
Script Client API

Version 4 Specification and Reference

First Draft

Revision Date

8/1/01

(c)2000 Computer Associates Int., Inc.
Princeton Lab
Rt.206 and Orchard Road
Princeton, New Jersey 08543-0008

REVISION DATE	1
8/1/01	1
INTRODUCTION AND OVERVIEW	4
<i>Major Features</i>	<i>4</i>
<i>Motivation.....</i>	<i>4</i>
<i>Typical Use-Cases</i>	<i>5</i>
A stand-alone client.....	5
A script or add-in component.....	5
MAJOR COMPONENTS AND TERMS	7
<i>Major Components</i>	<i>7</i>
<i>Access to Model Data</i>	<i>11</i>
<i>Object Identifiers</i>	<i>11</i>
<i>Object Identifiers and Type Codes.....</i>	<i>12</i>
<i>Collections and Automation.....</i>	<i>12</i>
<i>_NewEnum property of a collection object.....</i>	<i>13</i>
ACTIVE SCRIPTING	15
API DESCRIPTION	16
APPLICATION TIER.....	16
<i>Application Component</i>	<i>16</i>
<i>Application Environment</i>	<i>17</i>
<i>Application Services Collection.....</i>	<i>19</i>
<i>Persistence Units Collection.....</i>	<i>20</i>
<i>Persistence Unit.....</i>	<i>23</i>
<i>Property Bag.....</i>	<i>24</i>
<i>Property Bag for Persistence Units and Unit Collections</i>	<i>26</i>
SESSIONS TIER.....	27
<i>Sessions Collection</i>	<i>27</i>
<i>Session</i>	<i>28</i>
MODEL TIER.....	31
<i>Model Objects Collections.....</i>	<i>32</i>
<i>Model Object.....</i>	<i>35</i>
<i>Model Properties collection.....</i>	<i>38</i>
<i>Model Property</i>	<i>40</i>
<i>Property Values collection.....</i>	<i>44</i>
<i>Property Value Component.....</i>	<i>45</i>

INDEX.....47

Introduction and Overview

The Erwin Script Client API (SCAPI) provides advanced customization capabilities that enable you to access and manipulate modeling data in ERwin memory at run-time, as well as models persisted in files and in the ModelMart repository. The SCAPI interfaces are automation-compatible and provide extensive design and run-time facilities for third party integrators as well as end-users of script based environments.

The Script Client API enables you to complement the original modeling tool with custom components via extensive use of scripts, add-ins and COM-based API technologies. This promotes seamless integration of the modeling tool in a client development cycle and therefore contributes to overall success of the product. It demonstrates ERwin's dedication to comply with today's world of "open software" - the requirement to equip a product with extensive capacities for customization.

Major Features

The SCAPI is a name for a group of interfaces that include the following functionality:

- I. *Active Model Data Objects (AMDO)*. Supports the ability of a third-party client to access model data via a COM automation compatible API. This is the major component in the SCAPI functionality. All interfaces that comprise the SCAPI are automation based, hence are *dual*.

Collections and enumerators in the SCAPI are facilitating certain programming constructions in script languages that target these automation features.

In order to embrace events sync facilities of languages, the SCAPI delivers a collection of *connection points* interfaces and support for the *TypeInfo2* interface.

Automation rich error handling supported via *IErrorInfo* interfaces exposed by the SCAPI components.
- II. *Active Model Directory*. A set of features that allow a client to navigate available model storage, including ModelMart. The SCAPI delivers the ability for a client to open and/or to create a model in a file as well as from a ModelMart repository.
- III. *Active Scripting*. Hosts a scripting environment and provides an invocation mechanism for script and add-in components. Add-ins, scriptlets register with the *Active Scripting* environment via a provided mechanism.

Motivation

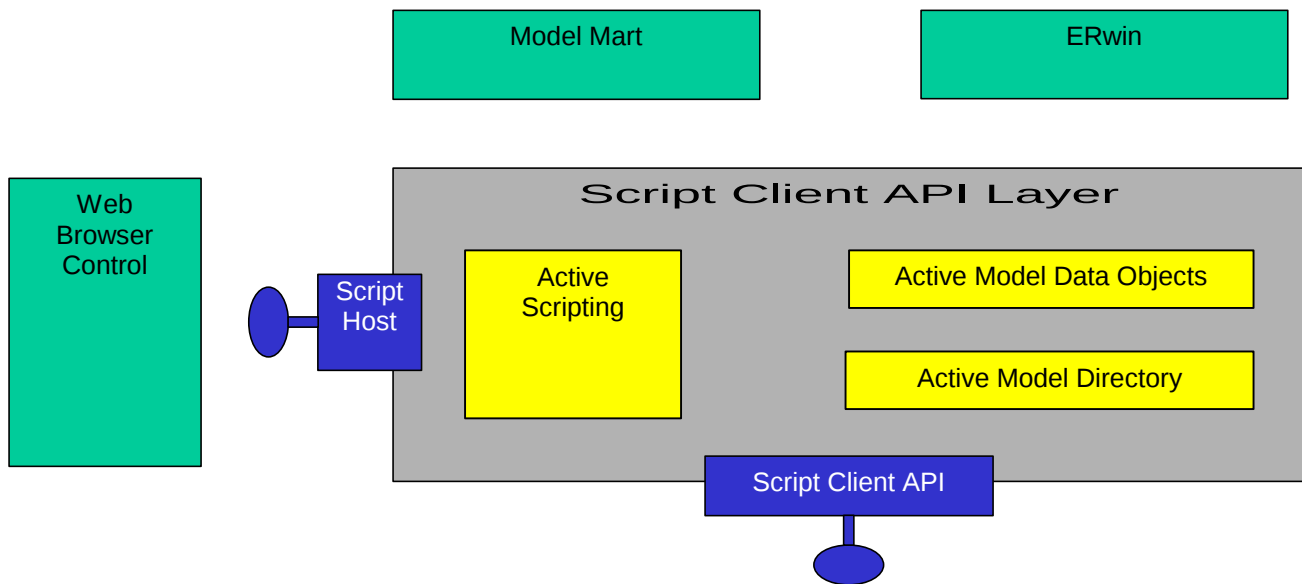
Third-party tool integrators represent a wide range of customers who wish to use the tool-provided functionality with complex business processes. From ERwin's perspective, such customers constitute a unique problem domain which the Script Client API seeks to address.

The majority of the targeted domain customers consist of automation and script-based clients. Such clients have very specific interface design requirements imposed by COM automation standards. For instance, they are often limited to one number of incoming and outgoing interfaces exposed by any particular COM object. The limitation is due to the fact that the only recognizable interface type for pure automation is *IDispatch* and it renders the use of *QueryInterface* functionality unfit. The common technique to address the problem includes Alternate Identities and read-only properties that expose secondary interfaces.

Another example of a targeted domain customer would be one using alternative (not C++) languages to implement a client. The list includes Visual Basic, VB Script, Java Script, etc. The list includes specially tailored language idioms to encapsulate language-COM binding, such as collections of objects, connection points, rich error handling, etc.

The SCAPI combines number of components and presents them as a set of interfaces accessible via COM.

On the list of integrated components are: ERwin, Model Mart, Microsoft Internet Explorer etc.



Typical Use-Cases

A stand-alone client

A third-party client can activate the API as an in-process server. The API component doesn't have visual representation, that is, it doesn't expose a user interface. The API provides Active Model Directory facilities to specify a target model from a list of available models. Active Model Data Objects provides a session-based access to model data.

The API clients compete with other parties over the access to model data. For model storage such as ModelMart, advanced model sharing facilities are available. Depending on the application-selected locking scenario, other parties could be prevented from accessing a model for the session duration.

A script or add-in component

ERwin hosts third-party add-in modules and scripts. The API component Active Scripting provides a mechanism for registration modules with a host tool, arranges representation in the host UI, add-in menus, and invokes it on the host menu selection or event.

An add-in module is a client DLL, activated in-process.

A script is a VB Script and J Script procedure embedded in a DHTML document, activated via a menu or a model event. The Active Scripting provides hosting for Web Browser control and makes the API objects available via the window.external property of the DHTML object model.

A client has an ability to observe changes in a model on the screen and can activate a pause to investigate the state of a model by accessing the modeling tool UI.

Major Components and Terms

The API is a collection of interfaces that represent ERwin functionality. The application exports the top-level interface, from which the client fetches lower-level interfaces as the need arises. Interfaces are logically grouped in tiers, where each tier includes interfaces that represent certain aspects of the application functionality.

Major Components

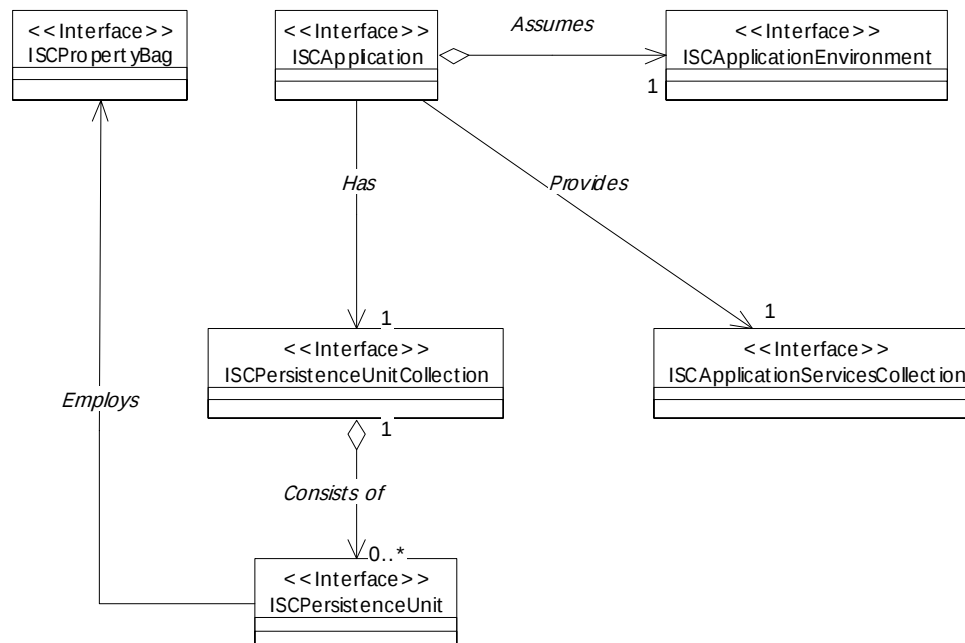
The API exposes the following major interfaces:

Interface	Role
Application Tier	
ISCAApplication	Represents application-wide functionality, and serves as the entry point for the API's interface hierarchy. Holds list of available persistence units, and connections between the client and persistence units.
ISCAApplicationEnvironment	Renders information about run-time environment.
ISCAApplicationServiceCollection	Provides access to the variety of application services, such as re-engineering, complete compare, etc.
ISCAApplicationService	Represents an application service
ISCPersistenceUnitCollection	Collects all active persistence units known to the application
ISCPersistenceUnit	Represents an active persistence unit (e.g., ERwin model) within the application. Clients can connect to persistence units to manipulate them and the data they contain.
ISCPROPERTYBag	Represents an array of properties for application tier interface calls.
Model Directory Tier	
ISCMModelDirectory	Represent model storages.
ISCMDElement	Renders information about available models.
Sessions Tier	
ISCSessionCollection	Collects all active sessions between the API client and persistent units.
ISCSession	Represents an active connection between client and a model. Clients create sessions then open them against persistence units. An open session exposes a single level (i.e., data, metadata, etc.) of a persistence unit.
ISCPProgress	Shared between a client and the application. Used to report progress and cancellation requests.
Model Tier	
ISCOBJECTFilter	Filters objects for the object iterator. Permits the API client to winnow out unwanted objects during a model traversal.
ISCOBJECTSorter	Sorts object for the object iterator. Permits the API client to determine the order in which objects are returned during a model traversal.

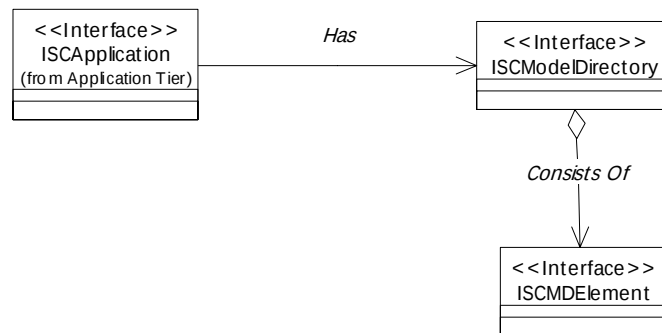
Interface	Role
ISCModelObjectCollection	Represents objects available for manipulation. Membership in collection can be limited by establishing filter criteria.
ISCModelObject	Access and manipulates a single object within a model
ISCPropertyFilter	Permits the API client to winnow unwanted properties while traversing an object's properties.
ISCPropertySorter	Permits the API client to set the order in which properties are returned while traversing an object's properties.
ISCModelPropCollection	Represents a list of properties owned by a single object. The list could be limited due to filtering.
ISCModelProperty	Accesses and manipulates a single property. Note that properties may contain multiple values. Values within a multi-valued property are accessed by keys. The current multi-valued property implementation treats the value list as an array, with the key being the array index.
ISCValueCollection	Represents a list of single property values.
ISCValue	Renders data and a key contained within a single value.

The objects are related in the following manner:

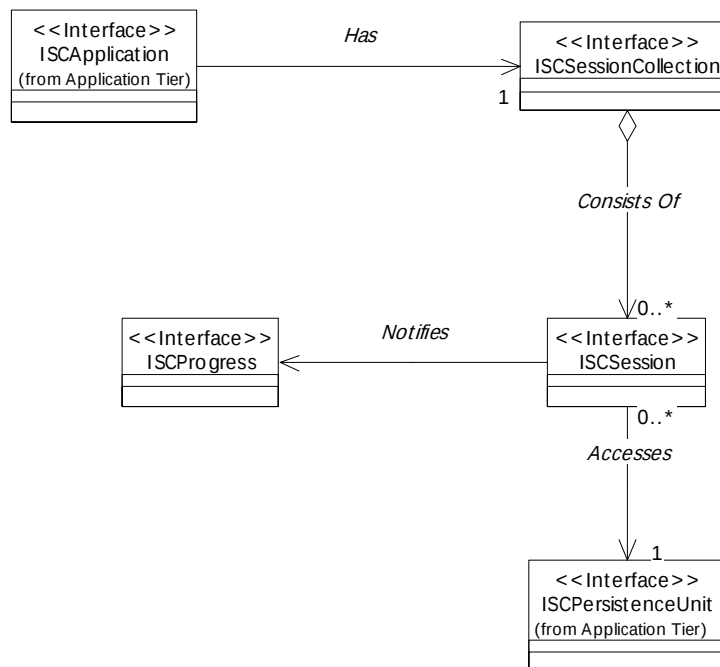
Application Tier



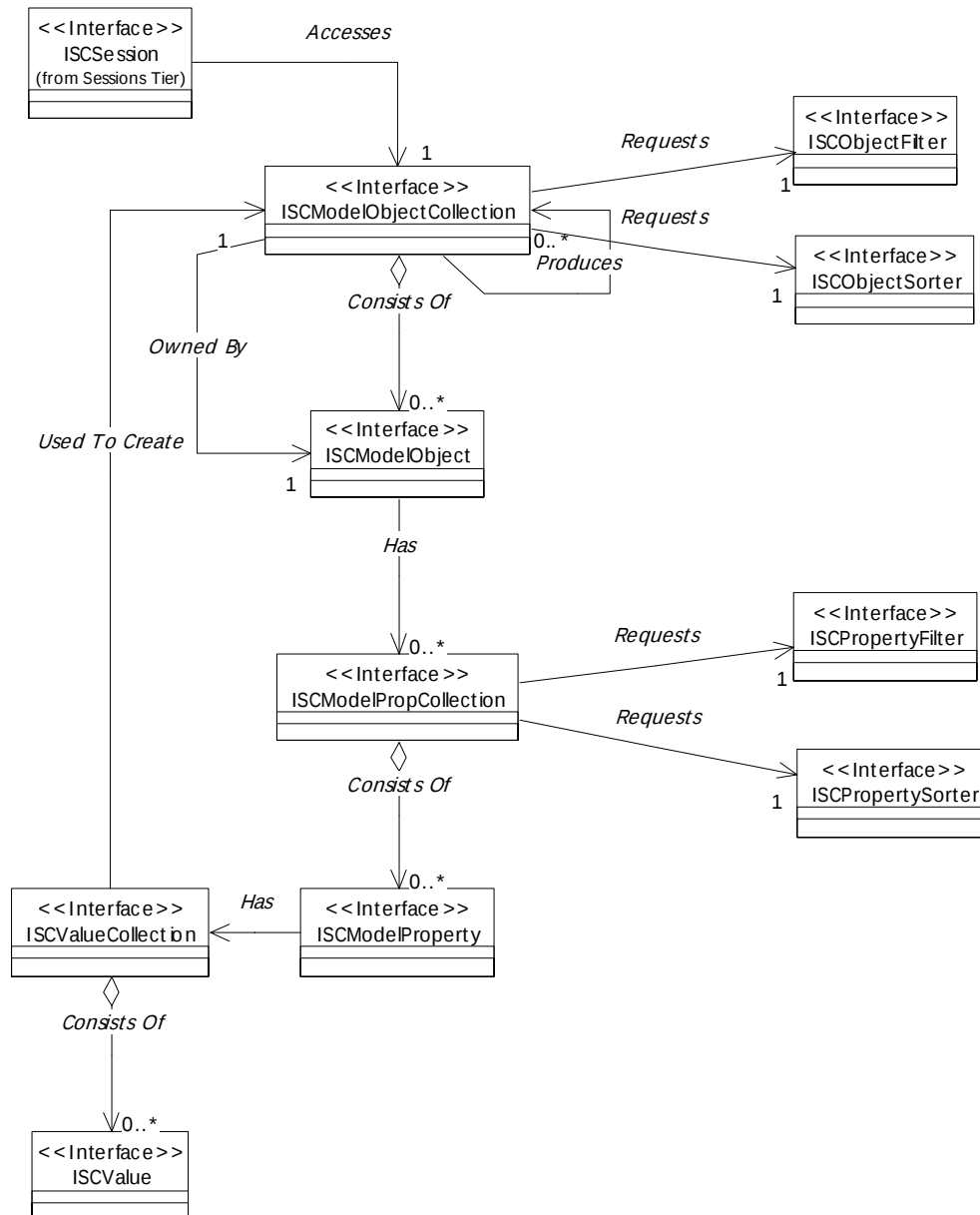
Model Directory Tier



Session Tier



Model Data Tier



Access to Model Data

ERwin implements the API to allow API clients to manipulate models.

The API clients locate models in persistence storages by using ISCModelDirectory. As soon as a model is located, the client registers it with the pool of available persistence units via ISCPersistenceUnits. At this point the client can specify access attributes such as read-only, ignore locks, etc. A new model can be created and registered with a persistence unit collection. ERwin itself can add or remove models from the pool as a response to UI actions.

A persistence unit maintains a set of properties to control visibility in the application UI, access attributes, etc.

The API clients access the model data by constructing a session and connecting it to a persistence unit via an ISCSession instance. GetObject file/ModelMart monikers provide an alternative way to register a model as a persistence unit and attach a session to it in one step.

A model contains several levels of data. It contains the data the application manipulates, such as entity instances, attribute instances, relationship instances and so on. The model also contains metadata: a description of the objects and properties that *may* occur within the application's data. In the case of ERwin, the metadata would define an entity class, an attribute class, a relationship class, and so on - one description for each class that may occur within a model. Clients specify the desired level of model data at the time of connecting a session to a persistence unit.

When a new model is created it acquires a set of default objects, such as model object, main subject area and stored display, etc.

The initial SCAPI implementation will support the following levels:

Name	Description	Supported Actions
SCD_SL_ M0	Model Level	Access model data, create and delete objects (including the entire model), set property values.
SCD_SL_ M1	Metamodel Level	Access object and property definitions, along with other metadata.

Levels are identified by long integer values. Values will have symbolic definitions.

Object Identifiers

The API presents data in object/property form. In an ERwin model, for example, an attribute would be represented by an instance of an attribute object; the attribute's name would be contained in the attribute object's name property.

Each object must bear an identifier – a value that uniquely identifies the object instance. Internally, object identifiers are 20-bytes long. They contain two components: a GUID (a.k.a. UUID) in the first 16 bytes, and a 32-bit unsigned integer suffix in the last 4 bytes.

A GUID contains the following components:

1. One 32-bit unsigned integer
2. Two 16-bit unsigned integers
3. Eight 8-bit unsigned integers (represented as unsigned characters)

for a total of 128 bits, or 16 bytes. Therefore, an object identifier contains an extra 32-bit unsigned integer (the 4-byte suffix) at the end for a total of 160 bits, or 20 bytes.

To simplify manipulating with object identifiers and due to COM automation limitations on datatypes, the API is using a string to represent identifiers.

The following table lists aliases used in the documentation and in the interface definitions:

Type Name	Format	Use
SC_OBJID	{xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxxx}+suffix	Object Identifier
SC_CLSID	{xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxxx}+suffix	Class (object or property type, etc.) identifier
SC_MODELTYPEID	{xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxxx}+suffix	Model type identifier
SC_CREATORID	{xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxxx}	Creator identifier

One set of object identifiers is predefined: those identifiers whose GUID component contains zero. If the final 4 bytes of the identifier also contain zero, the identifier represents a null identifier. Other values of the offset are reserved for future use.

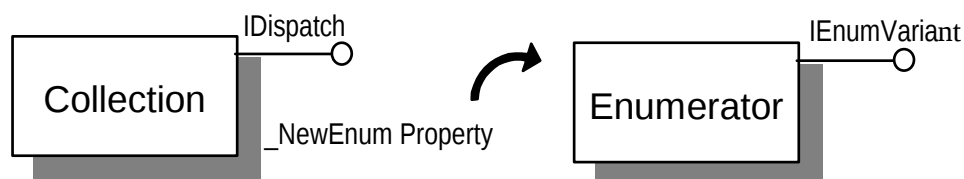
Object Identifiers and Type Codes

Consider the relationship between object instances in the SCD_SL_ M0 layer and object instances in the SCD_SL_ M1 layer. An instance in the SCD_SL_ M0 layer is described by an instance in the SCD_SL_ M1 layer. For instance, a single object in the SCD_SL_ M1 layer describes every entity instance in the SCD_SL_ M0 layer.

Since all type codes are also object identifiers, they must have the same format.

Collections and Automation

Automation defines the **IEnumVARIANT** interface to provide a standard way for the API clients to iterate over collections. Every collection interface in the API exposes a read-only property named **_NewEnum** to let the API clients know that the collection supports iteration. The **_NewEnum** property returns a pointer on **IEnumVARIANT** interface.



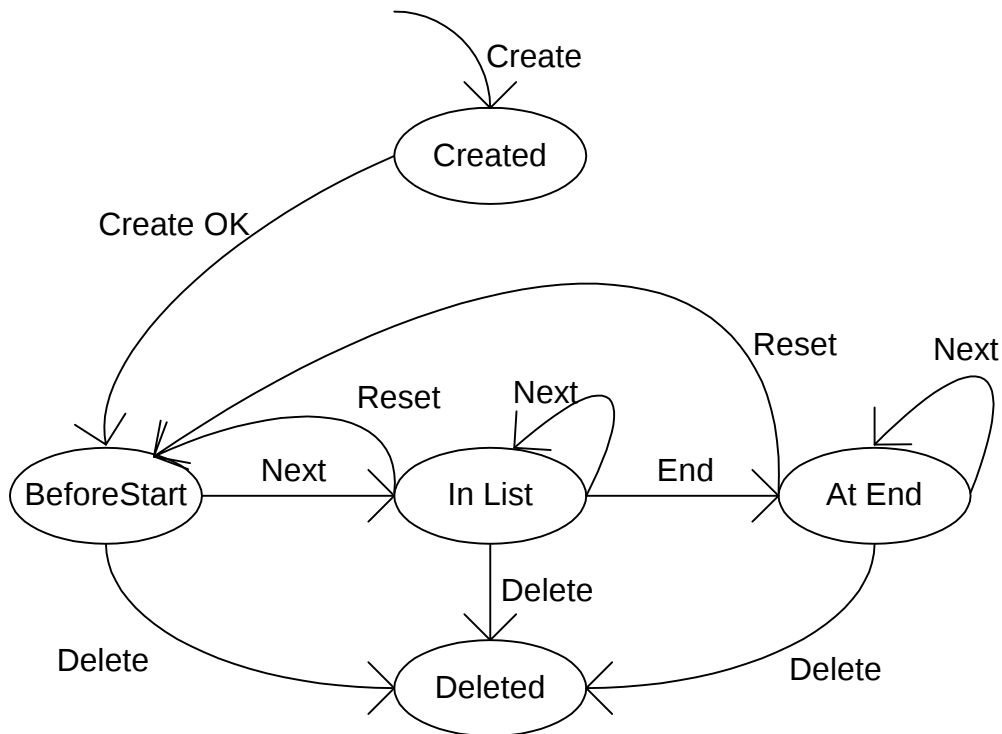
The **IEnumVARIANT** interface provides a way to iterate through the items contained by a collection. This interface is supported by an enumerator interface that is returned by the **_NewEnum** property of the collection, as in the figure above.

The **IEnumVARIANT** interface defines these member functions:

- Next — Retrieves one or more elements in a collection, starting with the current element.
- Skip — Skips over one or more elements in a collection.

- **Reset** — Resets the current element to the first element in the collection.
- **Clone** — Copies the current state of the enumeration so you can return to the current element after using **Skip** or **Reset**.

The **IEnumVARIANT** collection implements a Rogue Wave style “advance and return” iteration. Hence, they have the following life cycle.



When the iterator is created, it enters the Created state. It then forces itself into the BeforeStart state. A successful advance drives the iterator into the InList state, while an unsuccessful advance drives it into the AtEnd state. A Reset drives the iterator back to the BeforeStart state, and deletion drives it into the Deleted state.

The iterator is positioned over a member of the collection (i.e., is associated with a “current member”) if and only if it is in the InList state.

NewEnum property of a collection object

The NewEnum property identifies support for iteration through the IEnumVARIANT interface. This property has the following requirements:

- Named NewEnum.
- Returns a pointer to the enumerator IUnknown interface.

- Dispatch identification for the property is DISPID = DISPID_NEWENUM (-4).

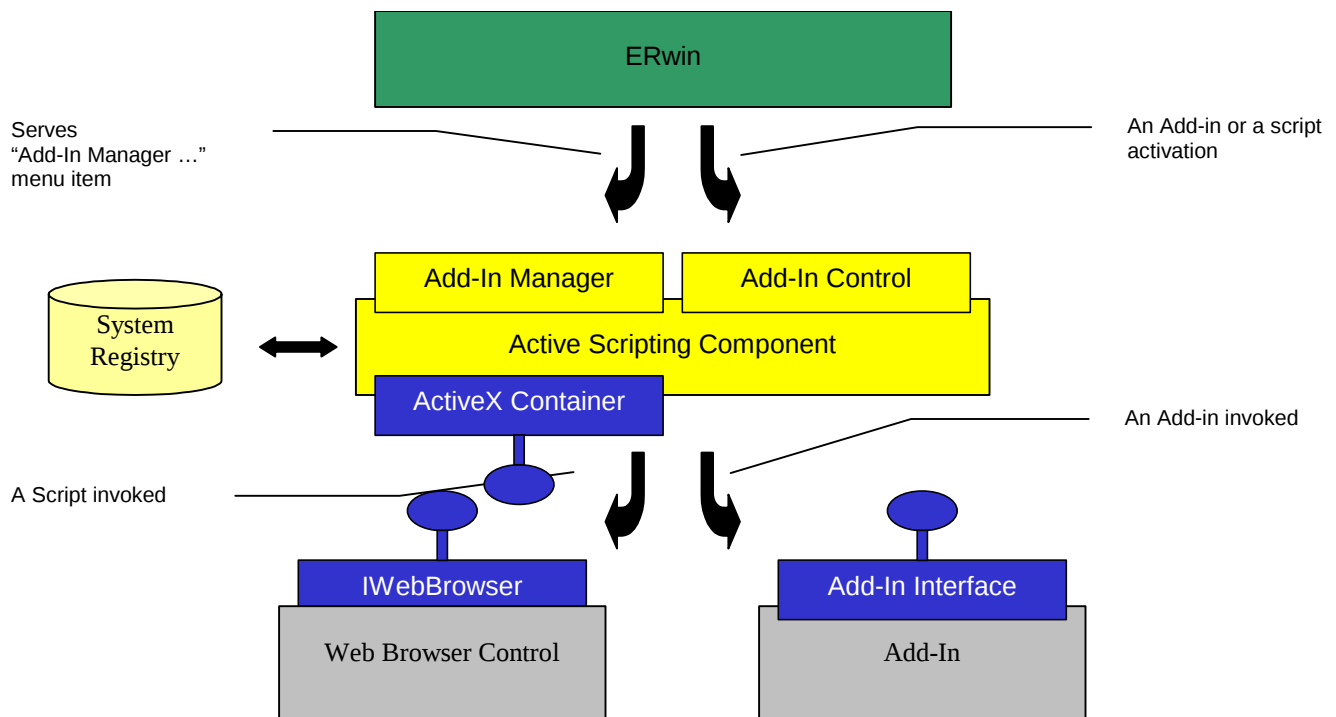
Active Scripting

To promote seamless integration of the modeling tool in a client development cycle and therefore contribute to the overall success of the product, ERwin features hosting of third-party add-in modules and scripts. The SCAPI component responsible for implementing hosting is Active Scripting. The Active Scripting provides a mechanism for registration modules with the application, arranges representation in the application UI and add-in menus, and invokes add-ins on behalf of the application menu selection or event.

An add-in module is a client DLL, activated in-process.

A script is a DHTML target that embeds graphics and script code.

The overall schema of the script and add-in hosting is illustrated in this picture:



API Description

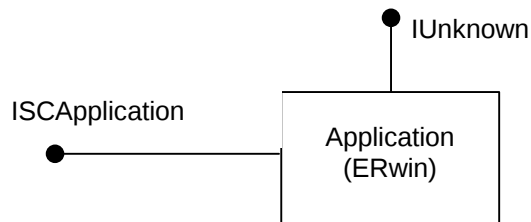
As stated above, the API is implemented as a tree of COM interfaces. The application exports the top-level interface, from which the client fetches lower-level interfaces as the need arises. The remainder of the API description documents the API in a top-down manner. It classifies each section into a **tier**, working down the topmost level that is associated with the exporting application.

Application Tier

The Application Tier provides application-wide functionality, and serves as the entry point for the API's interface hierarchy. This level exposes top-level services and directories.

Application Component

The Application Component encapsulates application-wide functionality, and serves as the root of the API's interface hierarchy. Only one instance of the component can be externally instantiated to activate the API. The client navigates the interface hierarchy by using interface properties and methods to gain access to the rest of the API functionality.



The following table describes the ISCAApplication interface properties:

ISCAApplication			
Property	Type	Read Only	Description
Name ^(Def.)	Bstr	Yes	Model tool application name (e.g., "Computer Associates ERwin").
Version	Bstr	Yes	Describes the application's version.
ApiVersion	Bstr	Yes	The API version

^(Def.) Default Property on an object. For the automation, a default property is the property that is accessed when the object is referred to without any explicit property or method call. The property dispatch identifier is DISPID_VALUE.

ApplicationEnvironment	ISCAApplicationEnvironment *	Yes	Reports attributes of run-time environment and available features, such as add-in mode, UI visibility, etc.
ApplicationServices	ISCAApplicationServiceCollection *	Yes	Provides access to the variety of application services, such as re-engineering, complete compare, etc.
ModelDirectories	ISCAModelDirectory *	Yes	Collects model directories accessible from this machine.
PersistenceUnits	ISCPersistenceUnitCollection *	Yes	Returns a collection of all persistence units loaded in the application.
Sessions	ISCSessionCollection *	Yes	Returns a collection of sessions created within the application.

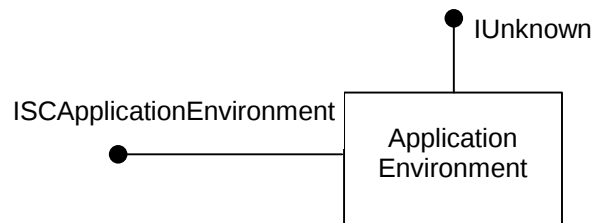
Note: The *Application* component exposes extra interfaces to assist *Active Scripting*.

The following table describes result codes generated by methods and properties of the interface.

ISCAApplication		
Error Code	Methods/Properties	Description
SCAPI_E_MD_CONSTRUCTIONFAILED	Run	Call to COM services to create an instance of ERwin server failed.
SCAPI_E_MD_CONSTRUCTIONFAILED1	Run	Requested mode is not supported

Application Environment

The Application Environment component renders information about the run-time environment. An API user accesses an Application Environment by using the ApplicationEnvironment property of the ISCAApplication interface.



The following table describes the ISCAApplicationEnvironment interface properties:

ISCAApplicationEnvironment			
Property	Type	Read Only	Description
Name ^(Def.)	Bstr		Application caption. Available only if the application UI is visible.
ProviderID	GUID	Yes	A GUID for the application object in the system registry.

The following table describes the ISCAApplicationEnvironment interface methods:

ISCAApplicationEnvironment	
Method	Description
long SupportsFeature(Bstr FeatureName)	Returns a level of support that is 0 if the application does not recognize the passed feature.

The API recognizes the following application features:

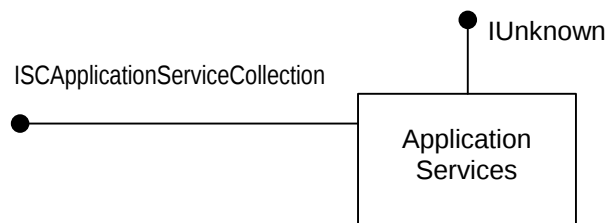
Name	Description	Returns
“Undo”	Ability to undo operations.	0 if undo not supported, nonzero if undo supported
“Redo”	Ability to redo undone operations.	0 if redo is not supported, nonzero if redo is supported.
“Change Logging”	Ability to report all changes since last synchronization with client.	0 if change logging not supported, nonzero if change logging supported.
“Ownership Support”	Queries the application’s support level for object ownership.	0 if the application does not support object ownership. 1 if the application supports ownership and the ownership meta-relation contains no cycles. 2 if the application supports ownership and the ownership metarelation contains cycles.
“Transactions”	Support for transaction control.	An integer indicating the level of support, as follows: 0 No support for transactions 1 Begin/End only. No nesting 2 Begin, End and Rollback.

^(Def.) Default Property on an object. For the automation, a default property is the property that is accessed when the object is referred to without any explicit property or method call. The property dispatch identifier is DISPID_VALUE.

Name	Description	Returns
		3 No nesting Begin, End, and Rollback, with arbitrary transaction nesting.

Application Services Collection

The Application Services Collection represents a variety of application services provided by the application, such as re-engineering, complete compare, etc. An API client accesses the Application Services Collection by using the ApplicationServices property of the ISCAApplication interface. eee



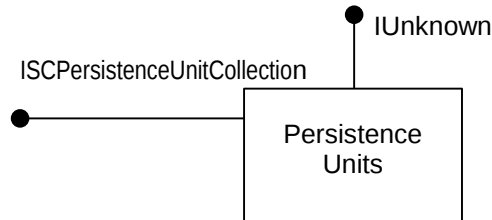
The following table describes the ISCAApplicationServicesCollection interface properties:

ISCAApplicationServicesCollection			
Property	Type	Read Only	Description
Count	long	Yes	Number of available services.
Item(long Index) ^(Def.)	IUnknown *	Yes	An instance of ISCAApplicationService interface associated with the passed index. The index is 0-based. Returns the collection if the argument is omitted.
_NewEnum	IUnknown*	Yes	Hidden property. Construct an instance of unit enumerator object and returns IEnumVariant interface pointer of the enumerator. Every call to the Next method of IEnumVariant will return one or more Variant elements with ISCAApplicationServices *, or NULL if no more objects are available.

^(Def.) Default Property on an object. For the automation, a default property is the property that is accessed when the object is referred to without any explicit property or method call. The dispatch index for the property is DISPID_VALUE.

Persistence Units Collection

A Persistence Units Collection contains all outer level persistence units loaded in the application. It contains one entry for each active data model. There is only one instance of the collection in the application. The interface pointer can be reached via the PersistenceUnits method of the ISCAApplication interface.



Existences of some persistence units in the application are dictated by a context in which an instance of the application was created. For instance:

- If a client is using the API as in-process server, in that case none of units will exist at launch time. Methods from the unit collection interface must be used to accumulate units in the collection.
- For a script or add-in component, the collection contains all units known to the application at the time when the client component was activated.

Similar to the arrangement for units when the client program is over:

- If a client is using the API as in-process server, then all units will be closed.
- For a script or add-in component, after the client program is over, units are still open and available in the application UI with exception of those that were explicitly closed and removed from the persistence unit collection before exiting the program.

Note that for ERwin, the collection is a snapshot. The collection will include only those units that exist at the moment of collection construction (i.e. at the moment when PersistenceUnits method of ISCAApplication interface was called). An exception to this, is units added or deleted from the collection. These changes will be reflected. All new collections will reflect changes as well.

The following table describes the ISCPersistenceUnitCollection interface properties:

ISCPersistenceUnitCollection			
Property	Type	Read Only	Description
Count	long	Yes	Number of units in the collection.
Item(VARIANT Selector) (Def.)	IUnknown*	Yes	Passes back an ISCPersistenceUnit pointer identified by <i>Selector</i> . The <i>Selector</i> parameter identifies a persistence unit by: • An ordered position index. The index is 0-based;

(Def.) Default Property on an object. For the automation, a default property is the property that is accessed when the object is referred to without any explicit property or method call. The dispatch index for the property is DISPID_VALUE.

ISCPersistenceUnitCollection			
Property	Type	Read Only	Description
			- Persistence Unit Object Id; For units with the same unit id, only one of units could be addressed in this way without guaranty which one. - A ISCPersistenceUnit pointer. In such case the object is cloned Passes back a pointer to self if the argument is omitted.
_NewEnum	IUnknown*	Yes	Hidden property. Construct an instance of unit enumerator object and returns IEnumVariant interface pointer of the enumerator. Every call to the Next method of IEnumVariant will return one or more Variant elements with ISCPersistsnceUnit interface pointers, or NULL if no more objects are available.

The following table describes the ISCPersistenceUnitCollection interface methods:

ISCPersistenceUnitCollection	
Method	Description
ISCPersistenceUnit * Add(VARIANT Locator, [optional] ^(opt) BSTR Disposition)	Adds a new persistence unit to the unit collection and returns a ISCPersistenceUnit interface pointer. Returns NULL if the method failed. The unit has hidden status in the application UI if the one active and the client responsible to render it visible via ISCPersistnceUnit interface if this is a requirement. The <i>Locator</i> parameter identifies a location for the persistence unit data source in one of the following ways: - As a string with a file or ModelMart item location along with attributes required for successful access to the storage. - As a ISCMDElement pointer with information sufficient for successful access. An optional <i>Disposition</i> arranges access attributes, such as read only, type of locking, etc.

^(opt) Due to support automation client requirements, all optional parameters are represented as VARIANT. For that reason, a parameter type in an interface description is only to document an expected type in VARIANT structure.

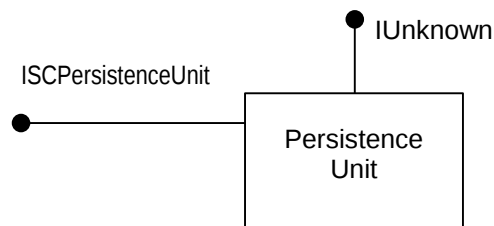
ISCPersistenceUnitCollection	
Method	Description
Boolean Remove(VARIANT Selector, [optional] ^(opt) Boolean Save)	Removes a persistence unit from the collection, closes associated window, if any. By default, all un-saved data will be saved unless <i>Save</i> parameter has false value or the unit has temporary status with unspecified location property. Failed if a save was requested while the unit has open sessions and the unit contents were modified. Otherwise, open sessions are ignored. The <i>Selector</i> parameter identifies a persistence unit by: • An ordered position index; • Persistence Unit Object Id; For units with the same unit id, only one of units could be addressed in this way without guaranty which one. • A ISCPersistenceUnit pointer
Boolean Clear()	Purges all units from the collection, closes associated window, if any. All active sessions are closed and all un-saved data will be lost.
Boolean Delete(VARIANT Selector)	Deletes data represented by persistent unit from the storage. Removes the persistence unit from the collection, closes associated window, if any. The <i>Selector</i> property identifies a persistence unit by: • An ordered position index; • Persistence Unit Object Id; For units with the same unit id, only one of units could be addressed in this way without guaranty which one. • A ISCPersistenceUnit pointer
ISCPersistenceUnit* Create(ISCPropertyBag * PropertyBag, [optional] SC_OBJID ObjectID)	Creates a new unit, registers the unit with the collection. <i>PropertyBag</i> supplies required and optional properties to the creating process, such as a type of the model, etc. Note that values provided as parameters in this method call override any instances of similar properties in the bag. <i>ObjectId</i> contains the new persistence unit's object identifier. If it contains a non-NULL value, it becomes the identifier for the new persistence unit. By default, the application assigns the new persistence unit's object identifier. Note that the new persistence unit contains data at all levels. Upon creation, its metamodel level (SCD_SL_ M1) contains the intrinsic metamodel associated with the passed model type, and the model

^(opt) Due to support automation client requirements, all optional parameters are represented as VARIANT. For that reason, a parameter type in an interface description is only to document an expected type in VARIANT structure.

ISCPersistenceUnitCollection	
Method	Description
	level (SCD_SL_ M0) includes all default objects, if such required, populated in the new model by the application.

Persistence Unit

A Persistence Unit encapsulates information required to connect to an existing, outer level persistence unit within an application. In the case of ERwin, the information allows the API client to connect to an ERwin model. Item and _NewEnum methods of ISCPersistenceUnitCollection interface help to locate Persistence Units.



The following table describes the ISCPersistenceUnit [VM2]interface properties:

ISCPersistenceUnit			
Property	Type	Read Only	Description
ObjectId	SC_OBJID	Yes	Object identifier for the persistence unit. Object identifiers are unique within the scope of the application. Note: The object identifier in ERwin is an identifier of a model object bounded to persistence unit, such as Model or Subject Area.
Name ^(Def.)	BSTR	Yes	Persistence unit name.
DirtyBit	Boolean		Triggered on if any changes were successfully applied to the data after the last time the property was set to false. Set to false when the unit was originally introduced to the

^(Def.) Default Property on an object. For the automation, a default property is the property that is accessed when the object is referred to without any explicit property or method call. The dispatch index for the property is DISPID_VALUE.

ISCPersistenceUnit			
Property	Type	Read Only	Description
			collection.
PropertyBag([optional]Boolean AsString)	ISCPROPERTYBag *		Returns or accepts a pointer on a property bag with the unit properties. When a property bag retrieved with optional <i>AsString</i> parameter set to true all values in the bag are presented as strings. The default is false for the parameter.

The following table describes the ISCPersistenceUnit interface methods:

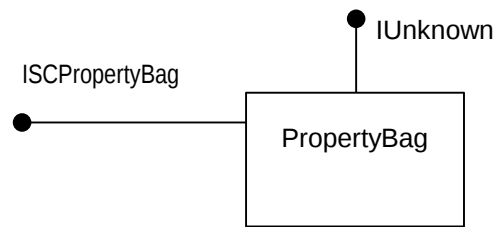
ISCPersistenceUnit	
Method	Description
Boolean Save([optional] ^(opt) VARIANT Locator, [optional]BSTR Disposition)	Persists model data to external storage regardless of which session provided changes. Storage attributes could be originated at the time when a unit was registered with the unit collection. The location and the persistence storage disposition also could be changed with this call as well by specifying <i>Locator</i> and <i>Disposition</i> parameters. The parameters are optional and render the call identical to Save As functionality. Uncommitted transactions are ignored.
Boolean HasSession()	Returns true if a unit has one or more sessions connected.
Boolean IsValid()	Returns TRUE if self is valid (i.e., occupies the Valid state), and FALSE otherwise. This method lets the caller detect when the referenced object is deleted.

Property Bag

A Property Bag is a placeholder for an array of properties. Some interfaces in the API, for instance ISCPersistenceUnit, employ the bag as a way to report and modify the state of data they represent. Hence, the content of the bag is dictated by such an interface.

A *Property Bag* interface pointer is available via the COM API CoCreateInstance call, as well as via employing interfaces.

^(opt) Due to support automation client requirements, all optional parameters are represented as VARIANT. For that reason, a parameter type in an interface description is only to document an expected type in VARIANT structure.



The following table describes the ISCPropertyBag interface properties:

ISCPropertyBag			
Property	Type	Read Only	Description
Count ^(Def.)	Long	Yes	Returns the number of properties in the property bag.
Value(VARIANT Property)	VARIANT		Retrieves, adds or changes the indicated property in the bag. <i>Property</i> either a name of a property in the property bag or a zero based property index. A property will be added, if the property with such name does not exist in the bag. Index cannot be used to add a new value.
Name(long Index)	BSTR	Yes	Retrieves a name of the indicated property in the bag. <i>Index</i> is zero based long number. The name could be used to retrieve a property value.

The following table describes the ISCPropertyBag interface methods:

ISCPropertyBag	
Method	Description
Void ClearAll()	Removes all properties from the bag
Boolean Add(BSTR Name, VARIANT Value)	Adds a new property to the bag. Does not check on duplicate names.
SAFEARRAY(VARIANT) GetPropInfo([optional] ^(opt) long Index)	Gets back an array of pairs for each property kept in the property bag. The first element of the pair, or an even element in the array is a property name, where the second or odd element is a property type. Optionally, zero based <i>Index</i> could be used to retrieve

^(Def.) Default Property on an object. For the automation, a default property is the property that is accessed when the object is referred to without any explicit property or method call. The dispatch index for the property is DISPID_VALUE.

^(opt) Due to support automation client requirements, all optional parameters are represented as VARIANT. For that reason, a parameter type in an interface description is only to document an expected type in VARIANT structure.

ISCPPropertyBag	
Method	Description
	description of a single parameter. The property type codes followed the one used for model properties. Please, see Model Property section for available SCAPI value type codes.

Property Bag for Persistence Units and Unit Collections

Property Bag provides access to properties of a persistence unit. In ERwin, this is a top-level model object, such as Model or Subject Area. The bag also has extras to represent *Locator* and *Disposition* parameters used to access the persistence unit.

An empty *Property Bag* could be obtained via a call to CoCreateInstance of the COM API. The client populates a bag and submits it as a parameter for Create a new unit call. Alternatively, the present state of persistence unit properties could be retrieved via the PropertyBag property of ISCPersistenceUnit. The retrieved value could be reviewed, modified and submitted back via the PropertyBag property of the same interface. The contents of the bag can have one of two available forms: native format or as strings based on the optional parameter of PropertyBag property of ISCPersistenceUnit. The client can populate the bag in any of these two forms and different forms could be mixed in the same instance of the bag.

Not all properties that exist in the unit have to be present in the bag when it is submitted. All property data as well as property names are validated by the API and either all are accepted or all rejected. The rejection will force a method call to fail. If the bag includes properties that are read-only at the moment, for instance the model type for an ERwin model, when the model was created beforehand, then such properties will be ignored and won't affect validation of the bag data.

The following is the list of *Property Bag* properties, recognized by ERwin, along with their datatypes:

Persistence Unit Property Bag			
Property Name	Type	Read Only	Description
Locator	BSTR	Yes	Returns the location of the persistence unit, such as file name, etc. Not available for the models without persistence location, such as new models that were never saved.
Disposition	BSTR	Yes	Returns the disposition of the persistence unit, such as read-only, etc.
Model Type	Long		Retrieves, sets type of the persistence unit, such as logical, logical-physical and physical models. Could be set when a persistence unit is created and read-only after then.
Target Server Target Server Version Target Server Minor Version	Long		Retrieves, sets target database properties for physical and logical-physical models. Could be set when a persistence unit is created and read-only after then.
Active Model	Boolean		True if the persistence unit represents the current model,

Persistence Unit Property Bag			
Property Name	Type	Read Only	Description
			active in ERwin user interface.
Hidden Model	Boolean		True if a model window with the persistence unit data is not visible in ERwin UI. Not available for the API stand-alone mode.

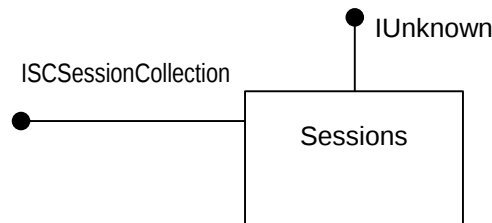
Sessions Tier

The Sessions Tier encapsulates session-wide services. The session represents a connection between a client and the application. Clients may maintain multiple active connections to applications.

The Sessions collection contains the active connections, each of which is represented by a Session instance.

Sessions Collection

The Sessions Collection contains the active connections between the API client and the application. The interface pointer can be reached via the *Sessions* method of the ISCAApplication interface.



The following table describes the ISCSessionsCollection interface properties:

ISCSessionsCollection			
Property	Type	Read Only	Description
Count	long	Yes	The number of <i>Session</i> in the collection.
Item(long Index) ^(Def.)	IUnknown*	Yes	Returns an ISCSessions interface pointer identified by its ordered position. The index is a long number and 0-based. Returns a pointer to self if the argument is omitted.
_NewEnum	IUnknown*	Yes	Hidden property. Construct an instance of session enumerator object and returns IEnumVariant interface

^(Def.) Default Property on an object. For the automation, a default property is the property that is accessed when the object is referred to without any explicit property or method call. The dispatch index for the property is DISPID_VALUE.

ISCSessionCollection			
Property	Type	Read Only	Description
			pointer of the enumerator. Every call to the Next method of IEnumVariant will return one or more Variant elements with ISCSession interface pointers, or NULL if no more objects are available.

The following table describes the ISCSessionCollection interface methods:

ISCSessionCollection	
Method	Description
ISCSession * Add ()	Construct a new, closed <i>Session</i> object, and adds it to the collection. Passes back an interface pointer to the newly created <i>Session</i> object.
Boolean Remove (VARIANT SessionID)	Removes a <i>Session</i> object from the collection. If the session is opened, it is closed before it is removed. All committed changes are saved in the persistence unit. The indexes for the remaining sessions will be adjusted to close the resulting gap. Returns FALSE if the supplied session identifier is invalid. The <i>SessionID</i> parameter is a variant that supports identifying a session in one of the following ways: • By numeric index; A number from 0 to Count-1; • By a <i>Session</i> object; A valid session object pointer.
Boolean Clear ()	Removes all <i>Session</i> objects from the collection. Has the same effect as calling Remove() for each session in the collection.

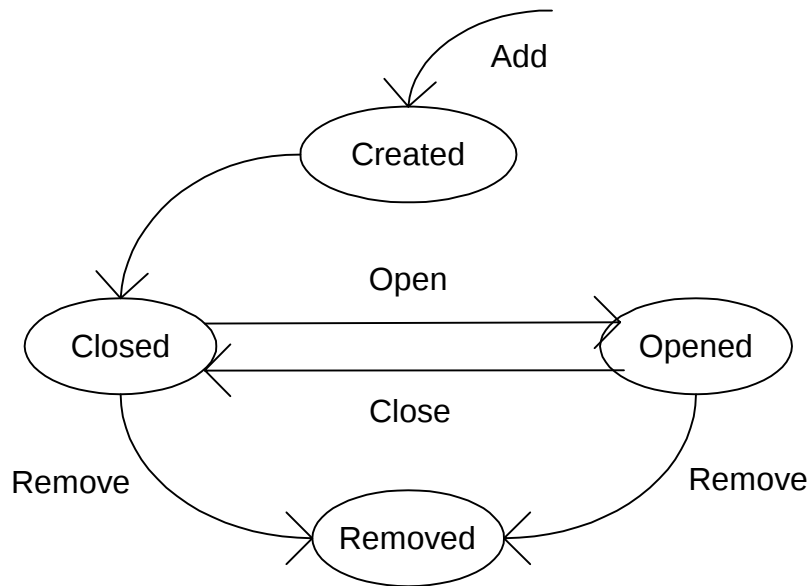
The following table describes result codes generated by methods and properties of the interface.

ISCSessionCollection		
Error Code	Methods/Properties	Description
SCAPI_E_SC_SESSIONOBJ	Remove	Provided pointer of an object either is not a session object or it is not registered in the session collection

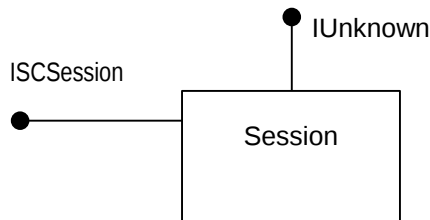
Session

The Session object connects an API client to a persistence unit in the application. Note that the persistence unit could represent a submodel. Item and _NewEnum methods of ISCSessionCollection interface help to locate Sessions.

It has the high-level status, illustrated in the following figure:



When a client creates a *Session* object, it enters the Created state, and immediately transitions into the Closed state, indicating that it is not yet associated with a persistence unit. The client can invoke one of several API methods to connect the *Session* object to a persistence unit (such as creating a new persistence unit, loading a persistence unit from the applications persistent store, etc.). The *Session* object will transition into the Opened state, allowing the client to manipulate the associated persistent unit. Later on, the client can disconnect the *Session* object from its associated persistence unit, and the *Session* object reenters the Closed state. It is then available for connection, and the cycle can be repeated. When the client is finished with the *Session* object, he removes it from its owning Sessions collection, and it enters its final state: Removed.



The following table describes the ISession interface properties:

ISession			
Property	Type	Read Only	Description
Name ^(Def.)	BSTR	Yes	Name of the associated persistence unit. Used for display only. Contains a valid name only when self is in the Opened state.

^(Def.) Default Property on an object. For the automation, a default property is the property that is accessed when the object is referred to without any explicit property or method call. The dispatch index for the property is DISPID_VALUE.

ModelObjects	ISCMModelObjectCollection *	Yes	Retrieves a <i>Model Objects</i> collection interface pointer for the session. If the session is open, the collection will contain all objects contained in the associated persistence unit. The returned collection will contain every object associated with the persistence unit. If the session is not open, the member will return NULL and reports an error.
PersistenceUnit	ISCPersistenceUnit *	Yes	<i>Persistence Unit</i> associated with the session. Contains a valid pointer only when self is in the Opened state.
Level	long	Yes	Returns the level at which the persistence unit is bound. This value is valid only if the session is open. See Access to Model Data for access level definition.

The following table describes an ISCSession interface methods:

ISCSession	
Method	Description
Boolean Close ()	Disconnects self from its associated persistence unit. Does not alter the associated persistence unit. Does nothing unless self is in the Opened state. Caller must have read access to the underlying model.
Boolean Open(ISCPersistenceUnit* Unit, [optional] ^(opt) long Level, [optional] Boolean Exclusive)	When self is in the closed state, this method binds self to the persistence unit identified by the <i>Unit</i> parameter. Binding occurs at the level specified in the <i>Level</i> parameter. If successful, self enters the Opened state and the root pointer becomes valid. Default for <i>Level</i> is model level. <i>Exclusive</i> identifies if a session requests an exclusive access to <i>Persistence Unit</i>. No other sessions would be allowed to access specified unit. In order to obtain exclusive privileges there should be no other sessions associated with the unit. By default, non-exclusive access requested. Note: if the passed the persistence unit absent, <i>Unit</i> contains NULL, the session is bound to the intrinsic model, if available. The intrinsic SCD_SL_M1 model contains all intrinsic metadata. The SCD_SL_M0 behavior of the intrinsic model is unspecified.
Boolean IsOpen()	Returns true if a session connected to a persistence unit.
Boolean ChangeAccess(Boolean Exclusive)	Changes the model access from regular to exclusive or reverse. In order to obtain exclusive privileges there should be no other sessions associated with the unit.
VARIANT BeginTransaction()	Opens a transaction on the session. The method passes a transaction identifier back. Implementations use the identifier to scope commit and rollback operations. If the application does not support nested transactions it

^(opt) Due to support automation client requirements, all optional parameters are represented as VARIANT. For that reason, a parameter type in an interface description is only to document an expected type in VARIANT structure.

ISCSession	
Method	Description
	passes back VT_EMPTY. Please note that transaction nesting is implicit. If a SCAPI client invokes BeginTransaction and a transaction is already open, the new transaction is nested inside the existing one.
Boolean CommitTransaction(VARIANT TransactionId)	Commits the specified transaction and all nested transactions contained within it. Returns TRUE if successful, and FALSE if the model provider does not support transactions, or if <i>TransactionId</i> contains an invalid value.
Boolean RollbackTransaction(VARIANT TransactionId)	Rolls back the specified transaction and all nested transactions contained within it. Returns TRUE if successful, and FALSE if the model provider does not support transactions, or if <i>TransactionId</i> contains an invalid value.
Boolean Undo()	Undoes the last operation. Notifies the callers of all resulting changes. Passes <i>Cookie</i> as part of the notification. Returns TRUE if successful and FALSE if self is closed. Model providers that do not support the undo operation returns FALSE as well.
Boolean IsValid()	Returns TRUE if self is valid (i.e., occupies the Valid state), and FALSE otherwise. This method lets the caller detect when the referenced object is deleted.

The following table describes result codes generated by methods and properties of the interface.

ISCSession		
Error Code	Methods/Properties	Description
SCAPI_E_SS_NOTINCOLLECTION	All	Attempt to access a session object that was removed from the collection of sessions.

Model Tier

The Model tier provides a uniform way to manipulate application data independently of the application's physical data organization. It presents the data in object/property form, and provides methods that enable the API clients to navigate through a model and manipulate its contents. Navigation is consistent with the approach used by most automated-based applications and follows industry-accepted concepts of collections, enumerating, and accessing elements in the collections.

Model Objects Collections

The Model Objects Collection is a collection is a group of zero or more model objects. Access to members of the collection is provided by enumerating and by direct id. The collection has a root object that “contains” the rest of objects in respect of their enumeration. For instance, a persistence unit represented by a root model object, such as **Model** in ERwin.

The default rule for enumerating is: depth first. The root object doesn’t participate in the enumeration.

The root object is used as a context object by Add method to add a new object to the collection.

The API clients access specific model object via ISCMModelObject interfaces while the ISCMModelObjectCollection interface serves collections of model objects. The collection of all model objects associated with a session are available from ISCSession interface via ModelObjects method.

The following table describes the ISCMModelObjectCollection interface properties:

ISCMModelObjectCollection			
Property	Type	Read Only	Description
Count	long	Yes	Returns the number of model objects in the collection. The number does not include the root object. Returns 0 if the collection has no members and –1 if the member count cannot be determined. To improve performance, the collection count could be maintained as a “high water mark” — the highest-numbered member yet seen as the user moves through the collection.
Root	ISCMModelObject *	Yes	Returns a pointer to the root object in a collection. If the collection doesn’t have any root object, then NULL pointer will be returned.
Item(VARIANT Index, [optional] (opt) VARIANT ClassId) (Def.)	IUnknown *	Yes	Returns an ISCMModelObject interface pointer identified by the <i>Index</i> and optional <i>ClassId</i> parameters. The <i>Index</i> is one of the following: • ObjectId of requested model object. • A string identifier, returned by Name property of ISCMModelObject interface and followed by the object <i>ClassId</i>. • A ISCMModelObject pointer. In such case it will be cloned. The <i>ClassId</i> is either a string with class id code or a string

(opt) Due to support automation client requirements, all optional parameters are represented as VARIANT. For that reason, a parameter type in an interface description is only to document an expected type in VARIANT structure.

(Def.) Default Property on an object. For the automation, a default property is the property that is accessed when the object is referred to without any explicit property or method call. The dispatch index for the property is DISPID_VALUE.

ISCMModelObjectCollection			
Property	Type	Read Only	Description
			with it name. The persistence unit associated with the session always limits the search. For ObjectId scenario, search is not limited by the collection. For an object name and class, to be effective, the search is executed inside of the collection. Returns Null if not succeeded. Returns a pointer to self if the arguments are omitted.
_NewEnum	IUnknown*	Yes	Hidden property. Construct an instance of model objects enumerator and returns IEnumVariant interface pointer of the enumerator. Every call to the Next method of IEnumVariant will return one or more Variant elements with ISCMModelObject interface pointers, or NULL if no more model objects are available. All enumerations are depth-first. The direct children of a root object are a starting point for collection enumerating, subject to filtering requirements.
ClassIds	SAFEARRAY(SC_CLSID)	Yes	Represents a value of <i>Model Objects</i> collection attribute that limited the membership in the collection at the time when this collection was created and can be used for reference purposes. <i>ClassIds</i> contains a list of acceptable class identifiers (i.e., object types). If this list is non-empty, the collection includes only those objects whose class identifier appears in the list. If the list is empty or returns a NULL pointer, then all objects included.
ClassNames	SAFEARRAY(BSTR)	Yes	Similar to <i>ClassIds</i> with exception that class names are presented instead of class ids.
Depth	long	Yes	Represents a value of <i>Model Objects</i> collection attribute that limited the membership in the collection at the time when this collection was created and can be used for reference purposes. Depth contains the maximum depth to which the iterator will descend, with 0 representing unlimited depth.
MustBeOn	long	Yes	Represents a value of <i>Model Objects</i> collection attribute that limited the membership in the collection at the time when this collection was created and can be used for reference purposes. MustBeOn returns a “Must Be On” flag filter. For a model object to be accepted as a member, all flag bits that match a ‘1’ bit in MustBeOn must be ‘1’ themselves. (Another way to state this condition is <code>Flags & MustBeOn == MustBeOn</code>.) MustBeOn has no effect if it contains 0.

ISCMModelObjectCollection			
Property	Type	Read Only	Description
MustBeOff	long	Yes	Represents a value of <i>Model Objects</i> collection attribute that limited the membership in the collection at the time when this collection was created and can be used for reference purposes. MustBeOff returns a “Must Be Off” flag filter. For a model object to be accepted as a member, all flag bits that match a ‘1’ bit in MustBeOff must be ‘0’. (Another way to state this condition is $\sim\text{Flags} \ \& \ \text{MustBeOff} == \text{MustBeOff}$). MustBeOff has no effect if it contains 0.

The following table describes the ISCMModelObjectCollection interface methods:

ISCMModelObjectCollection	
Method	Description
ISCMModelObjectCollection * Collect (VARIANT Root, [optional]VARIANT ClassIDs, [optional]long Depth, [optional]long MustBeOn, [optional]long MustBeOff)	Creates a <i>Model Objects</i> collection, which represents a subcollection of itself. All filtering criteria specified in the Collect call will be applied toward membership in the collection. The method always creates a valid collection even though the collection could be empty. If the filtering requirements are too rigid, the resulting collection will be empty. A collection held as empty if there is no object in the collection, including a root object. The <i>Root</i> parameter specifies a root object in the collection. This object used as a context when a new object added to the collection with Add method of the interface. <i>Root</i> is either an object ObjectId or a valid ISCMModelObject pointer. The root candidate does not have to belong to the collection - source, but it has to belong to the persistence unit associated with the session. <i>ClassIDs</i> contains a list of acceptable class identifiers (i.e., object types) or class names. If this list is non-empty, the collection will include only those objects whose class identifier appears in the list. If the list is empty or returns a NULL pointer, then all objects included. The list is one of the following: • SAFEARRAY with either SC_CLSID or class names; • A string with ids or names separated with semicolons; All enumerations are depth-first. The direct children of a root object are a starting point for collection enumerating, subjects to filtering requirements. <i>Depth</i> contains the maximum depth to which the iterator should descend, with -1 (default value) representing unlimited depth. <i>MustBeOn</i> holds a “Must Be On” flag filter. For a model object to pass, all flag bits that match a ‘1’ bit in <i>MustBeOn</i> must be ‘1’ themselves. (Another way to state this condition is $\text{Flags} \ \& \ \text{MustBeOn} ==$

ISCModelObjectCollection	
Method	Description
	MustBeOn.) <i>MustBeOn</i> by default contains zero and has no effect. <i>MustBeOff</i> holds a “Must Be Off” flag filter. For a model object to pass, all flag bits that match a ‘1’ bit in <i>MustBeOff</i> must be ‘0’. (Another way to state this condition is $\sim\text{Flags} \& \text{MustBeOff} == \text{MustBeOff}$). <i>MustBeOff</i> by default contains zero and has no effect. Note that no objects can pass the flag filter if <i>MustBeOn</i> & <i>MustBeOff</i> yields a nonzero result. The method returns E_INVALIDARG this happens.
ISCModelObject* Add(VARIANT Class, [optional] SC_OBJID ObjectId)	Adds an object of type Class to the model. Class is the class identifier given in the meta-model specification. A class ID or a class name could be used. Passes back NULL and returns an error code if no object is created If ObjectId contains a non-NULL object identifier, the new object’s identifier assumes the value passed on ObjectId. If ObjectId contains a NULL value, the application assigns the new object’s identifier. If an object exists having ObjectId as its identifier, the method will fail and return an error.
Boolean Remove(VARIANT Object)	Removes the specified model object from a model. Returns TRUE if successful and FALSE if <i>Object</i> contains an invalid value. The <i>Object</i> parameter provided: • By ID – an ObjectID of a target object • By an interface pointer – ISCModelObject pointer of the <i>Model Object</i> component that bound to the model object. Note: successful execution of the call renders all binds with the removed object invalid. The client should release all ISCModelObject pointers known to represent such association. Calls to ISCModelObject will fail and IsValid method returns FALSE. Recovering the model object with Undo does not guarantee re-binding exist <i>Model Object</i> back to the object.

Model Object

A Model Object component represents a single object in a model. An ISCModelObject interface pointer is available in number ways from ISCModelObjectCollection interface as well as via Root property of ISCSession interface.

The following table describes ISCModelObject interface properties:

ISCModelObject			
Property	Type	Read Only	Description
ObjectId	SC_OBJID	Yes	Uniquely identifies the current object. An object identifier

ISCModelObject			
Property	Type	Read Only	Description
			must be unique application-wide.
Name	BSTR	Yes	An object name or a string identifier. The string identifier is for display purposes primarily and not necessarily represents a value of any particular property of the modal object.
Class	SC_CLSID	Yes	Returns the current object's class identifier.
ClassName ^(Def.)	BSTR	Yes	Returns the current object's class name. This is provided for display only.
Context	ISCModelObject *	Yes	Passes back the object's context – the object immediately above the current object in the model's object tree. Passes NULL back if the current object is the tree root.
Flags	long	Yes	A 32 bit object flag word. The following low-order bits are currently defined: 0 Object is a persistence unit if set / SCD_MOF_PERSISTENCE_UNIT 1 Object is user-defined if set / SCD_MOF_USER_DEFINED 2 Object is the root object if set / SCD_MOF_ROOT 3 Object is maintained by the tool, if set / SCD_MOF_TOOL 4 Object is created by the tool and not removable / SCD_MOF_DEFAULT 5 Object is new or updated in a transaction and the transaction has not committed. SCD_MOF_TRANSACTION All other bits are set to zero. Zero / SCD_MOF_DONT_CARE returned when no flags are set
Properties	ISCModelPropertyCollection*	Yes	Returns an interface pointer for the property collection. The collection will include all available properties of the object. The method always returns a valid collection even though the collection could be empty. If the filtering required, CollectProperties method should be used instead.

The following table describes ISCModelObject interface methods:

^(Def.) Default Property on an object. For the automation, a default property is the property that is accessed when the object is referred to without any explicit property or method call. The dispatch index for the property is DISPID_VALUE.

ISCModelObject	
Method	Description
Boolean IsInstanceOf(VARIANT ClassId)	Returns TRUE if self is an instance of the passed class and FALSE if it does not. This method respects inheritance. If <i>ClassId</i> contains an ancestor class, the method must return TRUE. <i>ClassId</i> could be either a class SC_CLSID or a class name.
Boolean IsValid()	Returns TRUE if self is valid (i.e., occupies the Valid state), and FALSE otherwise. This method lets the caller detect when the referenced object is deleted.
ISCModelPropertyCollection* CollectProperties([optional] ^(opt) VARIANT ClassIds, [optional] long MustBeOn, [optional] long MustBeOff)	Returns an interface pointer for property collection of the desired type. The method always returns a valid collection even though the collection could be empty. If the filtering requirements are too rigid, the resulting collection will be empty. <i>ClassIds</i> contains a list of acceptable class identifiers (i.e., property types) or class identifier names. If this list is non-empty, the property collection includes only those properties whose class identifier appears on the list. If the list is empty or the caller supplies a NULL pointer, the collection includes all properties owned by the object. The list is one of the following: • SAFEARRAY with either SC_CLSID or class names; • A string with ids or names separated with semicolons; <i>MustBeOn</i> holds a “Must Be On” flag filter. For a property to pass, all flag bits that match a ‘1’ bit in <i>MustBeOn</i> must be ‘1’ themselves. (Another way to state this condition is $\text{Flags} \& \text{MustBeOn} == \text{MustBeOn}$.) <i>MustBeOn</i> has no effect if it contains 0. <i>MustBeOn</i> is zero by default. <i>MustBeOff</i> holds a “Must Be Off” flag filter. For a property to pass, all flag bits that match a ‘1’ bit in <i>MustBeOff</i> must be ‘0’. (Another way to state this condition is $\sim \text{Flags} \& \text{MustBeOff} == \text{MustBeOff}$.) <i>MustBeOff</i> has no effect if it contains 0. <i>MustBeOff</i> has SCD_MPF_NULL set by default. For ERwin it excludes properties with no values from the collection. Note that no properties can pass the flag filter if <i>MustBeOn</i> & <i>MustBeOff</i> yields a nonzero result.

^(opt) Due to support automation client requirements, all optional parameters are represented as VARIANT. For that reason, a parameter type in an interface description is only to document an expected type in VARIANT structure.

Model Properties collection

The Model Properties collection groups zero or more properties associated with a model object and provides methods to access members of the collection by enumerating and by property id. The API clients access specific properties via ISCMModelProperty interfaces while ISCMModelPropertyCollection interface serves property collections. ISCMModelPropertyCollection interface pointer is available via Properties property of ISCMModelObject interface.

Membership of the collection was specified by the ISCMModelObject Properties call as a result of which the collection was created.

The following table describes the ISCMModelPropertyCollection interface properties:

ISCMModelPropertyCollection			
Property	Type	Read Only	Description
Count	Long	Yes	Returns the number of properties that was accepted to the collection. The number could be a top mark number if class and/or flags filters are on.
Item(VARIANT Class) (Def.)	IUnknown *	Yes	Returns an ISCMModelProperty interface pointer identified by <i>Class</i> parameter as: • A property ID; an SC_CLSID of a property • A class name of a property • An ISCMModelProperty pointer to clone an object The method checks if property exists. If it does not, the method creates a property description, returns ISCMModelProperty instance and set NULL flag for the property. A new property value can be set by using <i>Value</i> property of the instance. However, it will fail to retrieve a value before it was set. The method allows you to create an instance of ISCMModelProperty for properties like ReadOnly, Maintained By the Tool, etc. The value for these properties cannot be changed or assigned. Yet, property flags, datatype, etc. are available even in a case when the collection does not have the property instance. Use HasProperty to check on the existence of the property for a model object instance. Returns a pointer to self if the arguments are omitted.
_NewEnum	IUnknown*	Yes	Hidden property. Construct an instance of properties enumerator and returns IEnumVariant interface pointer of the enumerator.

(Def.) Default Property on an object. For the automation, a default property is the property that is accessed when the object is referred to without any explicit property or method call. The dispatch index for the property is DISPID_VALUE.

ISCMModelPropertyCollection			
Property	Type	Read Only	Description
			Every call to the Next method of IEnumVariant will return one or more Variant elements with ISCMModelProperty interface pointers, or NULL if no more properties are available.
ClassIds	SAFEARRAY(SC_CLSID)	Yes	Represents a value of <i>ModelProperties</i> collection attribute that limited the membership at the time when this collection was created and can be used for reference purposes. <i>ClassIds</i> contains an array of acceptable class identifiers (i.e., property types). If this list is non-empty, the property collection includes only those properties whose class identifier appears on the list. If the list is empty or the caller supplies a NULL pointer, the collection includes all properties owned by the object.
ClassNames	SAFEARRAY(BSTR)	Yes	Same as <i>ClassIds</i> property, but holds property class names.
MustBeOn	Long	Yes	Represents a value of <i>ModelProperties</i> collection attribute that limited the membership at the time when this collection was created and can be used for reference purposes. MustBeOn returns a “Must Be On” flag filter. For a property to pass, all flag bits that match a ‘1’ bit in MustBeOn must be ‘1’ themselves. (Another way to state this condition is <code>Flags & MustBeOn == MustBeOn</code> .) MustBeOn has no effect if it contains 0.
MustBeOff	Long	Yes	Represents a value of <i>ModelProperties</i> collection attribute that limited the membership at the time when this collection was created and can be used for reference purposes. MustBeOff returns a “Must Be Off” flag filter. For a property to pass, all flag bits that match a ‘1’ bit in MustBeOff must be ‘0’. (Another way to state this condition is <code>~Flags & MustBeOff == MustBeOff</code> .) MustBeOff has no effect if it contains 0.

The following table describes ISCMModelPropertyCollection interface methods:

ISCMModelPropertyCollection	
Method	Description
Boolean HasProperty(VARIANT Class)	Returns TRUE if the object bound to self owns a property of the passed class, and FALSE otherwise. Treats properties as absent if fail to satisfy <i>ClassIds</i> , <i>MustBeOn</i> and <i>MustBeOff</i> attributes of the collection. Alternative <i>MustBeOn</i> , <i>MustBeOff</i> could be offered via optional parameters.

ISCMModelPropertyCollection	
Method	Description
	Returns an error if the property does not belong to this type of model objects. The <i>Class</i> parameter supports identifying a property: - By a property ID, SC_CLSID - By class name of the property.
ISCMModelProperty* Add(VARIANT Class)	Construct a new property for a bound model object if it does not exist. Returns ISCMModelProperty interface pointer to represent a property. The <i>Class</i> parameter supports identifying a property: - By a property ID; an SC_CLSID of a target property - By class name of the target property.
BOOL Remove (VARIANT Class, [optional] long MustBeOn, [optional] long MustBeOff)	Removes the indicated property from the bound object. Returns TRUE if the property could be removed, and FALSE if the property does not exist. This method always treats properties with the SCD_MPF_NULL flag as existing. The <i>Class</i> parameter supports identifying a property: - By a property ID; an SC_CLSID of a target property - By class name of the target property. If the property is not removable, for instance for ERwin, only user-defined properties could be removed, the property acquires the SCD_MPF_NULL flag. Note: successful execution of the call renders all binds with the removed property invalid. The client should release all ISCMModelProperty pointers, all related <i>Value Collection</i> and <i>Value</i> pointers known to represent such association. Calls to interfaces will fail and IsValid method returns FALSE. Recovering the model property with Undo does not guarantee re-binding exist interfaces back to the object.

Model Property

A *Model Property* represents a single property on a given object. Properties come in two varieties, single valued and multi-valued. A single-valued property must contain a single value, while a multi-valued property may contain multiple values. Note, however, that a multi-valued property may happen to contain a single value. If a property contains a single value, it may still be a multi-valued property.

Currently, ERwin supports scalar and simple homogeneously typed array-like properties. The API, however, is designed to support complex property implementations, such as hash dictionaries (i.e., a list containing keys and associated values), as well as heterogeneously typed properties. Hence, the API allows clients to use arbitrary identifiers to select values from vector properties. Initially, ERwin uses a simple integer index to navigate vector properties. In the future, strings could be used to select property values from a hashed dictionary.

Values and value identifiers are atomic – i.e., they have no internal structure. While we hold their values in VARIANT structures, we restrict their actual value types to the scalars that the VARIANT structure can hold plus GUIDs and the IDs of referenced objects.

The following table explains how the SCAPI value type codes maps to VARIANT types.

Description	Value Typecode	COM Type ¹	Comment/ SCAPI Type
Missing value (provided for completeness)	0	VT_NULL	SCVT_NULL
Signed 16-bit integer	1	VT_I2	SCVT_I2
Signed 32-bit integer	2	VT_I4	SCVT_I4
Unsigned 8-bit integer. Do not use this type to hold character data.	3	VT_UI1	SCVT_UI1
4-byte floating point real	4	VT_R4	SCVT_R4
8-byte double precision real	5	VT_R8	SCVT_R8
Boolean value (TRUE or FALSE)	6	VT_BOOL	SCVT_BOOLEAN
64-bit Currency Value	7	VT_CY	SCVT_CURRENCY
IUnknown * interface pointer (provided for completeness)	8	VT_UNKNOWN	SCVT_IUNKNOWN
IDispatch * interface pointer (provided for completeness)	9	VT_DISPATCH	SCVT_IDISPATCH
Date value in VARIANTDATE format	10	VT_DATE	SCVT_DATE
Basic (or Binary) String. All string information (including single characters) must be passed as a BSTR.	11	VT_BSTR	SCVT_BSTR
Unsigned 16-bit integer	12	VT_UI2	SCVT_UI2
Unsigned 32-bit integer	13	VT_UI4	SCVT_UI4
GUID, returned as a string in {...} format. See Object Identifiers above.	14	VT_BSTR	SCVT_GUID
Object identifier, returned as a string in {...}+off format. See Object Identifiers above.	15	VT_BSTR	SCVT_OBJID
SAFEARRAY of unsigned BYTES	16	VT_ARRAY & array element type (VT_UI1 for instance)	SCVT_BLOB
The default value type.	17	Varies	SCVT_DEFAULT
Signed 1-byte integer. Do not use this type to hold character data.	18	VT_I1	SCVT_I1

¹ As saved in a VARIANT structure. Users concerned about standard marshalling should refer to objidl.idl, lines 2198 ff. In addition, oaidl.idl defines the wireVARIANT type, which can marshal members of a VARIANT union, together with macros that automatically apply wireVARIANT marshalling to VARIANT structures. SCAPI.idl includes both objidl.idl and oaidl.idl.

Description	Value Typecode	COM Type ¹	Comment/ SCAPI Type
Machine-dependent signed integer. Please use with caution.	19	VT_INT	SCVT_INT
Machine-dependent unsigned integer. Please use with caution.	20	VT_UINT	SCVT_UINT
Rectangle property. An array of four integers	21	VT_ARRAY & VT_I2	SCVT_RECT
Point property. An array of two integers	22	VT_ARRAY & VT_I2	SCVT_POINT

ISCMModelProperty interface pointer could be obtained via methods in ISCMModelPropertyCollection.

The following table describes ISCMModelProperty interface properties:

ISCMModelProperty			
Property	Type	Read Only	Description
Class	SC_CLSID	Yes	Property class identifier. The class identifier uniquely identifies a property instance within the scope of its owning object.
ClassName ^(Def.)	BSTR	Yes	Property class name.
Count	long	Yes	Contains the number of values in the property. Scalar properties always have 1 value.
Flags	long	Yes	A 32 bit property flag word. The following low-order bits are currently defined: 0 Property has a NULL value or no value if set / SCD_MPF_NULL 1 Property is user-defined if set. / SCD_MPF_USER_DEFINED 2 Property is scalar if set. / SCD_MPF_SCALAR 3 Property is maintained by the tool, if set. / SCD_MPF_TOOL 4 Property is read-only if set. / SCD_MPF_READ_ONLY 5 Property has inherited/calculated/derived value if set. / SCD_MPF_DERIVED 6 Property is optional and could be removed / SCD_MPF_OPTIONAL All other low-order bits are zero. Zero / SCD_MPF_DONT_CARE is returned if no flags are set.

^(Def.) Default Property on an object. For the automation, a default property is the property that is accessed when the object is referred to without any explicit property or method call. The dispatch index for the property is DISPID_VALUE.

ISCModelProperty			
Property	Type	Read Only	Description
PropertyValues	ISCProperty ValueCollection *	Yes	Allocates the desired <i>Values</i> collection. The returned collection hosts every value in the property. The collection targeting value type conversions. Value type conversions for scalar properties supported via a collection that holds the single property value.
Value([optional] ^(opt) VARIANT ValueId, [optional] long ValueType)	VARIANT		Retrieves, sets or changes the indicated property value in requested format. If self contains a single-valued property, the caller may safely pass a VT_EMPTY <i>ValueId</i> parameter or skip it; it is ignored if supplied. If self contains a multi-valued property, the caller should supply the <i>ValueId</i> parameter. If the caller omits the <i>ValueId</i> argument for a multi-valued property, the method's behavior is unspecified. If a new value is assigned and element already exists for the passed <i>ValueId</i>, it is changed, otherwise it is added. ERwin supports multi-value properties in form of vector of elements. All elements have identical datatype. <i>ValueId</i> is numeric in case of ERwin. A value -1 for <i>ValueId</i> will cause a new value to be added at the end of the vector. <i>ValueId</i> that is equivalent to the current size of the vector will also have the same effect as -1. An optional, the requested type could be indicated by <i>ValueType</i> and must be supported (note that the default value type is always supported) or the method will fail.
DataType([optional] VARIANT ValueId)	long	Yes	Passes back the identifier of the default value type for the indicated property value. If self contains a single-valued property (i.e., the property has a simple data type), the caller may safely pass a VT_EMPTY <i>ValueId</i> parameter or skip it; it is ignored if supplied. If self contains a multi-valued property, the caller should supply the <i>ValueId</i> parameter. If the caller omits the <i>ValueId</i> argument for a multi-valued property, the method's behavior is unspecified.

The following table describes ISCModelProperty interface methods:

ISCModelProperty	
Method	Description
BSTR FormatAsString()	Formats the property value as a string. The returned string represents the value

^(opt) Due to support automation client requirements, all optional parameters are represented as VARIANT. For that reason, a parameter type in an interface description is only to document an expected type in VARIANT structure.

ISCModelProperty	
Method	Description
	of the <u>entire</u> property, even if the property contains multiple values.
Boolean RemoveValue([optional] VARIANT ValueId)	Removes the specified value from the property. If no values remain after the removal, the property will have NULL value. Returns TRUE if the value was removed, FALSE if the application rejected the change, and will raise an error if <i>ValueId</i> contains an invalid value. If self contains a simple-valued property, the caller may safely pass a VT_EMPTY <i>ValueId</i> parameter or omit it; it is ignored if supplied. If self contains a multi-valued property, the caller should supply the <i>ValueId</i> parameter. If the caller omits the <i>ValueId</i> argument for a multi-valued property, the method's behavior is unspecified.
Boolean RemoveAllValues()	Remove all values from the property. The property will have NULL value then.
Boolean IsValid()	Returns TRUE if self is valid (i.e., is in the Valid state, hence is bound to a valid property), and FALSE otherwise. This method allows the caller to detect property deletion.

Property Values collection

The Property Values collection targets value type conversions and holds all values for a selected model object property. The collection of values is available from *Model Property* via *PropertyValues* property of *ISCModelProperty* interface.

The following table describes the *ISCValueCollection* interface properties:

ISCValueCollection			
Property	Type	Read Only	Description
Count	LONG	Yes	Returns the number of values in the property
Item(VARIANT ValueId) ^(Def.)	IUnknown *	Yes	Returns an <i>ISCPROPERTYVALUE</i> interface pointer identified by the <i>ObjectId</i> parameter. Returns a pointer to self if the arguments are omitted.
_NewEnum	IUnknown*	Yes	Hidden property. Construct an instance of properties enumerator object and returns <i>IEnumVariant</i> interface pointer of the enumerator.

^(Def.) Default Property on an object. For the automation, a default property is the property that is accessed when the object is referred to without any explicit property or method call. The dispatch index for the property is *DISPID_VALUE*.

ISCValueCollection			
Property	Type	Read Only	Description
			Every call to the Next method of IEnumVariant will return one or more Variant elements with ISCValue interface pointers, or NULL if no more objects are available.

Property Value Component

The Property Value component represents a single value of a given property. An ISCPROPERTYVALUE interface pointer is available in number ways from ISCPROPERTYVALUECOLLECTION interface.

The following table describes ISCPROPERTYVALUE interface properties:

ISCPROPERTYVALUE			
Property	Type	Read Only	Description
ValueId ^(Def.) ([optional] ^(opt) long ValueType)	VARIANT	Yes	Converts the current value identifier to the passed format and passes back the converted identifier. The requested value data type (if supplied) must be supported for the current value identifier (note that the default value type is always supported).
PropertyClassId	SC_CLSID	Yes	Identifies the current property's context.
PropertyClassName	BSTR	Yes	Returns the current property's class name. This is provided for display only.
Value([optional] long ValueType)	VARIANT	Yes	Converts the current value to the passed value type and returns it. The requested data type must be supported (note that the default value type is always supported). Uses a default value type if <i>ValueType</i> is skipped. Passes back the converted value if successful.
ValueType	long	Yes	Passes back the identifier of the value default type.
ValueIdType	long	Yes	Passes back the identifier of the value identifier default type.

^(Def.) Default Property on an object. For the automation, a default property is the property that is accessed when the object is referred to without any explicit property or method call. The dispatch index for the property is DISPID_VALUE.

^(opt) Due to support automation client requirements, all optional parameters are represented as VARIANT. For that reason, a parameter type in an interface description is only to document an expected type in VARIANT structure.

The following table describes the ISCPROPERTYVALUE object methods:

ISCPROPERTYVALUE	
Method	Description
SAFEARRAY(long) GetSupportedValueTypes()	Groups a list of supported value types and return it as a SAFEARRAY. The GetValue method must be able to convert the current value into any value type whose code appears in the returned list. If the list is empty, the value is available only in its native (i.e., default) format. Reference properties should return an empty list.
SAFEARRAY(long) GetSupportedIdTypes()	Groups a list of supported value types for the current value identifier and return it as a SAFEARRAY. The GetValueId method must be able to convert the current value into any value type whose code appears in the returned list. If the list is empty, then the current identifier is available only in its native (i.e., default) format.

Index

A

Active Scripting, 5, 6, 16, 18

C

Components

- Application, 8, 17, 18, 19, 20
- Application Environment, 18
- Application Services, 20
- Model Object, 31, 33, 34, 35, 36
- Model Objects, 31, 33, 34, 35
- Model Properties, 39
- Model Property, 26, 41, 45
- Persistence Unit, 21, 24, 25, 27, 31
- Persistence Units, 21, 24, 25, 27
- Property Bag, 25, 27
- Property Value, 45, 46
- Property Values, 45
- Session, 28, 29, 30
- Sessions, 8, 18, 28, 29, 30
- Value, 41, 42, 44, 46
- Values, 9, 12, 42, 44, 45

I

Interfaces

- IEnumVARIANT, 13, 14
- IEnumVARIANT, 13, 14
- IPtSCElementValue, 45, 46
- IPtSCElementValueCollection, 45, 47
- IPtSCIDList, 20
- IPtSCModelDirectory, 18
- IPtSCModelDirectory, 18
- IPtSCModelEnterprise, 17, 18, 20
- IPtSCModelObject, 33, 34, 36, 37, 38, 39
- IPtSCModelObjectCollection, 31, 33, 35, 36
- IPtSCModelProperty, 39, 41, 43, 44, 45
- IPtSCModelPropertyCollection, 37, 39, 40, 43
- IPtSCModelProvider, 21, 28
- IPtSCPDElement, 18, 19
- IPtSCPersistenceUnit, 21, 24, 25, 26, 27, 31
- IPtSCPersistenceUnitCollection, 18, 21, 22
- IPtSCPropertyValue, 45, 46, 47
- IPtSCPropertyValueCollection, 45, 46
- IPtSCProvider, 18, 19, 22, 28, 34, 39, 45
- IPtSCSession, 28, 29, 30, 31, 36
- IPtSCSessionCollection, 18, 28, 29
- ISCAApplication, 8, 17, 18, 20, 21, 28
- ISCAApplicationEnvironment, 8, 18, 19
- ISCAApplicationService, 8, 20
- ISCAApplicationServiceCollection, 8, 20

- ISCMDElement, 8, 22
- ISCModelDirectory, 8, 12, 18
- ISCModelObject, 9, 33, 34, 35, 36, 37, 38, 39
- ISCModelObjectCollection, 9, 31, 33, 35, 36
- ISCModelProperty, 9, 39, 41, 43, 44, 45
- ISCModelPropertyCollection, 37, 39, 40, 43
- ISCPersistenceUnit, 8, 21, 22, 23, 24, 25, 26, 27, 31
- ISCPersistenceUnitCollection, 8, 18, 21, 22, 24
- ISCPropertyValue, 45, 46, 47
- ISCPropertyValueCollection, 46
- ISCSession, 8, 12, 28, 29, 30, 31, 32, 33, 36
- ISCSessionCollection, 8, 18, 28, 29
- ISCValue, 9, 45, 46, 47
- ISCValueCollection, 9, 44, 45, 46

M

- Model Object
 - flags, 37

O

Objects

- ElementValue, 46
- ElementValues, 44, 45
- ModelEnterprise, 17, 18
- ModelObject, 36
- ModelObjects, 31, 35
- ModelProperties, 37, 38, 39
- ModelProperty, 41, 45
- PDElement, 18
- PersistenceUnit, 24, 25, 31
- PersistenceUnits, 18, 21, 24, 29
- PropertyValue, 46
- PropertyValues, 44, 45
- Provider, 18
- Session, 28, 29, 30, 33
- Sessions, 18, 28, 30

P

- Persistence Unit, 21, 24, 25, 27, 31
- Property of model object
 - flags, 43
 - multi-valued, 9, 41, 44, 45
 - single-valued, 41, 44
 - value type code, 26, 42

S

- Script, 1, 5, 6

Session, 8, 18, 28, 29, 30, 33