collection.

HRESULT get_Count(
long **piCount*
);

Parameters

**piCount* [out]
The number of relationships in the collection.

Return Value

S_OK
The method completed successfully.

Error Values
This method failed to complete successfully.

IRelationshipCol::_NewEnum

See Also

This method retrieves an enumeration interface pointer for the relationship collection. This interface is a standard Automation enumeration interface. It supports the Clone, Next, Reset, and Skip methods. You can use the enumeration interface to step through the relationships in the collection.

```
HRESULT _NewEnum(
        IUnknown      **ppIEnumRelships
    );
```

Parameters

**ppIEnumRelships* [out]
The enumeration interface pointer.

Return Value

S_OK
The method completed successfully.

Error Values
This method failed to complete successfully.

IRelationshipCol::get_Source

See Also

This method retrieves an interface pointer for the collection's source object.

HRESULT get_Source(
IRepositoryObject ***ppIInterface***
);

Parameters

**ppIInterface* [out]
The interface pointer of the desired interface for the source object.

Return Value

S_OK
The method completed successfully.

Error Values
This method failed to complete successfully.

IRelationshipCol::get_Type

See Also

This method retrieves the type of the collection; that is, it returns the object identifier for the collection's definition object.

HRESULT get_Type(
    VARIANT **pColDefObjId*
);

Parameters

**pColDefObjId* [out]
The object identifier of the collection's definition object.

Return Value

S_OK
The method completed successfully.

Error Values
This method failed to complete successfully.

IRelationshipCol::Insert

See Also

This method adds a relationship to the collection at a specified point in the collection sequence. An interface pointer for the new relationship is passed back to the caller.

```
HRESULT Insert(
    IDispatch       *pIReposObj,
    long             iIndex,
    BSTR             Name,
    IRelationship   **ppIRelship
);
```

Parameters

**pIReposObj* [in]
The repository object to be inserted into the collection sequence via the new relationship.

iIndex [in]
The index of the sequence location where the relationship is to be inserted. If another relationship is already present at this sequence location, the new relationship is inserted before the existing relationship.

Name [in]
The name to be associated with the object that is connected by the new relationship.

**ppIRelship* [out]
The IRelationship interface pointer for the new relationship.

Return Value

S_OK
The method completed successfully.

Error Values
This method failed to complete successfully.

Remarks

Relationships may only be inserted into a collection when the collection's source object is also the collection's origin object.

This method can only be used for collections that are sequenced.

IRelationshipCol::get_Item

See Also

This method retrieves the specified relationship from the collection.

```
HRESULT get_Item(
    VARIANT          sItem,
    IRelationship    **ppIRelship
);
```

Parameters

sItem [in]
Identifies the item to be retrieved from the collection. This parameter can be either the index, the name, or the object identifier of the item.

**ppIRelship* [out]
The IRelationship interface pointer for the specified relationship from the collection.

Return Value

S_OK
The method completed successfully.

Error Values
This method failed to complete successfully.

Remarks

The variation of this method that specifies the relationship by destination object name can only be used for collections that require unique names.

IRelationshipCol::Move

See Also

This method moves a relationship from one point in the collection sequence to another point.

```
HRESULT Move(
        long      iIndexFrom,
        long      iIndexTo
    );
```

Parameters

iIndexFrom [in]
The index of the relationship to be moved in the collection sequence.

iIndexTo [in]
The index of the sequence location to which the relationship is to be moved.

Return Value

S_OK
The method completed successfully.

Error Values
This method failed to complete successfully.

Remarks

This method can only be used for collections that are sequenced.

IRelationshipCol::Refresh

See Also

This method refreshes the cached image of the collection. All unchanged data for relationships in the collection is flushed from the cache.

HRESULT Refresh(long *iMilliseconds* **);**

Parameters

iMilliseconds [in]
All unchanged data relating to the collection that has been in the cache for longer than *iMilliseconds* milliseconds is refreshed. Set to zero to refresh all unchanged data.

Return Value

S_OK
The method completed successfully.

Error Values
This method failed to complete successfully.

IRelationshipCol::Remove

See Also

This method removes the specified relationship from the collection. The destination object of the relationship is not deleted.

HRESULT Remove(VARIANT *sItem* **);**

Parameters

sItem [in]
Identifies the item to be retrieved from the collection. This parameter can be either the index or the name associated with the item.

Return Value

S_OK
The method completed successfully.

Error Values
This method failed to complete successfully.

Remarks

A relationship can only be removed by name if it is a unique-naming relationship.

IRepository Interface

See Also Properties Methods Collections

When you define a tool information model, the classes, relationships, properties, and collections for the model are stored in a repository. Multiple tool information models may be stored in the same repository.

When to Use

Use the Repository Interface to create and access repository databases. You can also use the Repository Interface to create and access repository objects in a repository database.

Methods

IUnknown Method	Description
QueryInterface	Returns pointers to supported interfaces.
AddRef	Increments the reference count.
Release	Decrements the reference count.

IDispatch Method	Description
GetIDsOfNames	Maps a single member and a set of argument names to a corresponding set of dispatch identifiers.
GetTypeInfo	Retrieves a type information object, which can be used to get the type information for an interface.
GetTypeInfoCount	Retrieves the number of type information interfaces that an object provides (either 0 or 1).
Invoke	Provides access to properties and methods exposed by an Automation object.

IRepository Method	Description
Create	Creates a repository database.
CreateObject	Creates a new repository object.
get_Object	Retrieves the IRepositoryObject interface pointer for a repository object.
get_RootObject	Retrieves the IRepositoryObject interface pointer for the root repository object.
get_Transaction	Retrieves the IRepositoryTransaction interface pointer for this repository instance.
InternalIDToObjectID	Translate an internal identifier to an object identifier.
ObjectIDToInternalID	Translate an object identifier to an internal identifier.
Open	Opens a repository database.
Refresh	Refreshes unchanged cached repository data.

IRepository::Create

See Also

Use this method to create a new repository. The fundamental repository tables are automatically created in the new repository. An IRepositoryObject interface pointer to the root repository object is passed back to the caller.

```
HRESULT Create(
    BSTR                Connect,
    BSTR                User,
    BSTR                Password,
    long                fFlags,
    IRepositoryObject   **ppIRootObj
);
```

Parameters

Connect [in]
The ODBC connection string to be used for accessing the database server that will host your new repository.

User [in]
The user name to use for identification to the database server.

Password [in]
The password that matches the *User* input parameter.

fFlags [in]
Flags that determine database access and caching behavior for the open repository. For details, see the ConnectionFlags Enumeration.

**ppIRootObj* [out]
The IRepositoryObject interface pointer for the new repository's root repository object.

Return Value

S_OK
The method completed successfully.

Error Values
This method failed to complete successfully.

IRepository::CreateObject

See Also

Creates a new repository object. The specified COM interface pointer to the new object is passed back to the caller.

```
HRESULT CreateObject(
    VARIANT                 sTypeId,
    VARIANT                 sObjId,
    IRepositoryObject   *ppIReposObj
);
```

Parameters

sTypeId [in]
The type of the new object; that is, the object identifier of the class definition to which the new object conforms.

sObjId [in]
The object identifier to be assigned to the new object. Pass in OBJID_NULL to have the repository assign an object identifier for you.

**ppIReposObj* [out]
The IRepositoryObject interface pointer for the new repository object.

Return Value

S_OK
The method completed successfully.

Error Values
This method failed to complete successfully.

Remarks

The new object will automatically create persistent storage for itself.

IRepository::get_Object

See Also

Retrieves an IRepositoryObject interface pointer to the specified repository object.

```
HRESULT get_Object(
    VARIANT           sObjectId,
    IRepositoryObject **ppIReposObj
);
```

Parameters

sObjectId [in]
The object identifier of the repository object to be retrieved.

**ppIReposObj* [out]
The IRepository interface pointer for the repository object.

Return Value

S_OK
The method completed successfully.

Error Values
This method failed to complete successfully.

IRepository::get_RootObject

See Also

Use this method to obtain a pointer to the currently open repository's root object. The root object is the repository object to which all other repository objects are (either directly or indirectly) connected.

HRESULT get_RootObject(
IRepositoryObject ***ppIRootObj*
);

Parameters

**ppIRootObj* [out]
The IRepositoryObject interface pointer for the root repository object.

Return Value

S_OK
The method completed successfully.

Error Values
This method failed to complete successfully.

IRepository::get_Transaction

See Also

Retrieves the IRepositoryTransaction interface pointer for this repository instance. Use the IRepositoryTransaction interface to manage repository transactions for this repository instance.

HRESULT get_Transaction(
IRepositoryTransaction ***ppIRepTrans*
);

Parameters

**ppIRepTrans* [out]
The IRepositoryTransaction interface pointer.

Return Value

S_OK
The method completed successfully.

Error Values
This method failed to complete successfully.

IRepository::InternalIDToObjectID

See Also

This method translates an internal identifier into an object identifier. Internal identifiers are used by the repository engine to identify repository objects.

```
HRESULT InternalIDToObjectID(
    VARIANT    sInternalID,
    VARIANT    *sObjectId
);
```

Parameters

sInternalID [in]
The internal identifier for the repository object.

**sObjectId* [out]
The object identifier for the repository object.

Return Value

S_OK
The method completed successfully.

Error Values
This method failed to complete successfully.

Remarks

Repository object identifiers are globally unique, and are the same across repositories for the same object. Repository internal identifiers are unique only within the scope of a single repository.

The translation performed by this method is performed without loading the object in question. This enables database queries involving an object or relationship type identifier to be constructed without having to load the definition object.

IRepository::ObjectIDToInternalID

See Also

This method translates an object identifier into an internal identifier. Internal identifiers are used by the repository engine to identify repository objects.

```
HRESULT ObjectIDToInternalID(
    VARIANT    sObjectID,
    VARIANT    *sInternalId
);
```

Parameters

sObjectID [in]
The object identifier for the repository object.

**sInternalId* [out]
The internal identifier for the repository object.

Return Value

S_OK
The method completed successfully.

Error Values
This method failed to complete successfully.

Remarks

Repository object identifiers are globally unique, and are the same across repositories for the same object. Repository internal identifiers are unique only within the scope of a single repository.

The translation performed by this method is performed without loading the object in question. This enables database queries involving an object or relationship type identifier to be constructed without having to load the definition object.

IRepository::Open

See Also

Use this method to open a repository. An IRepositoryObject interface pointer to the root repository object is passed back to the caller.

```
HRESULT Open(
    BSTR              Connect,
    BSTR              User,
    BSTR              Password,
    long              fFlags,
    IRepositoryObject **ppIRootObj
);
```

Parameters

Connect [in]
The ODBC connection string to be used for accessing the database server that hosts your repository.

User [in]
The user name to use for identification to the database server.

Password [in]
The password that matches the *User* input parameter.

fFlags [in]
Flags that determine database access and caching behavior for the open repository. For details, see the ConnectionFlags Enumeration.

**ppIRootObj* [out]
The IRepositoryObject interface pointer for the open repository's root repository object.

Return Value

S_OK
The method completed successfully.

Error Values
This method failed to complete successfully.

IRepository::Refresh

See Also

This method refreshes all of the cached data for this repository instance. Only cached data that has not been changed *by the current process* is refreshed.

HRESULT Refresh(long *iMilliseconds* **);**

Parameters

iMilliseconds [in]
All unchanged data that has been in the cache for longer than *iMilliseconds* milliseconds is refreshed. Set this parameter to zero to refresh all unchanged data.

Return Value

S_OK
The method completed successfully.

Error Values
This method failed to complete successfully.

IRepositoryDispatch Interface

The **IRepositoryDispatch** interface is an enhanced **IDispatch** interface; in addition to all of the standard IDispatch methods, IRepositoryDispatch also provides access to the Properties collection. The Properties collection gives you a convenient mechanism to enumerate through all of the persistent properties and collections of an interface.

When you instantiate an Automation object that represents an object from your tool information model, and that object conforms to a class for which there is no custom implementation (in other words, you have provided no software implementation of the class), the repository will provide an interface implementation for you. This interface implementation uses IRepositoryDispatch as its dispatch interface.

When to Use

Use the IRepositoryDispatch Interface to access the properties and collections of a repository object, when no custom implementation is available.

Methods

IUnknown Method	Description
QueryInterface	Returns pointers to supported interfaces.
AddRef	Increments the reference count.
Release	Decrements the reference count.

IDispatch Method	Description
GetIDsOfNames	Maps a single member and a set of argument names to a corresponding set of dispatch identifiers.
GetTypeInfo	Retrieves a type information object, which can be used to get the type information for an interface.
GetTypeInfoCount	Retrieves the number of type information interfaces that an object provides (either 0 or 1).
Invoke	Provides access to properties and methods exposed by an Automation object.

IRepositoryDispatch Method	Description
get_Properties	Retrieves the IReposProperties interface pointer. The IReposProperties interface provides access to the Properties collection.

Remarks

The repository engine will only supply an interface implementation for you if your interface is defined to inherit from IDispatch or IRepositoryDispatch.

IRepositoryDispatch::get_Properties

See Also

This method retrieves the IReposProperties interface pointer. The IReposProperties interface provides methods to access the Properties collection. The Properties collection gives you a convenient mechanism to enumerate through all of the persistent properties and collections of an interface.

HRESULT get_Properties(IReposProperties ***ppIReposProps* **);**

Parameters

**ppIReposProps* [out]
The IReposProperties interface pointer.

Return Value

S_OK
The method completed successfully.

Error Values
This method failed to complete successfully.

IRepositoryErrorQueue Interface

Errors that occur while accessing a repository are saved on a repository error queue. A repository error queue is a collection of **REPOSERROR** structures. Individual elements on a repository error queue can be managed in much the same way that elements can be managed in other repository collections. This interface provides those management capabilities.

When to Use

Use the IRepositoryErrorQueue Interface to manage the errors that belong to a particular repository error queue. With this interface, you can:

- Get a count of the number of error elements in the collection.
- Enumerate the elements in the collection.
- Insert and remove error elements to and from the collection.
- Retrieve one of the error elements in the collection.

Methods

IUnknown Method	Description
QueryInterface	Returns pointers to supported interfaces.
AddRef	Increments the reference count.
Release	Decrements the reference count.

IRepositoryErrorQueue Method	Description
Count	Returns a count of the number of errors on the queue.
Insert	Inserts a new error onto the error queue, in the specified location.
Item	Retrieves the specified error from the error queue.
Remove	Removes the specified error from the error queue.
_NewEnum	Creates an enumerator object for the error queue.

IRepositoryErrorQueue::Count

See Also

This method returns the number of errors that are currently on the error queue.

ULONG Count(void);

Return Value

The number of error elements on the queue.

IRepositoryErrorQueue::Insert

See Also

This method inserts a new element into the error queue. The element can either be inserted at a specific location in the queue, or it can be appended to the end of the queue.

```
HRESULT Insert(
    ULONG          iIndex,
    REPOSERROR     *psError
);
```

Parameters

iIndex [in]
The index of the location in the error queue where this element is to be inserted. To insert an element at the beginning of the error queue, set this parameter to one. Set this parameter to zero to append the element to the end of the error queue.

**psError* [in]
The error information for the element to be inserted. The information from this structure is copied and the copy is placed on the error queue.

Return Value

S_OK
The method completed successfully.

Error Values
This method failed to complete successfully.

IRepositoryErrorQueue::Item

See Also

This method retrieves the specified element from the error queue. There are two variations of this method.

```
HRESULT Item(
    ULONG           iIndex,
    REPOSERROR      *psError
);

HRESULT Item(
    ULONG           iIndex,
    IErrorInfo      **ppIErrorInfo
);
```

Parameters

iIndex [in]
The index of the location in the error queue of the element to be retrieved.

**ppError* [out]
The repository error information structure with information from the retrieved element.

**ppIErrInfoObj* [out]
The interface pointer to an error information object for the retrieved element.

Return Value

S_OK
The method completed successfully.

Error Values
This method failed to complete successfully.

IRepositoryErrorQueue::Remove

See Also

This method removes the specified element from the error queue.

```
HRESULT Remove(
    ULONG    iIndex
);
```

Parameters

iIndex [in]
The index of the location in the error queue of the element to be removed.

Return Value

S_OK
The method completed successfully.

Error Values
This method failed to complete successfully.

IRepositoryErrorQueue::_NewEnum

See Also

This method creates an enumeration object for the error queue. An interface pointer for the enumeration object is passed back to the caller.

HRESULT _NewEnum(
IEnumRepositoryErrors ***ppIEnum*
);

Parameters

**ppIEnum* [out]
The interface pointer to the enumeration object.

Return Value

S_OK
The method completed successfully.

Error Values
This method failed to complete successfully.

IRepositoryErrorQueueHandler Interface

Errors that occur while accessing a repository are saved on a repository error queue. A repository error queue is a collection of **REPOSERROR** structures. Each thread of execution with an open repository instance can access one active error queue at a time.

When to Use

Use the IRepositoryErrorQueueHandler Interface to create a repository error queue, assign an error queue to a thread of execution, or retrieve an interface pointer to a thread's currently assigned error queue.

Methods

IUnknown Method	Description
QueryInterface	Returns pointers to supported interfaces.
AddRef	Increments the reference count.
Release	Decrements the reference count.

IRepositoryErrorQueueHandler Method	Description
CreateErrorQueue	Creates a new repository error queue.
SetErrorQueue	Sets the active error queue for a thread.
GetErrorQeueue	Retrieves an interface pointer to the currently active error queue for a thread.

IRepositoryErrorQueueHandler::CreateErrorQueue

See Also

This method creates a repository error queue. Once created, the error queue is available to be assigned to a thread context.

HRESULT CreateErrorQueue(
IRepositoryErrorQueue ***ppIErrorQueue*
);

Parameters

**ppIErrorQueue* [out]
The interface pointer for the newly created repository error queue.

Return Value

S_OK
The method completed successfully.

Error Values
This method failed to complete successfully.

IRepositoryErrorQueueHandler::GetErrorQueue

See Also

This method retrieves the repository error queue that is assigned to the current thread.

HRESULT GetErrorQueue(
    IRepositoryErrorQueue ***ppIErrorQueue*
);

Parameters

**ppIErrorQueue* [out]
The interface pointer for the error queue that is currently assigned to this thread.

Return Value

S_OK
The method completed successfully.

Error Values
This method failed to complete successfully.

IRepositoryErrorQueueHandler::SetErrorQueue

See Also

This method assigns the specified repository error queue to the current thread context.

HRESULT SetErrorQueue(
    IRepositoryErrorQueue **pIErrorQueue*
);

Parameters

pIErrorQueue [in]
The interface pointer for the error queue to be assigned to this thread.

Return Value

S_OK
The method completed successfully.

Error Values
This method failed to complete successfully.

IRepositoryItem Interface

The IRepositoryItem interface contains methods that are common to both repository objects and repository relationships. It contains all of the general purpose methods that are used to manage repository items.

When to Use

Use the IRepositoryItem Interface to:

- Retrieve an item's type or name.
- Obtain a lock on an item.
- Change the name of an item.
- Refresh the cached image of an item.
- Delete an item.
- Get a pointer to an alternate interface that the item exposes.
- Get the open repository instance through which the item is accessed.

Methods

IUnknown Method	Description
QueryInterface	Returns pointers to supported interfaces.
AddRef	Increments the reference count.
Release	Decrements the reference count.

IDispatch Method	Description
GetIDsOfNames	Maps a single member and a set of argument names to a corresponding set of dispatch identifiers.
GetTypeInfo	Retrieves a type information object, which can be used to get the type information for an interface.
GetTypeInfoCount	Retrieves the number of type information interfaces that an object provides (either 0 or 1).
Invoke	Provides access to properties and methods exposed by an Automation object.

IRepositoryDispatch Method	Description
get_Properties	Retrieves the IReposProperties interface pointer. The IReposProperties interface provides access to the Properties collection.

IRepositoryItem Method	Description
Delete	Deletes a repository item.
get_Interface	Retrieves an interface pointer to the specified item interface.
get_Name	Retrieves the name associated with an

	item.
get_Repository	Retrieves the IRepository interface pointer for an item's open repository instance.
get_Type	Retrieves the type of an item.
Lock	Locks the item.
put_Name	Sets the name associated with an item.
Refresh	Refreshes the cached image of the item.

IRepositoryItem::Delete

See Also

This method deletes a repository item. If the item is a repository object, any relationships involving this object are also deleted. If the item is a repository relationship, it is removed from its origin and destination collections.

HRESULT Delete(void);

Return Value

S_OK

The method completed successfully.

Error Values

This method failed to complete successfully.

IRepositoryItem::get_Interface

See Also

This method retrieves the interface pointer for an alternate interface that the item exposes. The specified interface must be an Automation interface; that is, it must support the methods of the **IDispatch** interface.

```
HRESULT get_Interface(
    VARIANT     whichInterface,
    IDispatch   **ppInterface
);
```

Parameters

whichInterface [in]
Specifies the interface you wish to access. This parameter can be the name of the interface, the interface identifier, or the object identifier of the interface definition object in the repository.

**ppInterface* [out]
The interface pointer for the Automation interface.

Return Value

S_OK
The method completed successfully.

Error Values
This method failed to complete successfully.

Remarks

Some objects expose multiple interfaces. This method is provided as a mechanism for the Automation programmer to be able to easily access alternate interfaces, when no type library is available for the item's class.

IRepositoryItem::get_Name

See Also

Retrieves the name associated with a repository item. For repository relationships, this is the name defined by the relationship. For repository objects, this is either:

1. The Name property of the INamedObject interface, if the object exposes that interface.
2. The name defined by a relationship for which the object is the destination object.

HRESULT get_Name(BSTR **pName* **);**

Parameters

**pName* [out]
The name associated with the item.

Return Value

S_OK
The method completed successfully.

Error Values
This method failed to complete successfully.

Remarks

A repository object can be the destination of multiple naming relationships, and each relationship can use a different name to refer to the object. If this is the case for the current item, the name returned by this method is the name from the first naming relationship that is found. However, if the repository object exposes the **INamedObject** interface, then the name that is returned is always the value of the Name property of that interface.

IRepositoryItem::get_Repository

See Also

Retrieves an **IRepository** interface pointer for the open repository instance through which this repository item was instantiated.

HRESULT get_Repository(IRepository ***ppIRepository* **);**

Parameters

**ppIRepository* [out]
The IRepository interface pointer for the open repository instance.

Return Value

S_OK
The method completed successfully.

Error Values
This method failed to complete successfully.

IRepositoryItem::get_Type

See Also

Use this method to obtain the object identifier of the repository definition object to which the repository item conforms. This is the *type* of the repository item.

HRESULT get_Type(VARIANT **psTypeId* **);**

Parameters

**psTypeId* [out]
The object identifier of this repository item's definition object.

Return Value

S_OK
The method completed successfully.

Error Values
This method failed to complete successfully.

IRepositoryItem::Lock

See Also

Use this method to lock a particular repository item. Locking the item prevents other processes from updating the item while you are working with it. The lock is released when you end the current transaction.

HRESULT Lock(void);

Return Value

S_OK

The method completed successfully.

Error Values

This method failed to complete successfully.

IRepositoryItem::put_Name

See Also

Sets the name associated with a repository item. For repository relationships, this is the name defined by the relationship. For repository objects, this is either or both of:

1. The Name property of the INamedObject interface, if the object exposes that interface.
2. The name defined by a relationship for which the object is the destination object.

HRESULT put_Name(BSTR *Name* **);**

Parameters

Name [in]
The name to be associated with the item.

Return Value

S_OK
The method completed successfully.

Error Values
This method failed to complete successfully.

Remarks

A repository object can be the destination of multiple relationships, and each relationship can use a different name to refer to the object. If this is the case for the current item, then all names for the item are set to the new name. Furthermore, if the repository object exposes the **INamedObject** interface, then the Name property of that interface is also set to the new name value.

IRepositoryItem::Refresh

See Also

Use this method to refresh the cached image of a particular repository item. Only unchanged cache data is refreshed.

HRESULT Refresh(long *iMilliseconds* **);**

Parameters

iMilliseconds [in]
All unchanged data relating to the item that has been in the cache for longer than *iMilliseconds* milliseconds is refreshed. Set to zero to refresh all unchanged data.

Return Value

S_OK
The method completed successfully.

Error Values
This method failed to complete successfully.

IRepositoryObject Interface

The IRepositoryObject interface provides methods to manage repository objects.

When to Use

Use the IRepositoryObject Interface to:

- Retrieve the object identifier or the internal identifier for a repository object.
- Retrieve a repository object's type or name.
- Obtain a lock on a repository object.
- Change the name of a repository object.
- Refresh the cached image of a repository object.
- Delete a repository object.
- Get a pointer to an alternate interface that the object exposes.
- Get the open repository instance through which the object is accessed.

Methods

IUnknown Method	Description
QueryInterface	Returns pointers to supported interfaces.
AddRef	Increments the reference count.
Release	Decrements the reference count.

IDispatch Method	Description
GetIDsOfNames	Maps a single member and a set of argument names to a corresponding set of dispatch identifiers.
GetTypeInfo	Retrieves a type information object, which can be used to get the type information for an interface.
GetTypeInfoCount	Retrieves the number of type information interfaces that an object provides (either 0 or 1).
Invoke	Provides access to properties and methods exposed by an Automation object.

IRepositoryDispatch Method	Description
get_Properties	Retrieves the IReposProperties interface pointer. The IReposProperties interface provides access to the Properties collection.

IRepositoryItem Method	Description
Delete	Deletes a repository item.
get_Interface	Retrieves an interface pointer to the specified item interface.
get_Name	Retrieves the name associated with an

	item.
get_Repository	Retrieves the IRepository interface pointer for an item's open repository instance.
get_Type	Retrieves the type of an item.
Lock	Locks the item.
put_Name	Sets the name associated with an item.
Refresh	Refreshes the cached image of the item.

IRepositoryObject Method	**Description**
get_InternalID	Retrieves the internal identifier for a repository object.
get_ObjectID	Retrieves the object identifier for a repository object.

IRepositoryObject::get_InternalID

See Also

Use this method to obtain the internal identifier for a repository object.

HRESULT get_InternalID(VARIANT **psInternalId* **);**

Parameters

**psInternalId* [out]
The internal identifier of this repository object.

Return Value

S_OK
The method completed successfully.

Error Values
This method failed to complete successfully.

IRepositoryObject::get_ObjectID

See Also

Use this method to retrieve the object identifier for a repository object.

HRESULT get_ObjectID(VARIANT **psObjectId* **);**

Parameters

**psObjectId* [out]
The object identifier of this repository object.

Return Value

S_OK
The method completed successfully.

Error Values
This method failed to complete successfully.

IRepositoryObjectStorage Interface

The IRepositoryObjectStorage interface initializes the memory image for a repository object. New repository objects are initialized as empty objects. For existing repository objects, the state of the object is retrieved from the repository database.

When to Use

The IRepositoryObjectStorage Interface is used by the repository engine to materialize repository objects in memory. It is not intended for use by repository applications.

Methods

IUnknown Method	Description
QueryInterface	Returns pointers to supported interfaces.
AddRef	Increments the reference count.
Release	Decrements the reference count.

IDispatch Method	Description
GetIDsOfNames	Maps a single member and a set of argument names to a corresponding set of dispatch identifiers.
GetTypeInfo	Retrieves a type information object, which can be used to get the type information for an interface.
GetTypeInfoCount	Retrieves the number of type information interfaces that an object provides (either 0 or 1).
Invoke	Provides access to properties and methods exposed by an Automation object.

IRepositoryObjectStorage Method	Description
get_PropertyInterface	Retrieves an IRepositoryDispatch interface pointer for accessing the persistent members of one of the item's supported interfaces.
InitNew	Initializes memory for a new repository object.
Load	Initializes memory for an existing repository object.

IRepositoryObjectStorage::get_PropertyInterface

See Also

This method retrieves an **IRepositoryDispatch** interface pointer for accessing the persistent members of one of the object's supported Automation interfaces.

The IRepositoryDispatch interface can be used to get and set member values for the interface specified by the *InterfaceId* input parameter. The interface must be one that is exposed by this object.

```
HRESULT get_PropertyInterface(
    REFIID                  InterfaceId,
    IRepositoryDispatch     **ppInterface
);
```

Parameters

InterfaceId [in]
The interface identifier of the interface whose properties are to be accessed.

**ppInterface* [out]
The IRepositoryDispatch interface pointer.

Return Value

S_OK
The method completed successfully.

Error Values
This method failed to complete successfully.

Remarks

IRepositoryDispatch does not perform any parameter validation, and cannot be used to access custom methods or non-persistent properties. It is intended for use by custom class implementers.

IRepositoryObjectStorage::InitNew

See Also

The repository engine uses this method to initialize a new repository object in memory.

```
HRESULT InitNew(
    IRepository     *pIRepository,
    INTID               sInternalId
);
```

Parameters

**pIRepository* [in]
The repository that contains this object.

sInternalId [in]
The internal identifier for the new object.

Return Value

S_OK
The method completed successfully.

Error Values
This method failed to complete successfully.

IRepositoryObjectStorage::Load

See Also

The repository engine uses this method to load a repository object's state into memory from the repository database.

```
HRESULT Load(
    IRepository    *pIRepository,
    INTID          sInternalId
);
```

Parameters

**pIRepository* [in]
The repository that contains this object.

sInternalId [in]
The internal identifier for the repository object.

Return Value

S_OK
The method completed successfully.

Error Values
This method failed to complete successfully.

IRepositoryTransaction Interface

The repository supports transactional processing. Repository methods that are reading data from the repository may be executed outside of a transaction, but methods that write data must be bracketed within a transaction. Only one transaction may be active at a time for each opened repository instance. Nesting of Begin/Commit method invocations is permitted, but no actual nesting of transactions occurs.

When to Use

Use the Repository Transaction Interface to begin, commit, or abort a repository transaction. You can also use this interface to retrieve the information about the transactional state of an opened repository instance, and to set transaction options.

Methods

IUnknown Method	Description
QueryInterface	Returns pointers to supported interfaces.
AddRef	Increments the reference count.
Release	Decrements the reference count.

IDispatch Method	Description
GetIDsOfNames	Maps a single member and a set of argument names to a corresponding set of dispatch identifiers.
GetTypeInfo	Retrieves a type information object, which can be used to get the type information for an interface.
GetTypeInfoCount	Retrieves the number of type information interfaces that an object provides (either 0 or 1).
Invoke	Provides access to properties and methods exposed by an Automation object.

IRepositoryTransaction Method	Description
Abort	Cancels a currently active transaction.
Begin	Begins a new transaction.
Commit	Commits an active transaction.
Flush	Flushes uncommitted changes to the repository database.
GetOption	Retrieves a transaction option.
get_Status	Tells whether or not there is a currently active transaction.
SetOption	Sets a transaction option.

IRepositoryTransaction::Abort

See Also

This method cancels the currently active transaction for an open repository. All updates made during the transaction are undone. The nested transaction count is set to zero.

HRESULT Abort(void);

Return Value

S_OK

The method completed successfully.

Error Values

This method failed to complete successfully.

IRepositoryTransaction::Begin

See Also

This method increments the nested transaction count by one. If there is no active transaction, this method begins a transaction for the open repository instance.

HRESULT Begin(void);

Return Value

S_OK

The method completed successfully.

Error Values

This method failed to complete successfully.

IRepositoryTransaction::Commit

See Also

This method decrements the nested transaction count for an open repository instance. If the currently active transaction is not nested, all changes made to repository data within the transaction are committed to the repository database. A transaction is not nested if the nested transaction count equals one.

HRESULT Commit(void);

Return Value

S_OK

The method completed successfully.

Error Values

This method failed to complete successfully.

IRepositoryTransaction::Flush

See Also

This method flushes cached changes to the repository database.

Unless you have set the *exclusive-write-through-mode* transaction option, changes that you make within the scope of a transaction are cached, and are not written to the database until the transaction is committed. If a concurrent SQL query is run against the repository database, the result of the query will not reflect the uncommitted changes (this is normally the desired behavior).

If your repository application must be able to see uncommitted changes in SQL queries, you can use the Flush method to write uncommitted changes to the repository database. All changes made within the scope of the current transaction are flushed. Flushing uncommitted changes does not affect your ability to cancel a transaction via the **Abort** method.

HRESULT Flush(void);

Return Value

S_OK
  The method completed successfully.

Error Values
  This method failed to complete successfully.

IRepositoryTransaction::GetOption

See Also

Use this method to retrieve the value of a transaction option for an open repository instance.

```
HRESULT GetOption(
      long        iOption,
      VARIANT     *psValue
   );
```

Parameters

iOption [in]
The transaction option to retrieve. For a list of valid values and their meanings, see the TransactionFlags Enumeration.

**psValue* [out]
The value of the specified transaction option.

Return Value

S_OK
The method completed successfully.

Error Values
This method failed to complete successfully.

IRepositoryTransaction::get_Status

See Also

Use this method to determine whether or not there is a currently active transaction.

HRESULT get_Status(long ***piStatus*** **);**

Parameters

**piStatus* [out]
The current transaction status. If the value is zero, no transaction is active. If the value is nonzero, a transaction is active.

Return Value

S_OK
The method completed successfully.

Error Values
This method failed to complete successfully.

Remarks

A transaction is considered active until the **Commit** method has successfully executed and the nested transaction count has been decremented to zero. Depending upon the data-flushing capabilities of the underlying database server, the data associated with a committed transaction may or may not be written to the physical storage device when the Commit method returns control to its caller.

IRepositoryTransaction::SetOption

See Also

Use this method to set one of the transaction options for an open repository instance. You cannot set a transaction option while a transaction is active.

```
HRESULT SetOption(
    ULONG       iOption,
    VARIANT     sValue
);
```

Parameters

iOption [in]
The transaction option to set. For a list of valid values and their meanings, see the TransactionFlags Enumeration.

sValue [in]
The value of the specified transaction option.

Return Value

S_OK
The method completed successfully.

Error Values
This method failed to complete successfully.

IRepositoryODBC Interface

The repository engine stores information in a SQL database. The repository engine connects to the database server via an ODBC connection. The IRepositoryODBC interface provides you with access to the database through the same (or a similar) ODBC connection.

Care should be taken when accessing the repository database directly, especially when sharing the repository's ODBC connection. Specific restrictions are defined in the detailed information for each interface method. Directly accessing the repository database in a read-only manner is generally considered safe; however, if you tune your repository application to be dependent upon specific features of your database server, you limit the portability of your application.

When to Use

Use the IRepositoryODBC Interface to obtain or release an ODBC connection handle, or to retrieve the ODBC connection string used by the repository engine.

To obtain a pointer to this interface, use the IRepository::QueryInterface method.

Methods

IUnknown Method	Description
QueryInterface	Returns pointers to supported interfaces.
AddRef	Increments the reference count.
Release	Decrements the reference count.

IDispatch Method	Description
GetIDsOfNames	Maps a single member and a set of argument names to a corresponding set of dispatch identifiers.
GetTypeInfo	Retrieves a type information object, which can be used to get the type information for an interface.
GetTypeInfoCount	Retrieves the number of type information interfaces that an object provides (either 0 or 1).
Invoke	Provides access to properties and methods exposed by an Automation object.

IRepositoryODBC Method	Description
FreeConnection	Releases an ODBC connection handle.
get_ConnectionString	Retrieves the ODBC connection string that the repository engine uses to obtain an ODBC connection.
GetNewConnection	Obtains a new ODBC connection handle using the same connection settings that the repository engine is using to access the repository database.
get_ReposConnection	Retrieves the ODBC connection handle that the repository engine is using to

access the repository database.

IRepositoryODBC::FreeConnection

See Also

This method frees an ODBC connection handle.

HRESULT FreeConnection(long *Hdbc* **);**

Parameters

Hdbc [in]
The ODBC connection handle to be released.

Return Value

S_OK
The method completed successfully.

Error Values
This method failed to complete successfully.

Remarks

Use this method to free the handle obtained via either the **get_ReposConnection** method or the **GetNewConnection** method before releasing your open repository instance.

IRepositoryODBC::get_ConnectionString

See Also

This method retrieves the ODBC connection string that the repository engine uses to obtain an ODBC connection to the repository database.

HRESULT get_ConnectionString(BSTR *szString* **);**

Parameters

szString [out]
The ODBC connection string.

Return Value

S_OK
The method completed successfully.

Error Values
This method failed to complete successfully.

Remarks

The ODBC connection string can contain user identification and password information. Take care to protect this information from exposure to unauthorized access.

IRepositoryODBC::GetNewConnection

See Also

This method obtains a new ODBC connection handle using the same ODBC connection string that the repository engine is using to access the repository database. Using a new ODBC connection handle isolates you from changes made by the repository engine.

HRESULT GetNewConnection(long **pHdbc* **);**

Parameters

**pHdbc* [out]
A new ODBC connection handle.

Return Value

S_OK
The method completed successfully.

Error Values
This method failed to complete successfully.

Remarks

Be sure to free the handle obtained via this method before releasing your open repository instance. To free the connection handle, use the **FreeConnection** method.

IRepositoryODBC::get_ReposConnection

See Also

This method retrieves the ODBC connection handle that the repository engine is using to access the repository database.

HRESULT get_ReposConnection(long **pHdbc* **);**

Parameters

**pHdbc* [out]
A copy of the repository engine's ODBC connection handle.

Return Value

S_OK
The method completed successfully.

Error Values
This method failed to complete successfully.

Remarks

If you use the repository engine's ODBC connection handle, you are not isolated from changes made by the repository engine. For example, uncommitted changes made by the repository engine will be visible to your application.

When using the repository engine's ODBC connection handle, you must not change the state of the handle in a way that is incompatible with the repository engine. Specifically, do not:

- Change any ODBC connection options.
- Perform any accesses concurrent with repository method invocations.
- Directly commit or rollback a database transaction. The IRepositoryTransaction interface must always be used to manage transactions.

Be sure to free the handle obtained via this method before releasing your open repository instance. To free the connection handle, use the **FreeConnection** method.

IReposProperties Interface

The **IReposProperties** interface provides access to the Properties collection. The Properties collection gives you a convenient mechanism to enumerate through all of the persistent properties and collections of an interface, when you don't already know the names of all of the interface members.

When you instantiate an Automation object that represents an object from your tool information model, and that object conforms to a class for which there is no custom implementation (in other words, you have provided no software implementation of the class), the repository will provide an interface implementation for you. This interface implementation uses IRepositoryDispatch as its dispatch interface. This dispatch interface contains one additional method, the get_Properties method, that returns an IReposProperties interface pointer.

This support enables the Automation programmer to use syntax like:

```
Dim firstProperty As ReposProperty
Set firstProperty = repObject.Properties(1)
```

The second statement is resolved like this:

1. In this example, *repObject* is an Automation instantiation of a repository object where the default implementation has been used.
2. The **Properties** term is the Automation level name for the **get_Properties** method that is supplied by the **IRepositoryDispatch** dispatch interface.
3. The get_Properties method returns the interface pointer to the **IReposProperties** interface.
4. The default method of the IReposProperties interface is the **get_Item** method, which returns an **IReposProperty** interface pointer for the specified property object in the Properties collection.

At this point, the Automation programmer has access to the first property in the collection via the *firstProperty* object variable.

When to Use

Use the IReposProperties interface to access the properties and collections of a repository object, when no custom implementation is available, and you do not already know what members are exposed by the object's interface. With the IReposProperties interface, you can:

- Get a count of the number of members in the collection.
- Retrieve an IReposProperty interface pointer to one of the members in the collection.
- Enumerate the members in the collection.

Methods

IUnknown Method	Description
QueryInterface	Returns pointers to supported interfaces.
AddRef	Increments the reference count.
Release	Decrements the reference count.

IDispatch Method	Description
GetIDsOfNames	Maps a single member and a set of argument names to a corresponding set of dispatch identifiers.
GetTypeInfo	Retrieves a type information object, which can be used to get the type

	information for an interface.
GetTypeInfoCount	Retrieves the number of type information interfaces that an object provides (either 0 or 1).
Invoke	Provides access to properties and methods exposed by an Automation object.

IReposProperties Method	**Description**
get_Count	Retrieves the count of the number of members in the collection.
get_Item	Retrieves the IReposProperty interface pointer for the specified member of the collection.
_NewEnum	Retrieves a standard Automation enumeration interface pointer for the collection.

Remarks

Only persistent members (that is, members that are stored in the repository) are represented in the Properties collection.

IReposProperties::get_Count

See Also

This method is used to retrieve a count of the number of persistent members (properties and collections) that are in the Properties collection.

HRESULT get_Count(long **piCount* **);**

Parameters

**piCount* [out]
The number of members in the collection.

Return Value

S_OK
The method completed successfully.

Error Values
This method failed to complete successfully.

IReposProperties::get_Item

See Also

This method retrieves the specified member from the Properties collection.

```
HRESULT get_Item(
    VARIANT          sItem,
    IReposProperty   **ppIReposProperty
);
```

Parameters

sItem [in]
Identifies the item to be retrieved from the collection. This parameter can be either the index or the name of the member.

**ppIReposProperty* [out]
The **IReposProperty** interface pointer for the specified collection member.

Return Value

S_OK
The method completed successfully.

Error Values
This method failed to complete successfully.

IReposProperties::_NewEnum

See Also

This method retrieves an enumeration interface pointer for the Properties collection. This interface is a standard Automation enumeration interface. It supports the Clone, Next, Reset, and Skip methods. You can use the enumeration interface to step through the members in the collection.

HRESULT _NewEnum(IUnknown ***ppIEnumProps* **);**

Parameters

**ppIEnumProps* [out]
The enumeration interface pointer.

Return Value

S_OK
The method completed successfully.

Error Values
This method failed to complete successfully.

IReposProperty Interface

The **IReposProperty** interface provides access to a persistent member (a property or collection) of a tool information model interface.

When you instantiate an Automation object that represents an object from your tool information model, and that object conforms to a class for which there is no custom implementation (in other words, you have provided no software implementation of the class), the repository will provide an interface implementation for you. This interface implementation uses **IRepositoryDispatch** as its dispatch interface.

The IRepositoryDispatch interface is an enhanced **IDispatch** interface; in addition to all of the standard IDispatch methods, IRepositoryDispatch also provides access to the Properties collection. The Properties collection gives you a convenient mechanism to enumerate through all of the persistent properties and collections of an interface. The **IReposProperty** interface can be used to access the individual members in the Properties collection.

This support enables the Automation programmer to use syntax like:

```
Dim firstPropName As String
Let firstPropName = repObject.Properties(1).Name
```

The second statement resolves like this:

1. In this example, *repObject* is an Automation instantiation of a repository object where the default implementation has been used.
2. The **Properties** term is the Automation level name for the **get_Properties** method that is supplied by the **IRepositoryDispatch** dispatch interface.
3. The get_Properties method returns the interface pointer to the **IReposProperties** interface.
4. The default method of the IReposProperties interface is the **get_Item** method, which returns an **IReposProperty** interface pointer for the specified property object in the Properties collection.
5. The **Name** term is the Automation level name for the **get_Name** method that is supplied by the IReposProperty interface.

At this point, the Automation programmer has access to the name of the first property in the collection via the *firstPropName* variable.

When to Use

Use the IReposProperty interface to access a persistent interface member, when no custom implementation is available, and you do not already know the type or name of the member. With this interface, you can:

- Retrieve the name of a property or collection.
- Retrieve the type of a property or collection.
- Get or set the value of a property.

Methods

IUnknown Method	Description
QueryInterface	Returns pointers to supported interfaces.
AddRef	Increments the reference count.
Release	Decrements the reference count.

IDispatch Method	Description
GetIDsOfNames	Maps a single member and a set of

	argument names to a corresponding set of dispatch identifiers.
GetTypeInfo	Retrieves a type information object, which can be used to get the type information for an interface.
GetTypeInfoCount	Retrieves the number of type information interfaces that an object provides (either 0 or 1).
Invoke	Provides access to properties and methods exposed by an Automation object.

IReposProperty Method	**Description**
get_Type	Retrieves the type of a persistent interface member.
get_Name	Retrieves the name of a persistent interface member.
get_Value	Retrieves the value of a persistent interface member.
put_Value	Sets the value of a persistent property.

Remarks

Only persistent members (that is, members that are stored in the repository) can be accessed by the IReposProperty interface.

IReposProperty::get_Name

See Also

This method is used to retrieve the name of a persistent interface member (a property or collection).

HRESULT get_Name(BSTR **pName* **);**

Parameters

**pName* [out]
The name of the member.

Return Value

S_OK
The method completed successfully.

Error Values
This method failed to complete successfully.

IReposProperty::get_Type

See Also

This method retrieves the type of a persistent property or collection; that is, it returns the object identifier of the definition object to which the member conforms.

HRESULT get_Type(VARIANT **psTypeId* **);**

Parameters

**psTypeId* [out]
The object identifier of the member's definition object.

Return Value

S_OK
The method completed successfully.

Error Values
This method failed to complete successfully.

IReposProperty::get_Value

See Also

This method retrieves the value of a persistent interface member (a property or collection). If the member is a collection, the retrieved value is a pointer to the interface that supports that type of collection.

HRESULT get_Value(VARIANT **psValue* **);**

Parameters

**psValue* [out]
The property value.

Return Value

S_OK
The method completed successfully.

Error Values
This method failed to complete successfully.

IReposProperty::put_Value

See Also

This method sets the value of a persistent interface property. The type of the input parameter is converted to the storage data type of the property. If the type of the input parameter cannot be successfully converted to the storage data type, this method will return an error.

HRESULT put_Value(VARIANT *sValue* **);**

Parameters

sValue [in]
The property value to be set.

Return Value

S_OK
The method completed successfully.

Error Values
This method failed to complete successfully.

Remarks

You cannot set the value of a read-only property or a collection.

IClassDef Interface

See Also Properties Methods Collections

The IClassDef interface helps you create tool information models, by adding interfaces to a class. To insert a *new* class definition into a tool information model, use the **IReposTypeLib** interface.

To complete a class definition, once you have added all of the interfaces, commit the transaction that brackets your class definition modifications.

When to Use

Use the IClassDef interface to:

- Add a new or existing interface to a class definition.
- Retrieve the global identifier for the class.
- Access the collection of interfaces that are part of a class definition.

Properties

Property	Description
ClassID	The global identifier of the class.

Methods

IUnknown Method	Description
QueryInterface	Returns pointers to supported interfaces.
AddRef	Increments the reference count.
Release	Decrements the reference count.

IDispatch Method	Description
GetIDsOfNames	Maps a single member and a set of argument names to a corresponding set of dispatch identifiers.
GetTypeInfo	Retrieves a type information object, which can be used to get the type information for an interface.
GetTypeInfoCount	Retrieves the number of type information interfaces that an object provides (either 0 or 1).
Invoke	Provides access to properties and methods exposed by an Automation object.

IRepositoryDispatch Method	Description
get_Properties	Retrieves the IReposProperties interface pointer. The IReposProperties interface provides access to the Properties collection.

IClassDef Method	Description
AddInterface	Adds an existing interface to the class definition.

CreateInterfaceDef	Creates a new interface and adds it to the class definition.
ObjectInstances	Materializes an IObjectCol interface pointer for the collection of all objects in the repository that conform to this class.

Collections

Collection	Description
Interfaces	The collection of all interfaces that are implemented by a class.

IClassDef::AddInterface

See Also

The AddInterface method adds an existing interface to the collection of interfaces that are implemented by a particular class.

```
HRESULT AddInterface(
    IInterfaceDef   *plInterfaceDef,
    BSTR            Flags
);
```

Parameters

plInterfaceDef [in]
The interface pointer for the interface that is to be added to the collection of interfaces that are implemented by this class.

Flags [in]
If the interface that you are adding is the default interface for the class, pass in the string "Default". Otherwise, pass in a null string.

Return Value

S_OK
The method completed successfully.

Error Values
This method failed to complete successfully.

IClassDef ClassID Property

See Also

The global identifier that is assigned to this class.

Dispatch Identifier: DISPID_ClassID

Property Data Type: GUID

IClassDef::CreateInterfaceDef

See Also

The CreateInterfaceDef method creates a new interface definition, and adds the interface to the collection of interfaces that are implemented by the class.

```
HRESULT CreateInterfaceDef(
    VARIANT         sObjId,
    BSTR            Name,
    VARIANT         sIID,
    IInterfaceDef   *pIAncestor,
    BSTR            Flags,
    IInterfaceDef   **ppIInterfaceDef
);
```

Parameters

sIObjId [in]
The object identifier to be assigned to the new interface definition object. If this parameter is set to OBJID_NULL, the repository assigns an object identifier for you.

Name [in]
The name of the interface that is to be created.

sIID [in]
The global identifier associated with the signature for this interface. If there is none, set this parameter to zero.

**pIAncestor* [in]
The interface pointer to the base interface from which the interface being added is derived.

Flags [in]
If the interface that you are adding is the default interface for the class, pass in the string "Default". Otherwise, pass in a null string.

**ppIInterfaceDef* [out]
The interface pointer for the new interface.

Return Value

S_OK
The method completed successfully.

Error Values
This method failed to complete successfully.

IClassDef Interfaces Collection

See Also

The collection of all interfaces that are implemented by this class.

Dispatch Identifier: DISPID_Ifaces (32)

Collection Descriptor	Descriptor Value
Relationship Type	Class-Implements-Interface
Source Is Origin	Yes
Minimum Collection Size	Zero
Maximum Collection Size	Many
Sequenced Collection	No
Deletes Propagated	No
Destinations Named	No
Case Sensitive Names	Not applicable
Unique Names	Not applicable

IClassDef::ObjectInstances

See Also

This method materializes an IObjectCol interface pointer for the collection of all objects in the repository that conform to this class.

HRESULT ObjectInstances(IObjectCol ***ppIObjectCol* **);**

Parameters

**ppIObjectCol* [out]
The interface pointer for the object collection.

Return Value

S_OK
The method completed successfully.

Error Values
This method failed to complete successfully.

ICollectionDef Interface

See Also Properties Methods Collections

A collection type (also referred to as a collection definition) defines how instances of a particular type of collection will behave. The properties of the collection type determine:

- The minimum and maximum number of items in a collection.
- Whether or not the collection type is an origin collection type.
- Whether or not the collection type permits the naming of destination objects, and if so, whether those names are case sensitive, and required to be unique.
- Whether or not the collection type permits the explicit sequencing of items in the collection.
- What happens to related objects when objects or relationships in the collection are deleted.

The kind of relationship that a particular collection type uses to relate objects to each other is determined by its CollectionItem collection. The CollectionItem collection associates a single relationship type to the collection type.

To add a new collection type, use the **IInterfaceDef** interface.

When to Use

Use the ICollectionDef interface to retrieve or modify the properties of a collection type, or to determine the kind of relationship that the collection implements.

Methods

IUnknown Method	Description
QueryInterface	Returns pointers to supported interfaces.
AddRef	Increments the reference count.
Release	Decrements the reference count.

IDispatch Method	Description
GetIDsOfNames	Maps a single member and a set of argument names to a corresponding set of dispatch identifiers.
GetTypeInfo	Retrieves a type information object, which can be used to get the type information for an interface.
GetTypeInfoCount	Retrieves the number of type information interfaces that an object provides (either 0 or 1).
Invoke	Provides access to properties and methods exposed by an Automation object.

IRepositoryDispatch Method	Description
get_Properties	Retrieves the IReposProperties interface pointer. The IReposProperties interface provides access to the Properties collection.

Properties

Property	Description
Flags	Flags that determine the behavior of this type of collection.
IsOrigin	Indicates whether or not collections of this type are origin collections.
MaxCount	The maximum number of target objects that can be contained in a collection of this type.
MinCount	The minimum number of target objects that must be contained in a collection of this type.

Collections

Collection	Description
CollectionItem	The collection of one relationship type that defines the relationship between target objects of this type of collection and a single source object.

ICollectionDef Flags Property

See Also

For a particular type of collection, the **Flags** property determines:

- Whether or not the collection type permits the naming of destination objects, and if so, whether those names are case sensitive, and required to be unique.
- Whether or not the collection type permits the explicit sequencing of items in the collection.
- What happens to related objects when objects or relationships in the collection are deleted.

See the CollectionDefFlags Enumeration for a list of values and their specific purposes.

Dispatch Identifier: DISPID_ColFlags (54)

Property Data Type: long

ICollectionDef IsOrigin Property

See Also

This property indicates whether or not collections of this type are origin collections.

Dispatch Identifier: DISPID_IsOrigin (57)

Property Data Type: Boolean

ICollectionDef MaxCount Property

See Also

This property specifies the maximum number of target objects that can be contained in a collection of this type. This property is maintained for informational purposes, and is not enforced by the repository engine.

Dispatch Identifier: DISPID_MaxCount (56)

Property Data Type: short

ICollectionDef MinCount Property

See Also

This property specifies the minimum number of target objects that must be contained in a collection of this type. This property is maintained for informational purposes, and is not enforced by the repository engine.

Dispatch Identifier: DISPID_MinCount (55)

Property Data Type: short

ICollectionDef CollectionItem Collection

See Also

The collection of one relationship type that defines the relationship between target objects of this type of collection and a single source object.

Dispatch Identifier: DISPID_CollectionItem (38)

Collection Descriptor	Descriptor Value
Relationship Type	Collection-Contains-Items
Source Is Origin	Yes
Minimum Collection Size	Zero
Maximum Collection Size	One
Sequenced Collection	No
Deletes Propagated	No
Destinations Named	No
Case Sensitive Names	Not applicable
Unique Names	Not applicable

IInterfaceDef Interface

See Also Properties Methods Collections

The properties, methods, and collections that a class implements are organized into functionally related groups. Each group is implemented as a COM interface. The properties, methods, and collections of each interface are *members* of the interface. An interface definition is the template to which an interface conforms.

To add a new interface to the repository, use the **IClassDef** interface or the **IReposTypeLib** interface.

When to Use

Use the IInterfaceDef interface to:

- Retrieve or modify properties of an interface definition.
- Determine which members are attached to an interface definition.
- Determine which classes implement an interface.
- Determine the base interface from which an interface derives.
- Determine what interfaces derive from a particular interface.
- Determine what repository objects expose a particular interface.
- Add a new property, method or collection type to an interface definition.

Properties

Property	Description
Flags	Flags that specify whether the interface is extensible, and whether the interface should be visible to Automation interface queries.
InterfaceID	The global interface identifier for the interface.
TableName	The name of the SQL table that is used to store instance information for the properties of the interface.

Methods

IUnknown Method	Description
QueryInterface	Returns pointers to supported interfaces.
AddRef	Increments the reference count.
Release	Decrements the reference count.

IDispatch Method	Description
GetIDsOfNames	Maps a single member and a set of argument names to a corresponding set of dispatch identifiers.
GetTypeInfo	Retrieves a type information object, which can be used to get the type information for an interface.
GetTypeInfoCount	Retrieves the number of type information interfaces that an object provides (either 0 or 1).

Invoke	Provides access to properties and methods exposed by an Automation object.

IRepositoryDispatch Method	Description
get_Properties	Retrieves the IReposProperties interface pointer. The IReposProperties interface provides access to the Properties collection.

IInterfaceDef Method	Description
CreateMethodDef	Creates a new method definition, and attaches it to the interface definition.
CreatePropertyDef	Creates a new property definition, and attaches it to the interface definition.
CreateRelationshipColDef	Creates a relationship collection type. The collection type is attached to the interface definition.
ObjectInstances	Materializes an IObjectCol interface pointer for the collection of all objects in the repository that expose this interface.

Collections

Collection	Description
Ancestor	The collection of one base interface from which this interface derives.
Classes	The collection of classes that implement the interface.
Descendants	The collection of other interfaces that derive from this interface.
Members	The collection of members that are attached to the interface definition.

IInterfaceDef Flags Property

See Also

This property contains flags that specify whether the interface is extensible, and whether the interface should be visible to Automation interface queries. See the InterfaceDefFlags Enumeration for a list of values and their specific purposes.

Dispatch Identifier: DISPID_IfaceFlags (50)

Property Data Type: long

IInterfaceDef InterfaceID Property

See Also

This property is the global interface identifier for the interface.

Dispatch Identifier: DISPID_InterfaceID (48)

Property Data Type: GUID

IInterfaceDef TableName Property

See Also

The name of the SQL table that is used to store instance information for the properties of the interface. The length of the name must be 30 characters or less.

Dispatch Identifier: DISPID_TableName (49)

Property Data Type: string

IInterfaceDef::CreateMethodDef

See Also

This method creates a new method definition and attaches it to the interface definition.

```
HRESULT CreateMethodDef(
    VARIANT           sObjId,
    BSTR              Name,
    long              iDispId,
    IInterfaceMember  **ppIMethodDef
);
```

Parameters

sObjId [in]
The object identifier to be used for the new method definition object. The repository engine will assign an object identifier if you set this parameter to OBJID_NULL.

Name [in]
The name of the new method.

iDispId [in]
The dispatch identifier to be used for accessing the new method.

**ppIMethodDef* [out]
The interface pointer for the newly created method definition.

Return Value

S_OK
The method completed successfully.

Error Values
This method failed to complete successfully.

IInterfaceDef::CreatePropertyDef

See Also

This method creates a new property definition and attaches it to the interface definition.

```
HRESULT CreatePropertyDef (
    VARIANT         sObjId,
    BSTR            Name,
    long            iDispId,
    short           iCType,
    IPropertyDef    **ppIPropertyDef
);
```

Parameters

sObjId [in]
The object identifier to be used for the new property definition object. The repository engine will assign an object identifier if you set this parameter to OBJID_NULL.

Name [in]
The name of the new property.

iDispId [in]
The dispatch identifier to be used for accessing the new property.

iCType [in]
The C data type of the property. For a definition of valid values, see the *ODBC Programmer's Reference*.

**ppIPropertyDef* [out]
The interface pointer for the newly created property definition.

Return Value

S_OK
The method completed successfully.

Error Values
This method failed to complete successfully.

IInterfaceDef::CreateRelationshipColDef

See Also

This method creates a new collection type, attaches it to this interface, and associates it with the specified relationship type.

```
HRESULT CreateRelationshipColDef(
    VARIANT          sObjId,
    BSTR             Name,
    long             iDispId,
    boolean          IsOrigin,
    short            fFlags,
    IReposTypeInfo   *pIRelshipDef,
    ICollectionDef   **pICollectionDef
);
```

Parameters

sObjId [in]
The object identifier for the collection type. The repository engine will assign an object identifier if you set this parameter to OBJID_NULL.

Name [in]
The name of the new collection type.

iDispId [in]
The dispatch identifier to be used for Automation access to collections of this type.

IsOrigin [in]
Specifies whether collections of this type are origin collections.

fFlags [in]
Flags that specify naming, sequencing, and delete propagation behavior for the collection type. See the CollectionDefFlags Enumeration for a list of values and their specific purposes.

**pIRelshipDef* [in]
The interface pointer for the relationship definition object to which this collection type is connected.

**ppICollectionDef* [out]
The interface pointer for the new collection definition object.

Return Value

S_OK
The method completed successfully.

Error Values
This method failed to complete successfully.

Remarks

By default, the collection definition specifies that zero to many items are permitted in collections of this type. To specify a different minimum and maximum item count for the new collection type, change the MinCount and MaxCount properties *before committing the transaction* that contains this method invocation.

IInterfaceDef::ObjectInstances

See Also

This method materializes an IObjectCol interface pointer for the collection of all objects in the repository that expose this interface.

HRESULT ObjectInstances(IObjectCol ***ppIObjectCol* **);**

Parameters

**ppIObjectCol* [out]
The interface pointer for the object collection.

Return Value

S_OK
The method completed successfully.

Error Values
This method failed to complete successfully.

IInterfaceDef Classes Collection

See Also

This collection specifies which classes implement the interface.

Dispatch Identifier: DISPID_Classes (33)

Collection Descriptor	Descriptor Value
Relationship Type	Class-Implements-Interface
Source Is Origin	No
Minimum Collection Size	Zero
Maximum Collection Size	Many
Sequenced Collection	No
Deletes Propagated	No
Destinations Named	No
Case Sensitive Names	Not Applicable
Unique Names	Not Applicable

IInterfaceDef Members Collection

See Also

This collection specifies which members are attached to the interface.

Dispatch Identifier: DISPID_Members (36)

Collection Descriptor	Descriptor Value
Relationship Type	Interface-Has-Members
Source Is Origin	Yes
Minimum Collection Size	Zero
Maximum Collection Size	Many
Sequenced Collection	Yes
Deletes Propagated	Yes
Destinations Named	Yes
Case Sensitive Names	No
Unique Names	Yes

IInterfaceDef Ancestor Collection

See Also

This collection specifies the one base interface from which this interface derives.

Dispatch Identifier: DISPID_Ancestor (34)

Collection Descriptor	Descriptor Value
Relationship Type	Interface-InheritsFrom-Interface
Source Is Origin	Yes
Minimum Collection Size	One
Maximum Collection Size	One
Sequenced Collection	No
Deletes Propagated	No
Destinations Named	No
Case Sensitive Names	Not applicable
Unique Names	Not applicable

IInterfaceDef Descendants Collection

See Also

This collection specifies other interfaces that derive from this interface..

Dispatch Identifier: DISPID_Descendants (35)

Collection Descriptor	Descriptor Value
Relationship Type	Interface-InheritsFrom-Interface
Source Is Origin	No
Minimum Collection Size	Zero
Maximum Collection Size	Many
Sequenced Collection	No
Deletes Propagated	No
Destinations Named	No
Case Sensitive Names	Not applicable
Unique Names	Not applicable

IInterfaceMember Interface

See Also Properties Methods Collections

The properties, methods, and collections that a class implements are organized into functionally related groups. Each group is implemented as a COM interface. The properties, methods, and collections of each interface are *members* of the interface.

The IInterfaceMember interface maintains this information for an interface member:

- The member dispatch identifier.
- Information about member visibility.
- The relationship to the interface that exposes a particular interface member.

This information is common to properties, methods, and collection types. The **PropertyDef**, **MethodDef**, and **CollectionDef** classes all implement this interface.

When to Use

Use the IInterfaceMember interface to access the common properties of an interface member, or to determine which interface definition has a member of a particular property, method, or collection type.

Properties

Property	Description
DispatchID	The dispatch identifier to use when accessing an instance of this type of member.
Flags	Flags that specify details about this type of member.

Methods

IUnknown Method	Description
QueryInterface	Returns pointers to supported interfaces.
AddRef	Increments the reference count.
Release	Decrements the reference count.

IDispatch Method	Description
GetIDsOfNames	Maps a single member and a set of argument names to a corresponding set of dispatch identifiers.
GetTypeInfo	Retrieves a type information object, which can be used to get the type information for an interface.
GetTypeInfoCount	Retrieves the number of type information interfaces that an object provides (either 0 or 1).
Invoke	Provides access to properties and methods exposed by an Automation object.

IRepositoryDispatch Method	Description
get_Properties	Retrieves the IReposProperties

	interface pointer. The IReposProperties interface provides access to the Properties collection.

Collections

Collection	Description
Interface	The collection of one interface that exposes this type of member.

IInterfaceMember DispatchID Property

See Also

This property contains the dispatch identifier to use when accessing an instance of this type of member.

Dispatch Identifier: DISPID_DispID (51)

Property Data Type: long

IInterfaceMember Flags Property

See Also

This property contains a flag that specifies whether or not the interface member should be visible to Automation queries. See the InterfaceMemberFlags Enumeration for a list of values and their specific purposes.

Dispatch Identifier: DISPID_IfaceMemFlags (52)

Property Data Type: long

IInterfaceMember Interface Collection

See Also

For a particular property, method, or collection definition, the Interface collection specifies which interface exposes a member of this type.

Dispatch Identifier: DISPID_Iface (37)

Collection Descriptor	Descriptor Value
Relationship Type	Interface-Has-Members
Source Is Origin	No
Minimum Collection Size	One
Maximum Collection Size	One
Sequenced Collection	Yes
Deletes Propagated	Yes
Destinations Named	Yes
Case Sensitive Names	No
Unique Names	Yes

IManageReposTypeLib Interface

See Also Properties Methods Collections

Each tool information model that is stored in the repository is represented by a repository type library.

When to Use

Use the IManageReposTypeLib interface to:

- Create a repository type library for a new tool information model.
- Determine what tool information models are currently stored in the repository.

Methods

IUnknown Method	Description
QueryInterface	Returns pointers to supported interfaces.
AddRef	Increments the reference count.
Release	Decrements the reference count.

IDispatch Method	Description
GetIDsOfNames	Maps a single member and a set of argument names to a corresponding set of dispatch identifiers.
GetTypeInfo	Retrieves a type information object, which can be used to get the type information for an interface.
GetTypeInfoCount	Retrieves the number of type information interfaces that an object provides (either 0 or 1).
Invoke	Provides access to properties and methods exposed by an Automation object.

IRepositoryDispatch Method	Description
get_Properties	Retrieves the IReposProperties interface pointer. The IReposProperties interface provides access to the Properties collection.

IManageReposTypeLib Method	Description
CreateTypeLib	Creates a repository type library for a new tool information model.

Collections

Collection	Description
ReposTypeLibs	The collection of repository type libraries that are currently stored in the repository.

IManageReposTypeLib::CreateTypeLib

See Also

This method creates a new repository type library and attaches it to the root of the repository. Each repository type library represents a tool information model.

```
HRESULT CreateTypeLib(
    VARIANT          sObjId,
    BSTR               Name,
    VARIANT          TypeLibId,
    IReposTypeLib    **ppIRepTypeLib
);
```

Parameters

sObjId [in]
The object identifier to be used for the new repository type library object. The repository engine will assign an object identifier if you set this parameter to OBJID_NULL.

Name [in]
The name of the new repository type library.

TypeLibId [in]
The global identifier by which this repository type library is referenced.

**ppIRepTypeLib* [out]
The IReposTypeLib interface pointer to the new repository type library object.

Return Value

S_OK
The method completed successfully.

Error Values
This method failed to complete successfully.

IManageReposTypeLib ReposTypeLibs Collection

See Also

The collection of repository type libraries that are currently stored in the repository. Each repository type library represents a tool information model.

Dispatch Identifier: DISPID_ReposTypeLibs (40)

Collection Descriptor	Descriptor Value
Relationship Type	TlbManager-ContextFor-ReposTypeLibs
Source Is Origin	Yes
Minimum Collection Size	Zero
Maximum Collection Size	Many
Sequenced Collection	No
Deletes Propagated	Yes
Destinations Named	Yes
Case Sensitive Names	No
Unique Names	Yes

IPropertyDef Interface

See Also Properties Methods Collections

A property definition object specifies the characteristics of a particular type of property. These characteristics are defined by the properties *of the property definition object*.

To create a new property definition:

1. Use the **CreatePropertyDef** method of the **IInterfaceDef** interface.
2. Define any non-default characteristics of your new property definition by manipulating the properties of the property definition object.
3. Commit your changes to the repository database.

When to Use

Use the IPropertyDef interface to retrieve or modify the characteristics of a property definition.

Properties

Property	Description
APIType	The C data type of the property.
ColumnName	The name of the column in the SQL table for this property.
Flags	Specifies details about the property.
SQLScale	The number of digits to the right of the decimal point for a numeric property.
SQLSize	The size in bytes of the property.
SQLType	The SQL data type of the property.

Methods

IUnknown Method	Description
QueryInterface	Returns pointers to supported interfaces.
AddRef	Increments the reference count.
Release	Decrements the reference count.

IDispatch Method	Description
GetIDsOfNames	Maps a single member and a set of argument names to a corresponding set of dispatch identifiers.
GetTypeInfo	Retrieves a type information object, which can be used to get the type information for an interface.
GetTypeInfoCount	Retrieves the number of type information interfaces that an object provides (either 0 or 1).
Invoke	Provides access to properties and methods exposed by an Automation object.

IRepositoryDispatch Method	Description

get_Properties	Retrieves the IReposProperties interface pointer. The IReposProperties interface provides access to the Properties collection.

IPropertyDef APIType Property

See Also

The C data type of the property. For a definition of valid values, see the *ODBC Programmer's Reference*.

Dispatch Identifier: DISPID_APIType (59)

Property Data Type: short

IPropertyDef ColumnName Property

See Also

An SQL table is used to store instance information for the properties of an interface. By default, there is a column in this table for each property that is defined as a member of the interface. The **ColumnName** property specifies the name of the column in the SQL table for the property definition. The length of the column name must be 30 bytes or less.

Dispatch Identifier: DISPID_ColumnName (58)

Property Data Type: string

IPropertyDef Flags Property

See Also

A flag that specifies whether or not a column is created in the SQL table for the interface to which this property is attached. If no column is created, then instances of this property are only attached to individual objects, when the property value is set for that particular object. By default, a column *is* created for each property.

See the PropertyDefFlags Enumeration for the symbolic and numeric values of this flag.

Dispatch Identifier: DISPID_PropFlags (63)

Property Data Type: long

IPropertyDef SQLScale Property

See Also

The number of digits to the right of the decimal point for a numeric property. This parameter is ignored unless the **SQLType** property specifies an SQL_NUMERIC, SQL_DECIMAL, or SQL_TIME data type.

Dispatch Identifier: DISPID_SQLScale (62)

Property Data Type: short

IPropertyDef SQLSize Property

See Also

The size in bytes of the property. This parameter is ignored when the data type of the property inherently specifies the size of the property.

Dispatch Identifier: DISPID_SQL_Size (61)

Property Data Type: short

IPropertyDef SQLType Property

See Also

The SQL data type of the property. For a definition of valid values, see the *ODBC Programmer's Reference*.

Dispatch Identifier: DISPID_SQLType (60)

Property Data Type: short

IReposTypeInfo Interface

See Also Properties Methods Collections

This interface relates class, interface, and relationship definition objects to repository type libraries.

When to Use

Use the IReposTypeInfo interface to:

- Determine which repository type libraries contain a particular class, interface, or relationship type.
- Determine what collection types are associated with a particular relationship type.

Methods

IUnknown Method	Description
QueryInterface	Returns pointers to supported interfaces.
AddRef	Increments the reference count.
Release	Decrements the reference count.

IDispatch Method	Description
GetIDsOfNames	Maps a single member and a set of argument names to a corresponding set of dispatch identifiers.
GetTypeInfo	Retrieves a type information object, which can be used to get the type information for an interface.
GetTypeInfoCount	Retrieves the number of type information interfaces that an object provides (either 0 or 1).
Invoke	Provides access to properties and methods exposed by an Automation object.

IRepositoryDispatch Method	Description
get_Properties	Retrieves the IReposProperties interface pointer. The IReposProperties interface provides access to the Properties collection.

Collections

Collection	Description
ItemInCollections	The origin and destination collection types that are connected to a relationship definition object.
ReposTypeLibScopes	The collection of repository type libraries that contain a particular class, interface, or relationship type.

IReposTypeInfo ItemInCollections Collection

See Also

This collection contains the origin and destination collection types that are associated with a particular relationship type. This collection is empty for definition objects that are not relationship definitions.

Dispatch Identifier: DISPID_Collection (39)

Collection Descriptor	Descriptor Value
Relationship Type	Collection-Contains-Items
Source Is Origin	No
Minimum Collection Size	Zero
Maximum Collection Size	Two
Sequenced Collection	No
Deletes Propagated	No
Destinations Named	No
Case Sensitive Names	Not Applicable
Unique Names	Not Applicable

IReposTypeInfo ReposTypeLibScopes Collection

See Also

The collection of repository type libraries that contain a particular class, interface, or relationship type.

Dispatch Identifier: DISPID_ReposTypeLibScopes (43)

Collection Descriptor	Descriptor Value
Relationship Type	ReposTypeLib-ScopeFor-ReposTypeInfo
Source Is Origin	No
Minimum Collection Size	One
Maximum Collection Size	Many
Sequenced Collection	No
Deletes Propagated	Yes
Destinations Named	Yes
Case Sensitive Names	No
Unique Names	Yes

IReposTypeLib Interface

See Also Properties Methods Collections

There is one repository type library for every tool information model contained in the repository. Each tool information model provides a logical grouping of all of the type definitions related to a particular tool (or tool set).

To add a new repository type library to the repository, use the **IManageReposTypeLib** interface.

When to Use

Use the IReposTypeLib interface to:

- Define new classes, relationship types, and interfaces for a tool information model.
- Retrieve or modify the global identifier associated with a repository type library.
- Determine which type definitions are associated with a particular repository type library.

Properties

Property	Description
TypeLibID	The global identifier for the repository type library.

Methods

IUnknown Method	Description
QueryInterface	Returns pointers to supported interfaces.
AddRef	Increments the reference count.
Release	Decrements the reference count.

IDispatch Method	Description
GetIDsOfNames	Maps a single member and a set of argument names to a corresponding set of dispatch identifiers.
GetTypeInfo	Retrieves a type information object, which can be used to get the type information for an interface.
GetTypeInfoCount	Retrieves the number of type information interfaces that an object provides (either 0 or 1).
Invoke	Provides access to properties and methods exposed by an Automation object.

IRepositoryDispatch Method	Description
get_Properties	Retrieves the IReposProperties interface pointer. The IReposProperties interface provides access to the Properties collection.

IReposTypeLib Method	Description

CreateClassDef	Creates a new class definition object.
CreateInterfaceDef	Creates a new interface definition object.
CreateRelationshipDef	Creates a new relationship definition object.

Collections

Collection	Description
ReposTypeInfos	The collection of all classes, interfaces, and relationship types that are defined in the repository type library.
ReposTypeLibContexts	The collection of one repository root object that is the context for the repository type library.

IReposTypeLib TypeLibID Property

See Also

This property is the global identifier for the repository type library.

Dispatch Identifier: DISPID_TypeLibID (64)

Property Data Type: GUID

IReposTypeLib::CreateClassDef

See Also

This method creates a new class definition object. No interfaces are attached to the class.

```
HRESULT CreateClassDef(
    VARIANT    sObjId,
    BSTR       Name,
    VARIANT    sClsId,
    IClassDef  **ppIClassDef
);
```

Parameters

sObjId [in]
The object identifier to be used for the new class definition object. The repository engine will assign an object identifier if you set this parameter to OBJID_NULL.

Name [in]
The name of the new class.

sClsId [in]
The global identifier by which this class is referenced.

**ppIClassDef* [out]
The interface pointer to the new class definition object.

Return Value

S_OK
The method completed successfully.

Error Values
This method failed to complete successfully.

IReposTypeLib::CreateInterfaceDef

See Also

The CreateInterfaceDef method creates a new interface definition object. Use the **IClassDef::AddInterface** method to attach the interface to a class definition object.

```
HRESULT CreateInterfaceDef(
    VARIANT         sObjId,
    BSTR              Name,
    VARIANT         sIId,
    IInterfaceDef   *pIAncestor,
    IInterfaceDef   **ppIInterfaceDef
);
```

Parameters

sObjId [in]
The object identifier to be assigned to the new interface definition object. If this parameter is set to OBJID_NULL, the repository assigns an object identifier for you.

Name [in]
The name of the interface that is to be created.

sIId [in]
The interface identifier associated with the signature for this interface. If there is none, set this parameter to zero.

**pIAncestor* [in]
The IInterfaceDef interface pointer for the base interface from which the new interface is derived.

**ppIInterfaceDef* [out]
The interface pointer for the new interface.

Return Value

S_OK
The method completed successfully.

Error Values
This method failed to complete successfully.

IReposTypeLib::CreateRelationshipDef

See Also

This method creates a relationship definition object for a new relationship type. Once the relationship definition is created, use the **IInterfaceDef::CreateRelationshipColDef** method to create origin and destination collection definitions for the new relationship type.

```
HRESULT CreateRelationshipDef(
    VARIANT          sObjId,
    BSTR             Name,
    IReposTypeInfo   **ppIRelshipDef
);
```

Parameters

sObjId [in]
The object identifier for the new relationship type. The repository engine will assign an object identifier if you set this parameter to OBJID_NULL.

Name [in]
The name of the new relationship type.

**ppIRelshipDef* [out]
The COM interface pointer to the new relationship definition object.

Return Value

S_OK
The method completed successfully.

Error Values
This method failed to complete successfully.

IReposTypeLib ReposTypeLibContexts Collection

See Also

The collection of one repository root that is the context for a repository type library.

Dispatch Identifier: DISPID_ReposTLBContexts (41)

Collection Descriptor	Descriptor Value
Relationship Type	TlbManager-ContextFor-ReposTypeLibs
Source Is Origin	No
Minimum Collection Size	One
Maximum Collection Size	Many
Sequenced Collection	No
Deletes Propagated	Yes
Destinations Named	Yes
Case Sensitive Names	No
Unique Names	Yes

IReposTypeLib ReposTypeInfos Collection

See Also

The collection of all classes, interfaces, and relationship types that are associated with a repository type library. The repository engine uses this collection to enforce the unique naming of all classes, interfaces, and relationship types for a repository type library.

Dispatch Identifier: DISPID_ReposTypeInfos (42)

Collection Descriptor	Descriptor Value
Relationship Type	ReposTypeLib-ScopeFor-ReposTypeInfo
Source Is Origin	Yes
Minimum Collection Size	Zero
Maximum Collection Size	Many
Sequenced Collection	No
Deletes Propagated	Yes
Destinations Named	Yes
Case Sensitive Names	No
Unique Names	Yes

IReposRoot Interface

See Also Properties Methods Collections

The **IReposRoot** interface is a placeholder interface; it contains no properties, methods, or collections beyond Automation dispatch methods. It is provided as a convenient connection point to the root object. When you create a tool information model, you can attach a relationship collection to this interface that provides a navigational connection to the primary objects of your tool information model.

When to Use

Use the **IReposRoot** interface as a starting point to navigate to other objects in the repository.

Methods

IUnknown Method	Description
QueryInterface	Returns pointers to supported interfaces.
AddRef	Increments the reference count.
Release	Decrements the reference count.

IDispatch Method	Description
GetIDsOfNames	Maps a single member and a set of argument names to a corresponding set of dispatch identifiers.
GetTypeInfo	Retrieves a type information object, which can be used to get the type information for an interface.
GetTypeInfoCount	Retrieves the number of type information interfaces that an object provides (either 0 or 1).
Invoke	Provides access to properties and methods exposed by an Automation object.

IRepositoryDispatch Method	Description
get_Properties	Retrieves the IReposProperties interface pointer. The IReposProperties interface provides access to the Properties collection.

Relationship Type Descriptor

The type of relationship by which all items of the collection are connected to a common source object.

Source Is Origin Descriptor

Specifies whether or not the source object for the collection is also the origin object. Value can be set to *Yes* or *No*.

Maximum Collection Size Descriptor

The maximum number of items that can be contained in the collection. Value can be set to *Zero*, *One*, a specific numeric value, or *Many*. The repository engine does not enforce this maximum.

Minimum Collection Size Descriptor

The minimum number of items that must be contained in the collection. Value can be set to *Zero*, *One*, a specific numeric value, or *Many*. The repository engine does not enforce this minimum.

Deletes Propagated Descriptor

Specifies whether or not the deletion of an origin object or deletion of a relationship in the collection will cause the deletion of the corresponding destination object. Value can be set to *Yes* or *No*.

Sequenced Collection Descriptor

Specifies whether or not the items in the destination collection have an explicitly defined sequence. Collections of origin objects are never sequenced. Value can be set to *Yes* or *No*.

Destinations Named Descriptor

Specifies whether or not the relationship type for the collection permits the naming of destination objects. Value can be set to *Yes* or *No*.

Case Sensitive Names Descriptor

Specifies whether or not the relationship type for the collection permits the use of case sensitive names for destination objects. This descriptor is meaningful only for collections whose relationship type permits destination objects to be named. Value can be set to *Yes*, *No*, or *Not Applicable*.

Unique Names Descriptor

Specifies whether or not the relationship type for the collection requires that the name of a destination object be unique within the collection of destination objects. This descriptor is meaningful only for collections whose relationship type permits destination objects to be named. Value can be set to *Yes*, *No*, or *Not Applicable*.

Repository Data Structures and Constants

The various declarations and definitions for Microsoft Repository can be found in these files:

- The REPAPI.H source file contains C++ definitions specific to the repository engine.
- The REPTIM.H source file contains the constant definitions specific to the Type Information Model. Most of the various identifiers (class, interface, object, local, internal, and dispatch) that you may find useful are defined in this file.
- The REPAUTO.H file contains the definitions of the external enumerations, classes, and interfaces of the repository engine and the Type Information Model. All of the interfaces found in this file support Automation level access.

Repository Constants

See Also

These constants are defined for the Microsoft Repository application programming interface.

Constant	Value	Description
CARD_NOLIMIT	0xFFFF	The maximum count value for a collection with an unlimited number of items.
COLUMNNAMESIZE	32	Maximum length, in bytes, of an SQL column name.
INTID_NULL	0xFFFFFFFF	The null internal identifier.
MEMBERNAMESIZE	64	Maximum length, in bytes, of a property, method, or collection type name.
OBJID_NULL	See reptim.h	The null object identifier. Use this value when you want the repository to assign an object identifier for you.
PASSWORDSIZE	64	Maximum length, in bytes, of the password string that is used to connect to the repository database.
PROPVALSIZE	255	Maximum length, in bytes, of an annotational property string.
RELSHIPNAMESIZE	260	Maximum length, in bytes, of a name that a relationship assigns to its destination object.
REPOSERROR_OBJKNOWN	0x00000001L	Returned in the *fFlags* field of the REPOSERROR structure. Indicates that the object identifier is known.
REPOSERROR_SQLINFO	0x00000002L	Returned in the *fFlags* field of the REPOSERROR structure. Indicates that the SQL error information is valid.
REPOSERROR_HELPAVAIL	0x00000004L	Returned in the *fFlags* field of the REPOSERROR structure. Indicates that the *rcHelpFile* and *dwHelpContext* fields are valid.
REPOSERROR_MSG_SIZE	256	Maximum length, in bytes, of the message in the *rcMsg* field of the REPOSERROR structure.
TABLENAMESIZE	32	Maximum length, in bytes, of an SQL table name.
TYPEINFONAMESIZE	64	Maximum length, in bytes, of a class, interface, or relationship type name.
TYPELIBNAMESIZE	64	Maximum length, in bytes, of a repository type library name.
USERSIZE	64	Maximum length, in bytes, of the user name that is used to connect to the repository database.

CollectionDefFlags Enumeration

See Also

These values are used to define the behavior of a relationship collection. These flags are bit flags, and may be combined to set multiple options. The absence of a flag signifies that the option is not set.

```
enum { COLLECTION_NAMING          = 1,
       COLLECTION_UNIQUENAMING    = 2,
       COLLECTION_CASESENSITIVE   = 4,
       COLLECTION_SEQUENCED       = 8
       COLLECTION_PROPAGATEDELETE = 16
} CollectionDefFlags;
```

Value	Description
COLLECTION_CASESENSITIVE	Specifies that the relationship type for the collection permits the use of case sensitive names for destination objects. This descriptor is meaningful only for collections whose relationship type permits destination objects to be named.
COLLECTION_NAMING	Specifies that the relationship type for the collection permits the naming of destination objects.
COLLECTION_SEQUENCED	Specifies that the items in the collection have an explicitly defined sequence. Collections of origin objects are never sequenced.
COLLECTION_UNIQUENAMING	Specifies that the relationship type for the collection requires that the name of a destination object be unique within the collection of destination objects. This descriptor is meaningful only for collections whose relationship type permits destination objects to be named.
COLLECTION_PROPAGATEDELETE	Specifies that the relationship type for the collection requires that deletes be propagated to destination objects. The destination object is only deleted if this is the last relationship *of this type* that is connected to the object.

ConnectionFlags Enumeration

See Also

These values are used to define the characteristics of a connection to a repository database. These flags are bit flags, and may be combined to set multiple options. The absence of a flag signifies that the option is not set.

```
enum { REPOS_CONN_EXCLUSIVE  = 1,
       REPOS_CONN_NEWCACHE   = 2
} ConnectionFlags;
```

Value	Description
REPOS_CONN_EXCLUSIVE	Open the repository database in exclusive mode. No other users will be allowed access to the database.
REPOS_CONN_NEWCACHE	Create a new cache for this open repository instance. This will consume additional resources.

InterfaceDefFlags Enumeration

See Also

These values are used to define specific characteristics of an interface definition. These flags are bit flags, and may be combined to set multiple options. The absence of a flag signifies that the option is not set.

```
enum { INTERFACE_EXTENSIBLE  = 1,
       INTERFACE_HIDDEN      = 2
} InterfaceDefFlags;
```

Value	Description
INTERFACE_EXTENSIBLE	Specifies that the interface will support extensions.
INTERFACE_HIDDEN	Specifies that the interface is not to be visible to Automation queries.

InterfaceMemberFlags Enumeration

See Also

This enumeration is used to define specific characteristics of an interface member. The absence of the flag signifies that the option is not set.

```
enum { INTERFACEMEMBER_HIDDEN  = 1
} InterfaceMemberFlags;
```

Value	Description
INTERFACEMEMBER_HIDDEN	Specifies that the interface member is not to be visible to Automation queries.

PropertyDefFlags Enumeration

See Also

This enumeration is used to define specific characteristics of a property definition. The absence of the flag signifies that the option is not set.

```
enum { PROPERTY_INVERTED  = 1
} PropertyDefFlags;
```

Value	Description
PROPERTY_INVERTED	Specifies that the property is to be stored in inverted form; that is, a column is *not* reserved in the SQL table that contains the properties for the implementing interface. Instances of this property will only exist for objects that have a value set for the property. The property is restricted to the string data type.

TransactionFlags Enumeration

See Also

This enumeration is used to specify which transaction option is to be retrieved or set.

```
enum { TXN_RESET_OPTIONS           = 1,
       TXN_NORMAL                  = 2,
       TXN_EXCLUSIVE_WRITEBACK     = 3,
       TXN_EXCLUSIVE_WRITETHROUGH  = 4,
       TXN_TIMEOUT_DURATION        = 5,
       TXN_START_TIMEOUT           = 6
} TransactionFlags;
```

Value	Description
TXN_RESET_OPTIONS	Valid only for setting transaction options. Specifies that all options are to be reset to their default values. Any associated option value is ignored.
TXN_NORMAL	Specifies the non-exclusive write-back mode transaction option.
TXN_EXCLUSIVE_WRITEBACK	Specifies the exclusive write-back mode transaction option.
TXN_EXCLUSIVE_WRITETHROUGH	Specifies the exclusive write-through mode transaction option.
TXN_TIMEOUT_DURATION	Specifies the transaction option that determines the maximum time to wait for a lock.
TXN_START_TIMEOUT	Specifies the transaction option that determines the maximum time to wait before starting a transaction.

Repository Data Types

See Also

The following structures and data types are defined for the Microsoft Repository application programming interface.

Internal Identifier

```
struct INTID
{
   ULONG  iSiteID;
   ULONG  iLocalID;
};

typedef const INTID &REFINTID;
```

An **INTID** or a **REFINTID** variable is an internal identifier for a specific repository object that uniquely identifies the object within a particular repository database. It is *not* unique across all repositories. This is not the same thing as the interface identifier for an interface, or the class identifier that is used to create an instance of a class.

The internal identifier is composed of an internal site identifier (iSiteID), and an internal local identifier (iLocalID).

Object Identifier

```
typedef const OBJECTID OBJID;
typedef const OBJID&   REFOBJID;
```

An **OBJID** or a **REFOBJID** variable is an object identifier for a specific repository object in a particular repository database. An object identifier is unique across all repositories. An object identifier is not the same thing as the interface identifier for an interface, or the class identifier that is used to create an instance of a class.

An object identifier is composed of a global identifier (GUID) and a four byte local identifier appended to the GUID. The GUID portion of the object identifier specifies where the object identifier was created, and the local identifier has a value that is unique within the repository database.

REPOSERROR Data Structure

See Also

Repository methods return an **HRESULT** value that indicates whether or not the method completed successfully. If a repository method fails to complete successfully, an error object is created that contains details about the failure. The **REPOSERR** data structure contains these details.

```
struct REPOSERROR
{
   ULONG    iSize;
   ULONG    fFlags;
   HRESULT  hr;
   TCHAR    rcMsg[REPOSERROR_MSG_SIZE];
   TCHAR    rcHelpFile[_MAX_PATH];
   ULONG    dwHelpContext;
   long     iNativeError;
   TCHAR    rcSqlState[6];
   short    iReserved;
   OBJID    sObjID;
   GUID     clsid;
   GUID     iid;
};
```

iSize
: The size in bytes of this data structure.

fFlags
: Bit flags that define the validity of certain members of this data structure. Valid values are **REPOSERROR_OBJKNOWN**, **REPOSERROR_SQLINFO**, and **REPOSERROR_HELPAVAIL**. See Repository Constants for the meaning of these constants.

hr
: The **HRESULT** return value that was returned from the method that logged this error.

rcMsg
: The text message that is associated with this error.

rcHelpFile
: The name of the help file that contains more information about this error.

dwHelpContext
: The help context identifier that is associated with this error.

iNativeError
: The error code that was returned from the database engine. The value of this member is only valid if the *fFlags* member indicates that SQL information is present.

rcSqlState
: SQL state information supplied by the database engine. The value of this member is only valid if the *fFlags* member indicates that SQL information is present.

iReserved
: Reserved for use by the repository engine.

sObjID
: The object identifier of the object that is associated with this error. The value of this member is only valid if the *fFlags* member indicates that the object is known.

clsid
: The class identifier of the object that is associated with this error. The value of this member is only valid if the *fFlags* member indicates that the object is known.

iid
: The interface identifier of the interface that is associated with this error. If the interface is not

known, or not applicable, the value of this member is set to GUID_NULL.

Repository Errors

See Also

Nearly all repository methods return an **HRESULT** value that indicates whether or not the method succeeded in performing its function. The facility field of these HRESULT values is always set to FACILITY_ITF, which indicates that the meaning for any given error code value is specific to the interface from which the error is being reported. All of the standard repository interfaces (that is, interfaces that are automatically supplied with the repository) use the same set of error codes. These codes are listed in numerical order, and in alphabetical order.

Repository Errors (Numerical Order)

See Also

The error codes that can be returned as a part of the **HRESULT** return value by repository methods are listed in numerical order below. These codes are also listed in alphabetical order.

(0x1000) EREP_BADPARAMS

(0x1001) EREP_BADNAME

(0x1002) EREP_BADDRIVER

(0x1011) EREP_NOROWSFOUND

(0x1012) EREP_ODBC_CERROR

(0x1013) EREP_ODBC_MDBNOTFOUND

(0x1015) EREP_ODBC_UNKNOWNDRIVER

(0x1030) EREP_DB_EXISTS

(0x1031) EREP_DB_NOTCONNECTED

(0x1032) EREP_DB_ALREADYCONNECTED

(0x1033) EREP_DB_DBMSONETHREAD

(0x1034) EREP_DB_CORRUPT

(0x1035) EREP_DB_NOSCHEMA

(0x1041) EREP_TXN_NOTXNACTIVE

(0x1042) EREP_TXN_AUTOABORT

(0x1043) EREP_TXN_TOOMANY

(0x1044) EREP_TXN_TIMEOUT

(0x1045) EREP_TXN_NODATA

(0x1046) EREP_TXN_NOSETINTXN

(0x1070) EREP_REPOS_CACHEFULL

(0x1071) EREP_REPOS_NONEXTDISPID

(0x1100) EREP_RELSHIP_EXISTS

(0x1101) EREP_RELSHIP_INVALID_PAIR

(0x1102) EREP_RELSHIP_NOTFOUND

(0x1105) EREP_RELSHIP_ORGONLY

(0x1106) EREP_RELSHIP_OUTOFDATE

(0x1107) EREP_RELSHIP_INVALIDFLAGS

(0x1108) EREP_RELSHIP_NAMEINVALID

(0x1109) EREP_RELSHIP_DUPENAME

(0x1120) EREP_TYPE_TABLEMISMATCH

(0x1121) EREP_TYPE_COLMISMATCH

(0x1122) EREP_TYPE_NOTNULLABLE

(0x1123) EREP_TYPE_MULTIDEFIFACES

(0x1124) EREP_TYPE_INVERTEDNOTALLOWED

(0x1125) EREP_TYPE_INVALIDSCALE

(0x1200) EREP_LOCK_TIMEOUT

(0x1300) EREP_OBJ_NOTINITIALIZED

(0x1301) EREP_OBJ_NOTFOUND

(0x1302) EREP_OBJ_NONAMINGRELSHIP

(0x1303) EREP_OBJ_EXISTS

(0x1400) EREP_PROP_MISMATCH

(0x1401) EREP_PROP_SETINVALID

(0x1402) SREP_PROP_TRUNCATION

(0x1403) EREP_PROP_CANTSETREPTIM

(0x1404) EREP_PROP_READONLY

Repository Errors (Alphabetical Order)

See Also

The error codes that can be returned as a part of the **HRESULT** return value by repository methods are listed below in alphabetical order, according to the symbolic name for each error code. These codes are listed in numerical order.

EREP_BADDRIVER (0x1002)

EREP_BADNAME (0x1001)

EREP_BADPARAMS (0x1000)

EREP_DB_ALREADYCONNECTED (0x1032)

EREP_DB_CORRUPT (0x1034)

EREP_DB_DBMSONETHREAD (0x1033)

EREP_DB_EXISTS (0x1030)

EREP_DB_NOSCHEMA (0x1035)

EREP_DB_NOTCONNECTED (0x1031)

EREP_LOCK_TIMEOUT (0x1200)

EREP_NOROWSFOUND (0x1011)

EREP_OBJ_EXISTS (0x1303)

EREP_OBJ_NONAMINGRELSHIP (0x1302)

EREP_OBJ_NOTFOUND (0x1301)

EREP_OBJ_NOTINITIALIZED (0x1300)

EREP_ODBC_CERROR (0x1012)

EREP_ODBC_MDBNOTFOUND (0x1013)

EREP_ODBC_UNKNOWNDRIVER (0x1015)

EREP_PROP_CANTSETREPTIM (0x1403)

EREP_PROP_MISMATCH (0x1400)

EREP_PROP_READONLY (0x1404)

EREP_PROP_SETINVALID (0x1401)

EREP_RELSHIP_DUPENAME (0x1109)

EREP_RELSHIP_EXISTS (0x1100)

EREP_RELSHIP_INVALIDFLAGS (0x1107)

EREP_RELSHIP_INVALID_PAIR (0x1101)

EREP_RELSHIP_NAMEINVALID (0x1108)

EREP_RELSHIP_NOTFOUND (0x1102)

EREP_RELSHIP_ORGONLY (0x1105)

EREP_RELSHIP_OUTOFDATE (0x1106)

EREP_REPOS_CACHEFULL (0x1070)

EREP_REPOS_NONEXTDISPID (0x1071)

EREP_TXN_AUTOABORT (0x1042)

EREP_TXN_NODATA (0x1045)

EREP_TXN_NOSETINTXN (0x1046)

EREP_TXN_NOTXNACTIVE (0x1041)

EREP_TXN_TIMEOUT (0x1044)

EREP_TXN_TOOMANY (0x1043)

EREP_TYPE_COLMISMATCH (0x1121)

EREP_TYPE_INVALIDSCALE (0x1125)

EREP_TYPE_INVERTEDNOTALLOWED (0x1124)

EREP_TYPE_MULTIDEFIFACES (0x1123)

EREP_TYPE_NOTNULLABLE (0x1122)

EREP_TYPE_TABLEMISMATCH (0x1120)

SREP_PROP_TRUNCATION (0x1402)

EREP_BADPARAMS (0x1000)

One or more invalid parameters have been passed to a repository method. Correct the input parameters and try again.

EREP_BADNAME (0x1001)

The name that you have supplied for a table or column name either contains invalid characters, or is a reserved word for the database management system. Change the name and try your request again.

EREP_BADDRIVER (0x1002)

The currently installed ODBC driver is too old, and is incompatible with Microsoft Repository. Update your ODBC driver.

EREP_NOROWSFOUND (0x1011)

A query operation against the repository database yielded no rows. If you expected data to be returned, verify that your query is properly constructed.

EREP_ODBC_CERROR (0x1012)

An database error has occurred; check the error queue for more information. Determine the source of the problem, and correct it, then try again.

EREP_ODBC_MDBNOTFOUND (0x1013)

You have specified a repository database that does not exist or is not accessible. Make sure the database exists, that the name is correct, and try again.

EREP_ODBC_UNKNOWNDRIVER (0x1015)

The specified ODBC driver is not a valid driver, or is not known to the repository engine. Obtain an ODBC driver that is compatible with Microsoft Repository. See the product release notes for information about compatible drivers.

EREP_DB_EXISTS (0x1030)

You have requested that a repository database be created with a name that is the name of an already existing database. If you can use the existing database, then use the **Open** method instead of the **Create** method. If the existing database is no longer needed, delete it. Otherwise, choose a different name, and try again.

EREP_DB_NOTCONNECTED (0x1031)

You have requested an operation that requires a connection to an open repository database, and you do not currently have such a connection. **Open** the appropriate repository, and try your request again.

EREP_DB_ALREADYCONNECTED (0x1032)

You have attempted to connect to a repository database when you are already connected. Skip the redundant **Open** method invocation and proceed with the repository interactions that follow that Open invocation.

EREP_DB_DBMSONETHREAD (0x1033)

The repository database that you have attempted to access is managed by a database server that does not support multi-threaded access. The thread attempting the access is not the same as the thread that currently has the open repository instance for the database. Either move your repository database to a database server that supports multi-threaded access, or modify the logic of your program to use a single thread for repository database access.

EREP_DB_CORRUPT (0x1034)

The repository database has been damaged. See your database server documentation to determine what facilities are available for restoring or rebuilding the database.

EREP_DB_NOSCHEMA (0x1035)

The repository database does not contain the Type Information Model schema. If your repository has not yet been populated with data, install the Type Information Model schema by using the **Create** method to open the repository database. If your repository has been populated with data, restore the database from a backup copy.

EREP_TXN_NOTXNACTIVE (0x1041)

You have attempted to update the repository database, but there is no transaction active. Bracket your repository updates between **Begin** and **Commit** transaction method invocations.

EREP_TXN_AUTOABORT (0x1042)

You have released the resources for an open repository instance, and that repository instance had a transaction active. The transaction has been canceled; all changes associated with the transaction will be undone. You should always complete an active transaction (via either the **Commit** or the **Abort** method) before releasing an open repository instance.

EREP_TXN_TOOMANY (0x1043)

A new transaction cannot be started because the maximum number of concurrent transactions has been exceeded. Reduce the number of concurrently executing transactions (within the same process).

EREP_TXN_TIMEOUT (0x1044)

Your transaction has timed out while waiting to begin. Either increase the start transaction time-out value and retry the transaction, or wait for the item to become available and *then* retry the transaction. For details on how to change the start transaction time-out value, see the **SetOption** method.

EREP_TXN_NODATA (0x1045)

You have attempted to retrieve the value of a property that is null, or does not exist. The corrective action that you must take is dependent upon the requirements of your task. Consider handling this exception, with special processing for the case where a property has a null value.

EREP_TXN_NOSETINTXN (0x1046)

You have attempted to modify the current transaction option settings while a transaction is active. Either complete the current transaction and then modify the transaction options, or set the transaction options prior to beginning the transaction.

EREP_REPOS_CACHEFULL (0x1070)

The repository cache is full. If you are writing new and changed data to the repository, and you cannot reduce the number of steps in the transaction, consider setting either the TXN_EXCLUSIVE_WRITEBACK or the TXN_EXCLUSIVE_WRITETHROUGH transaction option to avoid this problem.

EREP_REPOS_NONEXTDISPID (0x1071)

You have attempted to add a member to an interface that is defined in the repository, but there are no more dispatch identifier values available. Factor the interface into several smaller interfaces.

EREP_RELSHIP_EXISTS (0x1100)

You have attempted to create a relationship that already exists in the repository. Either ignore this error, or eliminate the redundant **Add** or **Insert** method invocation from your program.

EREP_RELSHIP_INVALID_PAIR (0x1101)

An attempt to add a new relationship between two objects has failed. One or both of the classes to which these objects conform does not support this type of relationship. Verify that the relationship type and the object classes are correct. Check your tool information model to verify that it supports the type of relationship that you are trying to create.

EREP_RELSHIP_NOTFOUND (0x1102)

You have attempted to retrieve a *specific* relationship that does not exist, or you have attempted to retrieve a relationship from an empty collection. If multiple users are accessing the repository concurrently, this error can occur if one user deletes a relationship while a second user is attempting to retrieve the relationship. Consider handling this exception, with special processing for the case where a collection is empty, or a specific relationship no longer exists.

EREP_RELSHIP_ORGONLY (0x1105)

An attempt to move or insert a relationship in a sequenced collection has failed because the **Move** or **Insert** method was invoked via the destination object instead of the origin object. Use the origin object to move or insert a relationship in a sequenced collection.

EREP_RELSHIP_OUTOFDATE (0x1106)

Your request has failed because the sequenced relationship collection that you are attempting to update has been changed by another process. Refresh the collection and try the update again.

EREP_RELSHIP_INVALIDFLAGS (0x1107)

Your attempt to add or modify a relationship collection has failed. Either the combination of flags are invalid, or you are attempting to set flag values on a destination collection. Verify that the origin collection is being used for the operation, and that the flag combinations are valid. See CollectionDefFlags for details on relationship collection flags.

EREP_RELSHIP_NAMEINVALID (0x1108)

You have attempted to add a relationship that has an invalid name specified for the destination object. Verify that the name is non-null, and is not longer than the maximum allowed length. For details about repository text string lengths, see Repository Constants.

EREP_RELSHIP_DUPENAME (0x1109)

You have attempted to add a relationship with a name that is not unique within the collection, and the collection requires unique names. Either choose a different name for the relationship, or delete the existing relationship with the same name, if it is no longer needed.

EREP_TYPE_TABLEMISMATCH (0x1120)

An attempt to extend an interface for a tool information model has failed. The SQL table that is designated as the table to be used for storing property values for the interface does not contain the expected columns. Check the table to determine if it has been damaged or if columns have been dropped from the table. Restore the table to its prior state and try the request again.

EREP_TYPE_COLMISMATCH (0x1121)

The conversion of a property value between the stored data type and the data type as specified by the caller has failed. Check the caller-specified data type to verify that it can be converted to the storage data type, as defined by the associated property definition object.

EREP_TYPE_NOTNULLABLE (0x1122)

You have attempted to set a property value to the null value, and the property definition does not allow this. Choose one of the permitted property values and try the update operation again.

EREP_TYPE_MULTIDEFIFACES (0x1123)

You have attempted to set more than one interface as the default interface for a class definition. Choose one of the interfaces to be the default interface.

EREP_TYPE_INVERTEDNOTALLOWED (0x1124)

You have attempted to add a property to an interface using the PROPERTY_INVERTED option, and the option is not permitted for the interface. Correct either the property definition or the interface definition.

EREP_TYPE_INVALIDSCALE (0x1125)

You have attempted to set the SQLScale property of a property definition to an invalid value. Correct the value that you are using and try the operation again.

EREP_LOCK_TIMEOUT (0x1200)

An attempt to obtain a lock on a repository item has timed out. Either increase the lock time-out value and try again, or wait for the item to become available and *then* try again. For details on how to change the lock time-out value, see the **SetOption** method.

EREP_OBJ_NOTINITIALIZED (0x1300)

An attempt has been made to interact with a repository object that has not been initialized with valid data from the repository database. Ensure that all repository objects in your program are initialized before attempting to interact with them.

EREP_OBJ_NOTFOUND (0x1301)

You have attempted to retrieve a repository object that does not exist. If multiple users are accessing the repository concurrently, this error can occur if one user deletes a repository object while a second user is attempting to retrieve the object. It can also occur if you are using an object identifier that has been saved from prior interactions with the repository, and the object has been deleted between the time that you obtained the object identifier and the time that you attempted to retrieve the repository object. Consider handling this exception with special processing for the case where a repository object no longer exists.

EREP_OBJ_NONAMINGRELSHIP (0x1302)

The retrieval of the name associated with a repository item has failed because the item is not a participant in a naming relationship. Check your tool information model to see if the relationship type is correctly defined. If it is, and the relationship type does not specify that destination objects are to be named, consider adding logic to your program to check for the presence of a naming relationship before requesting the repository item name.

EREP_OBJ_EXISTS (0x1303)

You have attempted to create a repository object that already exists in the repository. This situation can occur if multiple users are attempting to add the same object to the repository concurrently. If this is not the case, eliminate the redundant **Add**, **CreateObject**, or **Insert** method invocation from your program.

EREP_PROP_MISMATCH (0x1400)

An attempt to update a property value in the repository has failed. The data type of the input property cannot be converted to the storage data type. Correct the data type of the input property and try the update again.

EREP_PROP_SETINVALID (0x1401)

You have attempted to modify a collection as if it were a property. This type of operation is not supported by Microsoft Repository.

SREP_PROP_TRUNCATION (0x1402)

Your request to set the value of a property has succeeded; however, the value of the property has been truncated because the input property value was too long.

EREP_PROP_CANTSETREPTIM (0x1403)

You have attempted to modify a property of a definition object that is part of the Type Information Model. Modification of Type Information Model properties is not supported.

EREP_PROP_READONLY (0x1404)

Your request to set the value of a property has failed because the property is a read-only property.

Repository SQL Schema

See Also

The repository SQL schema consists of a standard schema and an extended schema. The standard schema consists of tables that contain the core information that is needed to manage all repository objects, relationships, and collections. The standard schema also contains tables that are used by the repository to store the definition information for tool information models.

SQL schema extensions are automatically generated by the repository engine when you create or extend a tool information model. Normally, one table is created for each interface that is defined in the repository. The table contains the instance data for persistent properties that are attached to the interface. One column in the table is created for each property. If an interface is defined that has no member properties, a table is not created.

Although the extended schema is automatically generated, you can tune the extended schema to optimize performance and data retrieval, if you are an experienced SQL administrator. For example, by default, the properties of each interface are stored in a separate SQL table. You can map the properties of *multiple* interfaces to a *single* table, or you can map the properties of a *single* interface to *multiple* tables. You can also specify the column names and data types to be used for property data. You can add indexes to tables, but you must not remove indexes that have been automatically defined by the repository.

You can construct a SQL query to extract specific information from the repository. You need to be familiar with the SQL schema to build such a query. The set of SQL tables that comprise the standard schema are shown below.

SQL Table Name	Description
RTblClassDefs	Contains information about classes that are defined in the repository.
RTblIfaceDefs	Contains information about interfaces that are defined in the repository.
RTblObjects	Contains information about repository objects.
RTblPropDefs	Contains information about property definitions.
RTblProps	Contains property values for annotational properties that are attached to repository objects.
RTblRelshipProps	Contains property values for annotational properties that are attached to relationships.
RTblRelships	Contains information about relationships.
RTblRelColDefs	Contains information about collection types.
RTblRelColPairs	Defines which types of objects can assume origin and destination roles for any given relationship type.
RTblSites	Translates local site identifiers to global site identifiers.
RTblTypeLibs	Contains information about repository type libraries.

Microsoft Repository can access repository databases that are managed by either Microsoft Jet or Microsoft SQL Server. Since the underlying database management system (DBMS) can vary, the utilities and tools that you use to administer the repository database (*at the database level)* also vary. For example, if your repository database is damaged due to a power outage or system failure, you should use the recovery tools that are provided with your DBMS to repair the damage. Similarly, if

your repository database requires periodic defragmentation, you should use the defragmentation tools that are provided with your DBMS.

Repository SQL Data Types

See Also

Because data types can vary between database management systems, the repository engine maps its own set of repository data types to the SQL data types that are supported by the underlying database server.

This table translates repository data types into SQL data types known by the database server. For data types that vary between different database servers, the data type used for each database server is shown. In these cases, the Microsoft SQL Server data type is shown with (S) appended to it, and the Jet server data type is shown with (J) appended to it.

Repository Data Type	Database Data Type	Description
RTBoolean	bit (S) boolean (J)	A true / false value.
RTCount	2 byte integer	The count, or cardinality of a collection.
RTDispID	4 byte integer	An Automation dispatch identifier.
RTFlags	2 byte integer	Flag bits that define the behavior of an entity.
RTGUID	16 byte binary	A globally unique identifier.
RTIntID	8 byte binary	An internal identifier.
RTLocalID	4 byte binary	A local identifier; part of an internal identifier.
RTLongBinary	image (S) longbinary (J)	A long binary stream of data.
RTLongString	text (S) memo (J)	A string with a maximum length of approximately 1 gigabyte.
RTNameString	200 byte varchar	A special truncated name string used for indexing.
RTScale	2 byte integer	Scale for numeric data; the number of digits after the decimal point.
RTShortString	240 byte varchar	A special shortened string value used for indexing.
RTSiteID	4 byte binary	A site identifier; part of an object identifier.
RTSize	2 byte integer	The size of a data type, in bytes.
RTSQLName	30 byte varchar	A SQL identifier; a table or column name.
RTSQLType	2 byte integer	The ODBC representation of a SQL data type.

RTblClassDefs SQL Table

See Also

This SQL table contains one row for each class that is defined in the repository. The IntID column is the primary key for this table. A unique index is defined on the Class ID column.

Column Name	Data Type	Description
IntID	RTIntID	The internal identifier for the class object.
ClassID	RTGUID	The global identifier of the class, as recorded in the system registry.
PropDescs	RTLongBinary	Definition information for the class. Reserved for proprietary usage by the repository engine. This field can be null.

RTblIfaceDefs SQL Table

See Also

This SQL table contains one row for each interface that is defined in the repository. The IntID column is the primary key for this table. A unique index is defined on the InterfaceID column.

Column Name	Data Type	Description
IntID	RTIntID	The internal identifier for the class object.
InterfaceID	RTGUID	The global identifier of the interface, as recorded in the system registry.
SQLTableName	RTSQLName	The name of the SQL table used to store property instance data for the interface. This field can be null.

RTblIfaceMem SQL Table

See Also

This SQL table contains one row for each property definition, method definition, and collection definition that is stored in the repository. The information contained in this table is used by the repository to create interface-specific SQL tables when an interface is added to the repository. The IntID column is the primary key for this table.

Column Name	Data Type	Description
IntID	RTIntID	The internal identifier for the property definition object.
DispID	RTDispID	The Automation dispatch identifier for the member. This field can be null.
Flags	RTFlags	Flags that determine member behavior. For a list of valid values and their meanings, see the InterfaceMemberFlags Enumeration.

RTblObjects SQL Table

See Also

This SQL table contains a row for every repository object in the repository. The IntID column is the primary key for this table.

Column Name	Data Type	Description
IntID	RTIntID	The internal identifier for the object.
TypeID	RTIntID	The internal identifier for the class definition object to which this object conforms.

RTblPropDefs SQL Table

See Also

This SQL table contains one row for each property definition object that is stored in the repository. The information contained in this table is used by the repository to create interface-specific SQL tables when an interface is added to the repository. The IntID column is the primary key for this table.

Column Name	Data Type	Description
IntID	RTIntID	The internal identifier for the property definition object.
SQLType	RTSQLType	The SQL data type for the property.
APIType	RTSQLType	The C language data type for the property. This is the type of the property when it is passed through a repository programming interface.
SQLSize	RTSize	The length in bytes of the property in terms of its SQL data type.
SQLColName	RTSQLName	The name of the column in the SQL table for this property. This field can be null.
SQLScale	RTScale	The scale for a numeric property; the number of digits after the decimal point. This field can be null.
Flags	RTFlags	Flags that determine property behavior. See the PropertyDefFlags Enumeration for a list of values and their specific purposes.

RTblProps SQL Table

See Also

An annotational property is a property that is associated with an individual repository object or relationship, and is not required to exist for all repository objects or relationships of a particular type. This SQL table contains one row for each annotational property instance that is attached to a *repository object*. The IntID and PropID columns comprise the primary key for this table. A single non-unique index is defined on the concatenation of the PropID and PropValue columns.

Column Name	Data Type	Description
IntID	RTIntID	The internal identifier for the object to which this annotational property is attached.
PropID	RTIntID	The internal identifier for the inverted property definition object to which this annotational property instance conforms.
PropValue	RTShortString	The value of the annotational property instance.

RTblRelshipProps SQL Table

See Also

An annotational property is a property that is associated with an individual repository object or relationship, and is not required to exist for all repository objects or relationships of a particular type. This SQL table contains one row for each annotational property instance that is attached to a *relationship*. The OrgID, RelTypeID, DstID, and PropID columns comprise the primary key for this table. A single non-unique index is defined on the concatenation of the PropID and PropValue columns.

Column Name	Data Type	Description
OrgID	RTIntID	The internal identifier for the origin object of the relationship.
RelTypeID	RTIntID	The internal identifier for the relationship type.
DstID	RTIntID	The internal identifier for the destination object of the relationship.
PropID	RTIntID	The internal identifier for the inverted property definition object to which this annotational property instance conforms.
PropValue	RTShortString	The value of the annotational property instance.

RTblRelships SQL Table

See Also

This SQL table contains one row for each relationship that is stored in the repository. The OrgID, RelTypeID, and DstID columns comprise the primary key for this table.

Column Name	Data Type	Description
OrgID	RTIntID	The internal identifier for the origin object of the relationship.
RelTypeID	RTIntID	The internal identifier for the relationship type.
DstID	RTIntID	The internal identifier for the destination object of the relationship.
OrgTypeID	RTIntID	The internal identifier for the class to which the origin object conforms. Redundantly stored in this table for performance reasons.
DstTypeID	RTIntID	The internal identifier for the class to which the destination object conforms. Redundantly stored in this table for performance reasons.
PrevDstID	RTIntID	For sequenced relationship collections, this field contains the internal identifier of the previous relationship in the collection sequence. If the relationship is not a sequencing relationship, then this field is null.
DstName	RTNameString	The name of the destination object, as defined by this relationship. If the relationship is not a naming relationship, then this field is null.
DstNameLong	RTLongString	If the name of the destination object is longer than what can be fit into the *DstName* field, then this field contains the full name. Otherwise, this field is null.

RTblRelColDefs SQL Table

See Also

Each relationship in the repository is associated with two relationship collections: an origin collection and a destination collection. Each relationship collection conforms to a collection type (also referred to as a collection definition). The collection type defines the *role* that the collection plays in the relationship. This SQL table contains one row for each collection type that is defined in the repository. The IntID column is the primary key for this table. A non-unique index is defined for the RelTypeID column.

Column Name	Data Type	Description
IntID	RTIntID	The internal identifier for the collection definition object.
RelTypeID	RTIntID	Internal identifier of the relationship definition object.
IsOrigin	RTBoolean	Determines whether or not collections that conform to this collection type are origin collections (True), or destination collections (False).
Flags	RTFLAGS	Flags that determine the behavior of collections that conform to this collection type. See CollectionDefFlags for a list of values and their specific purposes.
MinCount	RTCount	The minimum number of repository items that are expected for this type of collection. This field can be null.
MaxCount	RTCount	The maximum number of repository items that are expected for this type of collection. This field can be null.

RTblRelColPairs SQL Table

See Also

Each relationship in the repository is associated with two repository objects: an origin object and a destination object. The RTblRelcolPairs table is used by the repository engine to determine, for any given relationship type, which object classes can assume the origin and destination roles.

This SQL table contains one row for each pair of object classes that are valid participants in a particular relationship type. Since relationships are actually attached to interfaces, and since multiple classes can implement a particular interface, more than one type of object can potentially assume the role of origin or destination for a particular relationship type. Consequently, there can be multiple rows in the table for a single relationship type.

The RelTypeID, OrgTypeID, and DstTypeID columns comprise the primary key for this table.

Column Name	Data Type	Description
RelTypeID	RTIntID	The internal identifier for the relationship definition object.
OrgTypeID	RTIntID	The internal identifier for the class whose instances can assume the role of origin in this type of relationship.
DstTypeID	RTIntID	The internal identifier for the class whose instances can assume the role of destination in this type of relationship.

RTblSites SQL Table

See Also

This SQL table contains one row for each repository site that is known to this repository. The table provides a translation capability between the global site identifier that uniquely identifies a site across all repositories and the local site identifier, which is unique only within the current repository. The smaller local site identifier is a part of the internal identifier that is used to identify a repository object within the repository.

The SiteID column is the primary key for this table. There is a unique index defined on the SiteGUID column.

Column Name	Data Type	Description
SiteID	RTSiteID	The site identifier for a repository site that is known to this repository.
SiteGUID	RTGUID	The global identifier for the site.
NextLocalID	RTLocalID	Reserved for proprietary usage by the repository engine.

RTblTypeLibs SQL Table

See Also

This SQL table relates the internal object identifiers for repository type libraries (tool information models) to their corresponding global identifiers. The IntID column is the primary key for this table.

Column Name	Data Type	Description
IntID	RTIntID	The internal identifier for the repository type library.
TypeLibID	RTGUID	The global identifier for the repository type library, as recorded in the system registry.

Repository Glossary

A
B
C
D
E
F
G
H
I
J
K
L
M
N
O
P
Q
R
S
T
U
V
W
X
Y
Z

A–C

annotational property

class

collection

collection type

COM object

D–F

default interface

definition

destination collection

destination collection type

destination object

G–I

global identifier

instance

interface

interface definition

internal identifier

inverted property

item

J–M

MDO Model

member

metadata

method

Microsoft Development Objects Model

N

naming collection

naming relationship

O

object

object identifier

origin collection

origin collection type

origin object

P–Q

populate

property

property value

R

relationship

relationship type

repository

repository database

repository engine

repository instance

repository item

repository object

repository type library

RepVB

S

sequenced collection

sequenced relationship

source object

T

target object

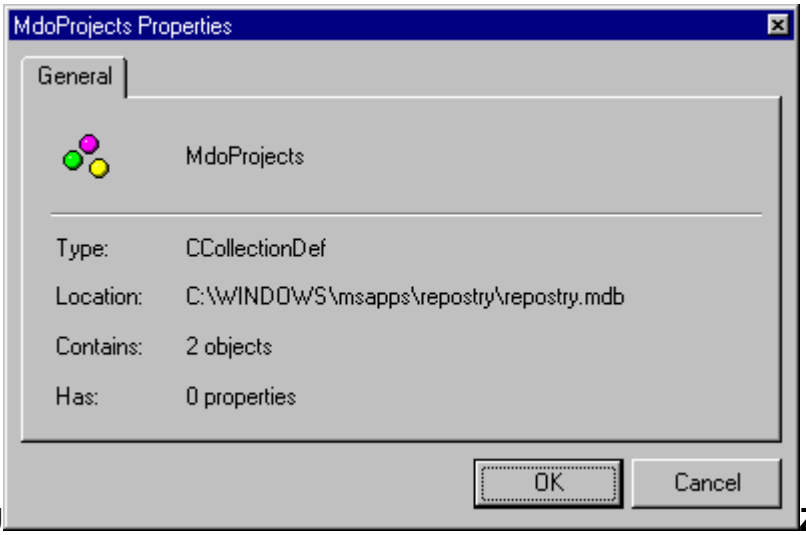
tool information

tool information model

type

type information model

type library



U Z

unique-naming collection

unique-naming relationship

annotational property

A special class of repository property that can be associated with individual repository objects or relationships. Properties that are not annotational are defined for all objects of a class, and cannot be associated with relationships.

class

The definition of a COM object. The class is the template from which an object instance is created. The class specifies which interfaces the object implements.

collection

1. A set of repository relationships of the same relationship type that are all connected to a common source object.
2. A set of repository objects that are all connected via instances of the same relationship type to a common source object. See origin collection, destination collection.
3. A set of items returned from a query to a repository.

collection type

The type of a collection. It is determined by the relationship type associated with the collection relative to the role that the objects in the collection play; that is, whether the objects are origin objects or destination objects. See origin collection type, destination collection type.

COM object

An object that conforms to the Component Object Model. A COM object implements the COM interfaces that support object interaction, as well as interfaces that are specific to that class of object. All repository objects are instantiated as COM objects.

default interface

The primary interface of a class. Visual Basic allows programmers to refer to the members of the default interface directly, without having to specify the interface by name.

definition

Sometimes used as a synonym for type; for example relationship definition is a synonym for relationship type.

destination collection

A collection of repository items that all have a common destination object.

destination collection type

The type of a collection of repository items that all have a common destination object.

destination object

A repository object that is the destination of a relationship. Consider the relationship: Project-Has-Component. Project is the origin of the relationship, and Component is the destination of the relationship.

global identifier

A globally unique identifier, or GUID. Used to uniquely identify COM classes and interfaces.

instance

An item that conforms to a particular definition, class, or template. For example, the instance of a class is an object; the instance of a relationship type is a relationship.

interface

A defined set of properties, methods, and collections that form a logical grouping of behaviors and data. Classes are defined by the interfaces that they implement. An interface may be implemented by many different classes.

interface definition

A synonym for interface; used when the perspective is manipulating the definition of an interface in the repository, as opposed to conforming to an interface during the execution of a program.

internal identifier

Internal identifier of an object within the repository. The internal identifier represents a more compact form of an object identifier. An internal identifier is only guaranteed to be unique within a single repository. An object identifier is guaranteed to be unique across all repositories.

inverted property

Synonym for annotational property.

item

A repository object or relationship. More specifically, any instance of a repository-supplied class that implements the IRepositoryItem interface.

MDO Model

The Microsoft Development Objects Model.

member

A property, method, or collection type that has been defined to be a part of an interface.

metadata

Metadata is data about data.

method

An invocable function that is defined to be a member of an interface.

Microsoft Development Objects Model

A tool information model provided with the repository that models Visual Basic projects.

naming relationship

A repository relationship that provides a name for its destination object.

naming collection

A collection of repository items whose associated relationship type requires the naming of destination objects.

object

An instance of a class. The term object is not a synonym for the term repository object. While all repository objects are indeed objects, all objects are not repository objects. See COM object, repository object.

object identifier

The identifier of an object within the repository. An object identifier is guaranteed to be unique across all repositories. See internal identifier.

origin collection

A collection of repository items that all have a common origin object.

origin collection type

The type of a collection of repository items that all have a common origin object.

origin object

A repository object that is the origin of a relationship. Consider the relationship: Project-Has-Component. Project is the origin of the relationship, and Component is the destination of the relationship.

populate

To provide with instances. For example, you populate the MDO Model with projects, components, and references.

property

A scalar attribute that is defined as a member of an interface. A property has an assigned data type; for example, string, or 32 bit integer. A property is a part of the definition of an interface. A property value is an instance of a property.

property value

The value stored for a particular instance of an interface property.

relationship

A logical connection between one object, the origin object, and a second object, the destination object. Relationships are instantiated by the repository as COM objects.

relationship type

The definition of a kind of relationship.

repository

A database containing tool information models, in conjunction with the executable software that manages the database. An installation of the Microsoft Repository.

repository database

A database used as persistent storage for tool information. The type information model is also stored in the repository database.

repository engine

The object-oriented Microsoft Repository software that provides management support for and customer access to a repository database.

repository instance

An instance of the Repository class. A repository instance represents an open connection to a specific repository database.

Note An instance of the Repository class is typically *not* referred to as a repository object to avoid confusion between instances of the Repository class and instances of the RepositoryObject class.

repository item

A repository object or a repository relationship.

repository object

An instance of the RepositoryObject class. All repository objects are COM objects; however, not all COM objects are repository objects. See object, COM object.

repository type library

A repository type library groups together the various definition objects that comprise a tool information model. Repository type libraries are used by the repository engine to provide Automation support. Not the same as a type library; see type library.

RepVB

The Microsoft Repository Add-in module for Visual Basic. It populates and maintains the MDO Model with tool information about the customer's Visual Basic projects.

sequenced relationship

A repository relationship that provides an explicitly specified position for its destination object among the collection of destination objects.

sequenced collection

A collection of repository items whose associated relationship type supports the explicit sequencing of destination objects.

source object

The single object to which all objects in a particular collection are connected via relationships that are all of the same relationship type. For destination collections, the source object is the destination object. For origin collections, the source object is the origin object.

target object

One of the objects in a particular collection to which a source object is connected via a relationship. For destination collections, the target objects are origin objects. For origin collections, the target objects are destination objects.

tool information

Instance data for a tool information model.

tool information model

An object model that is stored in a repository. Built using the basic elements of the type information model: relationship types, classes, interfaces, properties, methods, and collection types.

type

The definition of a particular kind of data.

type information model

The object model that the repository uses to store tool information models.

type library

A file (or component within another file) that contains Automation standard descriptions of exposed objects, properties, methods, and collections. Object library (.OLB) files contain type libraries. Type libraries that are shipped as stand-alone files use the extension .TLB

unique-naming collection

A collection of repository items whose associated relationship type requires the unique naming of destination objects.

unique-naming relationship

A repository relationship that provides a name for its destination object that is unique among the collection of destination objects.