Interaction Designer COM API Help

Printed Documentation

PureConnect powered by Customer Interaction Center® (CIC)

2017 R4

Last updated August 08,2017
(See Change Log for summary of changes.)

Table of Contents

Designer COM API Reference

Introduction

Objects exposed by Designer COM API

In Interaction Designer, handler developers lay out logic that responds to a given event by arranging and connecting on-screen graphical objects. Each object performs a specific task, such as sending a fax, email, or routing a call. These event-processing programs are called *handlers*.

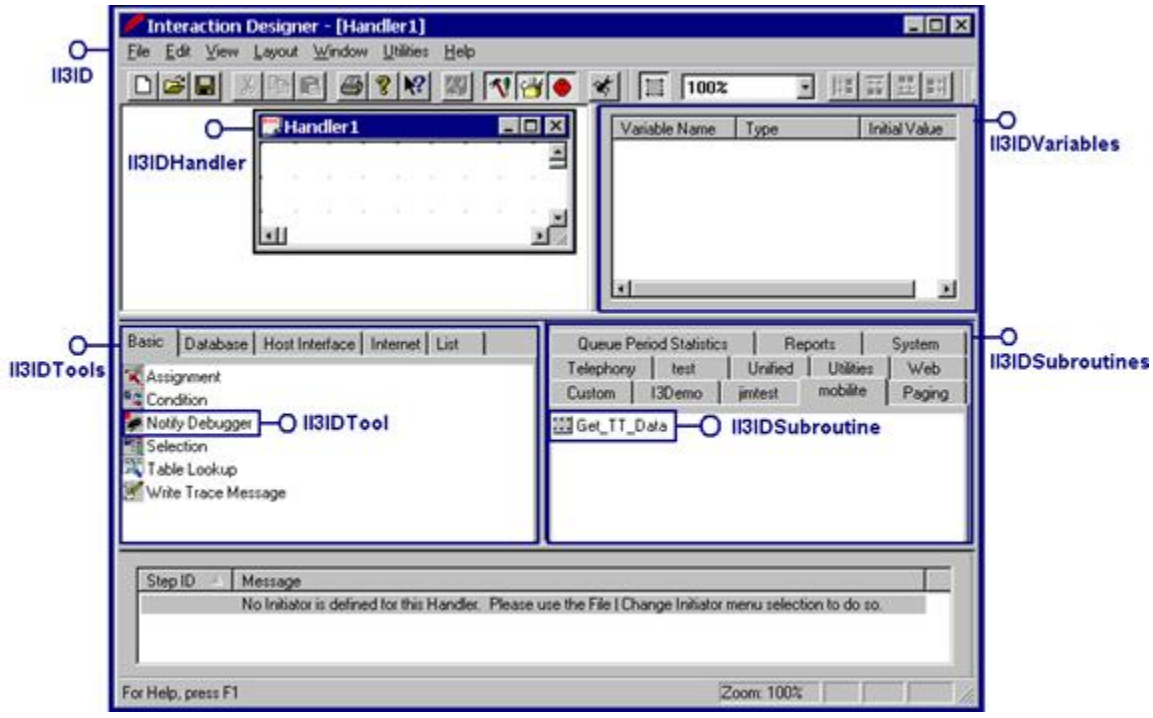
Handlers encapsulate program logic within graphically connected nodes, called *steps*, for events processed by the IC server. Each step is created by dragging a *tool* (a template definition of each step) from a palette, pasting it in the workspace, and editing the tool's properties. Interaction Designer provides an extensible palette of tools that serve as building blocks for building handlers.

The first step in each handler is an *initiator*—it identifies the event that starts the handler. An example of an event might be 'Incoming Call'. When Interaction Processor receives a notification that an 'Incoming Call' event has occurred, it starts the handler whose initiator is configured for 'Incoming Call'.

Interaction Designer *publishes* handler logic. This process converts visually designed logic to Java program code that is compiled to create a highly efficient server application.

The *Interaction Designer COM API* gives developers programmatic control over the familiar visual objects exposed by Interaction Designer. Standard object-oriented programming techniques are used to manipulate the methods and properties of non-visual API objects (e.g. handlers, tools, exit paths, etc.)

Most non-visual objects (supported by the API) are analogs of familiar visual objects in Interaction Designer. For example, a handler object in the API provides access to a collection of step objects. The image below shows the correlation between visual elements in Interaction Designer and objects exposed by the API:



Each step has properties such as the step name, position, and exit paths that can be managed programmatically. Likewise, you can call methods that perform actions on Designer objects. For example, you can publish a handler by calling the Publish method of a handler object, or open a handler by calling the OpenHandler method of the Designer object.

Other objects represent steps, links between steps, and tool step exit paths.

Interaction Designer must be running when Designer COM applications are executed.

This API provides a comprehensive set of classes that wrapper tools, handlers, steps, messages, subroutines, initiators, exit paths, and properties of Designer itself. New objects (handlers, steps, etc.) can be created and published under program control, and existing objects can be enumerated and modified on the fly.

For detailed information about the properties and methods supported by each interface, refer to the Interfaces section of this document.

COM Interface Information

Naming conventions in this documentation

This documentation describes COM components in a structured way. Hyperlinks make it easy to navigate and drill down, without becoming "lost in hyperspace."

Interface list

At the topmost level, interfaces are described in an interface list. You can drill down to specific interface pages by clicking on interface name hyperlinks. The description of each interface is followed by a list of properties that the interface supports (if any), and by an alphabetical list of methods. To make it easier to identify the interface associated with each method, interface and method names are appended together in topic titles, using two colons as a delimiter.

Syntax

```
InterfaceName::MethodName
```

Example

```
IEICCallObject::Dial
```

Property and Method pages

Each interface page contains a list of **properties** supported by the interface. This list is followed by a list of **methods** that the interface supports. You can drill down to a specific *property page* or *method page* by clicking on hyperlinks.

Each property or method page is divided into sections:

- A *Synopsis* statement describes each property or method.
- *Function Prototypes* describe methods and properties in a generic way, independent of specific language syntax. Those of you who are familiar with IDL (Interface Definition Language) will immediately recognize the format used here. Function prototypes specify the data type of each argument, and indicate whether arguments are optional, supplied as an input variable, or return value.
- *GET vs. PUT*: property pages are divided into two sections that discuss GETs and PUTs separately. Separate function prototypes, calling syntax, and examples are provided for get (read) and put (write) operations. Methods have only a single function prototype.
- *Calling Syntax*: example syntax is provided for C++. In some cases, the C++ example is followed by a list of C++ Return values that describe constants associated with a particular method or property.
- Parameter List: parameters are described individually.

About GUIDs, UUIDs, and CLSIDs

COM objects are assigned a 16-byte ID number, so that the operating system can identify the application that manages the object. (By application, we mean a COM DLL or COM executable file.) This 16-byte value assigned to objects is commonly called a GUID (Globally Unique Identifier), UUID (Universally Unique Identifier), or CLSID (Class Identifier).

In this documentation, the GUID for each interface appears before its list of methods and properties.

Regardless of the terminology used (GUID, UUID, or CLSID), the ID number is used by the operating system to identify each object, irrespective of other names used by programmers. If two programmers inadvertently assign the same name to an interface or method, the system can determine which application to call by looking at the object's ID number.

COM Clients and Servers

COM is a refinement of the thinking behind OLE, DDE, and ActiveX. COM makes it possible to create truly reusable software components that are compatible across programming languages. COM provides some relief from version conflicts associated with software updates, and creates a platform for truly distributed network computing. As with DDE, there are COM *clients* and COM *servers*. However, in the world of COM, these terms have new meaning.

- A COM *client* is a program or object that calls methods in one or more interfaces provided by a COM server.
- A COM *server* interacts with clients by implementing interfaces that direct the client to methods supported by the interface. *In-process* COM servers are DLLs that run on the local machine. *Out-of-process* COM servers are .EXE files that can reside on the local machine, or on a networked computer. The COM specification has been extended to provide support for *distributed* processes (DCOM) running on remote computers. The term COM is now synonymous with both COM and DCOM.

General Programming Tips

This page contains tips for using the API effectively.

1. Know the difference between modifiable and non-modifiable handlers. The handlers that you create can be modified as needed. However, all handlers that you are debugging in Interaction Designer are read-only.

   > All currently open handlers are returned when a collection is returned. Some are modifiable, and some are not. For example, a handler that is being debugged is not modifiable, while a handler that you are creating within Interaction Designer is modifiable. If the II3IDHandler.IsReadOnly is True, all aspects of that handler are read-only, since the handler is being debugged in Interaction Designer.

2. Check for batch publishing before displaying a dialog.

   Before displaying a dialog, your application should check to see if Interaction Designer is batch publishing. If you display dialogs during this process, you may halt a batch publish operation occurring on the server.

3. Validate object IDs and notification events before publishing a handler.

   In the event that you do decide to implement the II3IDStepEvent interface for an initiator, it will be up to your initiator's code to in II3IDStepEvents::Validate to ensure that the proper Object ID and Notification Event are set. Designer will not disallow publishing if you have assigned an Object ID or Notification Event that do not match one of the Notification Events or Object ID's that were originally registered for the initiator. For instance, to disable publishing because a Notification Event was not set properly, you could add the following code to the II3IDEvents::Validate processing:

   > Please remember to internationalize the message strings submitted to LogMessage. The above uses a hard-coded string of "Notification Event Not Valid" which is done for demonstration purposes only.

Interfaces

II3ID Interface
II3ID::CreateHandler Method

Synopsis

Creates a new handler and returns an interface pointer to the handler.

IDL Function Prototype

HRESULT CreateHandler(

[out, retval] II3IDHandler ** NewHandler

);

C/C++ Syntax

HRESULT CreateHandler(II3IDHandler> ** NewHandler);

Parameters

NewHandler

The return value is an interface pointer to the new handler object.

II3ID::CurrentHandler Method

Synopsis

Returns a handler object to the currently active handler in Interaction Designer. Returns S_FALSE, rather than an exception if no handler is open.

IDL Function Prototype

HRESULT CurrentHandler(

[out, retval] II3IDHandler ** theHandler

);

C/C++ Syntax

HRESULT CurrentHandler(II3IDHandler ** theHandler);

Parameters

theHandler

The return value is an interface pointer to the handler that is currently active in Interaction Designer. If no handlers are currently loaded, a null interface pointer is returned.

II3ID::EscapeString Method

Synopsis

Escapes a string sequence for usage to the II3IDParameter::put_Expression call.

This method converts non-printing and extended ASCII characters in a string to a string that represents special characters using escape strings.

IDL Function Prototype

HRESULT EscapeString(

[in] BSTR StringToEscape,

[out, retval] BSTR * EscapedString

);

C/C++ Syntax

HRESULT EscapeString(BSTR StringToEscape, BSTR * EscapedString);

Parameters

StringToEscape

The input parameter is a string that contains non-printing or extended ASCII characters.

EscapedString

The return value is a string that has escape codes substituted for special characters.

II3ID::GetErrorMessage Method

Synopsis

Returns the error string for the specified eIdToolRetCode value defined in the i3idTool.h file. The string returned is internationalized for the local for which Designer was compiled.

IDL Function Prototype

HRESULT GetErrorMessage(

[in] long ErrorNumber,

[out, retval] BSTR * Message

);

C/C++ Syntax

HRESULT GetErrorMessage(long ErrorNumber, BSTR * Message);

Parameters

ErrorNumber

The error number.

Message

A string that describes the error.

II3ID::GetLicenseInfo Method

Synopsis

This method returns True if a valid license is available for the feature specified.

IDL Function Prototype

HRESULT GetLicenseInfo(

[in] BSTR FeatureString,

[out, retval] VARIANT * LicenseInfo

);

C/C++ Syntax

HRESULT GetLicenseInfo(BSTR FeatureString, VARIANT * LicenseInfo);

Parameters

FeatureString

A string that identifies the features licensed. E.g.: I3_FEATURE_VERSION_EIC, I3_FEATURE_VERSION_CIC, I3_FEATURE_VERSION_SIC, etc.

LicenseInfo

Returns a variant of VT_BOOL (True if the specified feature is licensed; otherwise False).

II3ID::MessageBox Method

Synopsis

Prompts the user with a message box and returns the entry back to the caller. See the Windows SDK help documentation about Messagebox for valid values for Type parameter.

This function will fail if you attempt to display a message box during a publish operation. You can use the IsBatchPublishing property to determine whether or not a publishing process is currently underway.

IDL Function Prototype

```
HRESULT MessageBox(
  [in] BSTR Message,
  [in, optional] BSTR Title,
  [in, optional, defaultvalue(0)] long Type,
  [out, retval] long * MessageBoxReturnValue
);
```

C/C++ Syntax

HRESULT MessageBox(BSTR Message, BSTR Title, long Type, long * MessageBoxReturnValue);

Parameters

Message

The message text that the tool displays to the user.

Title

The text that appears in the window caption.

Type

The uType as is defined in the Windows SDK call (e.g. MB_OK)

MessageBoxReturnValue

The value returned when the user presses a button to close the message box.

II3ID::OpenHandler Method

Synopsis

Opens the specified handler and returns a handler object.

IDL Function Prototype

HRESULT OpenHandler(

[in] BSTR HandlerFilePathToOpen,

[out, retval] II3IDHandler ** TheHandler

);

C/C++ Syntax

HRESULT OpenHandler(BSTR HandlerFilePathToOpen, II3IDHandler ** TheHandler);

Parameters

HandlerFilePathToOpen

The fully qualified path to an .ihd file.

TheHandler

An II3IDHandler object is returned.

II3ID::QueryNativeTypeName Method

Synopsis

Returns the name associated with an internal data type defined in Interaction Designer. For example, 'ID_String' returns 'String'.

IDL Function Prototype

HRESULT QueryNativeTypeName(

[in] I3IDNativeDataType NativeType,

[out, retval] BSTR * TypeName

);

C/C++ Syntax

HRESULT QueryNativeTypeName(I3IDNativeDataType NativeType, BSTR * TypeName);

Parameters

NativeType

One of the I3IDNativeDataType constants.

TypeName

The return value is the name of a data type.

II3ID::ReadProfileLong Method

Synopsis

This method reads a numeric value from the registry. To retrieve a value, specify the name of a section and the name of an entry, as if you were reading an INI file. Use II3ID::WriteProfileLong to write an integer value to the registry.

IDL Function Prototype

HRESULT ReadProfileLong(

[in] BSTR Section,

[in] BSTR Entry,

[in, optional, defaultvalue(0)] long DefaultValue,

[out, retval] long * Value

);

C/C++ Syntax

HRESULT ReadProfileLong(BSTR Section, BSTR Entry, long DefaultValue, long * Value);

Parameters

Section

Section name is an arbitrary string that defines a location in the registry (e.g.: MyTotals)

Entry

Entry is the name of a value that you wish to return from the section (e.g. TotalHits).

DefaultValue

This optional default value is used when no value is returned by the read operation.

Value

The value returned is a long integer.

II3ID::ReadProfileString Method

Synopsis

This method reads a string value from the registry. To retrieve a value, specify the name of a section and the name of an entry, as if you were reading an INI file. Use II3ID::WriteProfileString to write a string to the registry.

IDL Function Prototype

HRESULT ReadProfileString(

[in] BSTR Section,

[in] BSTR Entry,

[in, optional, defaultvalue(NULL)] BSTR DefaultValue,

[out, retval] BSTR * Value

);

C/C++ Syntax

HRESULT ReadProfileString(BSTR Section, BSTR Entry, BSTR DefaultValue, BSTR * Value);

Parameters

Section

is an arbitrary string that defines a location in the registry (e.g.: Employees)

Entry

Entry is the name of the item whose value you wish to retrieve (e.g. John Doe).

DefaultValue

This optional default value is used when no value is returned by the read operation.

Value

The return value is a string.

II3ID::RegisterForIdEvents Method

Synopsis

Register an interface pointer (i.e. II3IDEvents) to receive event notifications from Interaction Designer.

IDL Function Prototype

HRESULT RegisterForIdEvents(

[in] VARIANT * EventNotifier

);

C/C++ Syntax

HRESULT RegisterForIdEvents(VARIANT * EventNotifier);

Parameters

EventNotifier

Specifies what object Interaction Designer should call to process a menu event. This can be a variant that contains an IDispatch pointer, an IUnknown pointer, or a BSTR. If you specify a BSTR in the VARIANT, Interaction Designer will treat that string as a ProgId, create the object using that ProgId and QueryInterface the created object for an II3IDEvents interface pointer. For an IDispatch or IUnknown pointer, Interaction Designer will call QueryInterface on that pointer for the II3IDEvents interface.

II3ID::UnescapeString Method

Synopsis

Unescapes a string expression returned from II3IDParameter::get_Expression.

This method converts a string that contains escape codes that represent non-printing and extended ASCII characters back to a string that contains the non-printing or extended characters.

IDL Function Prototype

HRESULT UnescapeString(

[in] BSTR StringToUnescape,

[out, retval] BSTR * UnescapedString

);

C/C++ Syntax

HRESULT UnescapeString(BSTR StringToUnescape, BSTR * UnescapedString);

Parameters

StringToUnescape

The input parameter is a string that contains escape codes instead of non-printing or extended characters.

UnescapedString

The return value is a string that contains non-printing or extended characters instead of escape codes.

II3ID::WriteProfileLong Method

Synopsis

This method saves a numeric value to the registry. You must specify section and entry names, and the value to save. This process is comparable to writing a value to an INI file. Use II3ID::ReadProfileLong to read an integer value from the registry.

IDL Function Prototype

HRESULT WriteProfileLong(

[in] BSTR Section,

[in] BSTR Entry,

[in] long Value

);

C/C++ Syntax

HRESULT WriteProfileLong(BSTR Section, BSTR Entry, long Value);

Parameters

Section

The section that contains the entry.

Entry

The item you wish to save a value for.

Value

The value to save in the registry on the local machine.

II3ID::WriteProfileString Method

Synopsis

This method saves a string value to the registry. You must specify section and entry names, and the value to save. This process is comparable to writing a value to an INI file. Use II3ID::ReadProfileString to read an integer value from the registry.

IDL Function Prototype

HRESULT WriteProfileString(

[in] BSTR Section,

[in] BSTR Entry,

[in] BSTR Value

);

C/C++ Syntax

HRESULT WriteProfileString(BSTR Section, BSTR Entry, BSTR Value);

Parameters

Section

The section that contains the entry.

Entry

The item you wish to save a value for.

Value

The string value of the specified Entry that will be saved in the registry on the local machine.

II3ID::CurrentHandlerSelectedSteps Property

get_CurrentHandlerSelectedSteps

Returns an II3IDSteps collection of the currently selected steps for the handler displayed in the active window.

IDL Function Prototype

HRESULT CurrentHandlerSelectedSteps(

[out, retval] II3IDSteps ** TheSelectedSteps

);

C/C++ Syntax

HRESULT get_CurrentHandlerSelectedSteps(II3IDSteps ** TheSelectedSteps);

Parameters

TheSelectedSteps

An II3IDSteps collection object is returned.

II3ID::Handlers Property

get_Handlers

This property returns all of the currently loaded handlers in Interaction Designer.

IDL Function Prototype

HRESULT Handlers(

[out, retval] II3IDHandlers ** theHandlers

);

C/C++ Syntax

HRESULT get_Handlers(II3IDHandlers ** theHandlers);

Parameters

theHandlers

An II3IDHandlers collection is returned.

II3ID::ICServer Property

get_ICServer

Returns an interface pointer to the primary server Interaction Designer is connected to.

IDL Function Prototype

HRESULT ICServer(

[out, retval] II3IDICServer ** ICServer

);

C/C++ Syntax

HRESULT get_ICServer(II3IDICServer ** ICServer);

Parameters

ICServer

The return value is an II3IDICServer object.

II3ID::Initiators Property

get_Initiators

This property returns a collection of Initiators registered in Interaction Designer. This collection represents initiators displayed to a user after the File | Change Initiator menu operation is selected.

An initiator is always the first step in a handler. It tells Interaction Processor which event starts an instance of that handler. When one of the modules in CIC, such as Telephony Services, generates an event, Notifier sees that event and tells other modules about that event. One of these modules is Interaction Processor, where the handlers are registered. When the Notifier tells the Interaction Processor about an event, Interaction Processor starts an instance of a handler.

When you publish a handler, the handler's initiator tells Interaction Processor which event to watch for. An event is something that happens to an object. For example, a call (object) can be sent to voice mail (event). If an initiator is configured in a handler to start when calls are sent to voice mail, then Interaction Processor starts that handler any time it is notified of that event.

Note: Subroutine initiators are different from other initiators because they are started by a call from another handler instead of an event that occurs on the CIC system. For more information about initiators, see the Interaction Designer help system.

IDL Function Prototype

```
HRESULT Initiators(
   [out, retval] II3IDInitiators ** theInitiators
);
```

C/C++ Syntax

```
HRESULT get_Initiators( II3IDInitiators ** theInitiators);
```

Parameters

theInitiators

The return value is a collection of II3IDInitiators objects.

II3ID::IsBatchPublishing Property

get_IsBatchPublishing

This property indicates whether or not Interaction Designer is currently batch publishing. Batch publishing is a way of publishing a group of handlers automatically rather than one at a time through Interaction Designer.

By default, batch publishing publishes all of the default-packaged handlers listed in the i3handlers.lst file. Batch publishing can also process a custom .lst file. See "Batch Publishing" in the Interaction Designer help for more information.

Note: Don't try to display messages boxes or modal dialogs while a batch publishing operation is underway. If you do, you may halt a batch publish operation that is occurring on a server.

IDL Function Prototype

```
HRESULT IsBatchPublishing(
   [out, retval] VARIANT_BOOL * batchPublishing
);
```

C/C++ Syntax

HRESULT get_IsBatchPublishing(VARIANT_BOOL * batchPublishing);

Parameters

batchPublishing

This Boolean value is True if Interaction Designer is currently batch publishing; otherwise False.

II3ID::LastErrors Property

get_LastErrors

Retrieve the current automation error messages.

IDL Function Prototype

HRESULT LastErrors(

[out, retval] II3IDMessages ** AutomationErrorMessages

);

C/C++ Syntax

HRESULT get_LastErrors(II3IDMessages ** AutomationErrorMessages);

Parameters

AutomationErrorMessages

The return value is a collection of II3IDMessages that were logged for the last error.

II3ID::Locale Property

get_Locale

The locale that Interaction Designer is currently running in. This returns a Windows LCID (locale identifier) as the output.

IDL Function Prototype

HRESULT Locale(

[out, retval] long * theLocale

);

C/C++ Syntax

HRESULT get_Locale(long * theLocale);

Parameters

theLocale

The return value is a long that corresponds to a Window's LCID.

II3ID::MenuManager Property

get_MenuManager

Returns a pointer to the Interaction Designer Menu Manager. This allows you to add menu items below the Utilities menu.

IDL Function Prototype

HRESULT MenuManager(

  [out, retval] II3IDMenuManager ** theMenuManager

);

C/C++ Syntax

HRESULT get_MenuManager(II3IDMenuManager ** theMenuManager);

Parameters

theMenuManager

The return value is an interface pointer to an II3IDMenuManager object.

II3ID::ShowPublishErrorDialogs Property

get_ShowPublishErrorDialogs

Whether or not the user has specified the /LogPublishEvents command line switch. If so, your tool should not pop any modal dialogs for a publish and write to the event log directly.

IDL Function Prototype

HRESULT ShowPublishErrorDialogs(

  [out, retval] VARIANT_BOOL * ShowDialogs

);

C/C++ Syntax

HRESULT get_ShowPublishErrorDialogs(VARIANT_BOOL * ShowDialogs);

Parameters

ShowDialogs

If True, your tool should not display modal dialogs.

II3ID::Subroutines Property

get_Subroutines

Returns a collection of subroutines loaded in Interaction Designer.

IDL Function Prototype

HRESULT Subroutines(

  [out, retval] II3IDSubroutines ** theSubs

);

C/C++ Syntax

HRESULT get_Subroutines(II3IDSubroutines ** theSubs);

Parameters

theSubs

The return value is an II3IDSubroutines collection.

II3ID::Tools Property

get_Tools

Returns a collection of the currently registered tools within Interaction Designer. This can be used to query whether or not a tool is already registered.

IDL Function Prototype

HRESULT Tools(

  [out, retval] II3IDTools ** theTools

);

C/C++ Syntax

HRESULT get_Tools(II3IDTools ** theTools);

Parameters

theTools

The return value is an II3IDTools collection.

II3ID::Types Property

get_Types

Returns a collection of data types registered in Interaction Designer.

IDL Function Prototype

HRESULT Types(

  [out, retval] II3IDTypes ** theTypes

);

C/C++ Syntax

HRESULT get_Types(II3IDTypes ** theTypes);

Parameters

theTypes

An II3IDTypes object is returned.

II3ID::UserName Property

get_UserName

Returns the name of the currently registered user. This user name has been verified against Notifier, either explicitly or implicitly.

IDL Function Prototype

HRESULT UserName(

[out, retval] BSTR * IDUserName

);

C/C++ Syntax

HRESULT get_UserName(BSTR * IDUserName);

Parameters

IDUserName

The return value is the name of the user. That user name has been defined in Interaction Administrator.

II3ID2 Interface
II3ID2::BeginReplaySession Method

Synopsis

Replays a handler session.

IDL Function Prototype

HRESULT BeginReplaySession(

[in] BSTR HandlerLoadDirectory,

[in] BSTR InitialHandlerName,

[in] VARIANT ReplayEventNotifier,

[in, optional] BSTR InitialHandlerValidationClassName,

[out, retval] II3IDReplaySession ** NewSession

);

C/C++ Syntax

HRESULT BeginReplaySession(BSTR HandlerLoadDirectory, BSTR InitialHandlerName, VARIANT ReplayEventNotifier, BSTR InitialHandlerValidationClassName, II3IDReplaySession ** NewSession);

Parameters

HandlerLoadDirectory

The directory that Designer should use when loading handlers for a replay session.

InitialHandlerName

This is the initial handler to use in the replay session, such as System_IncomingCall. This is similar to picking the initial handler to use while debugging.

ReplayEventNotifier

The ReplayEventNotifier specifies a COM object that implements the II3IDReplaySessionEvents interface. It can be used to receive replay session events. ReplayEventNotifier is a VARIANT and can be one of the following types:

- VT_UNKNOWN - Designer will call QueryInterface on the IUnknown member (ReplayEventNotifier.punk) contained in the VARIANT for the II3IDReplaySessionEvents interface.
- VT_DISPATCH - Designer will call QueryInterface on the IDispatch member (ReplayEventNoifier.pdisp) contained in the VARIANT for the II3IDReplaySessionEvents interface.
- VT_BSTR - Designer will treat the string contained in ReplayEventNotifier as a ProgID and create the COM object specified by the ProgID. If the COM object could be created, then Designer will QueryInterface that object for the II3IDReplaySessionEvents interface.

InitialHandlerValidationClassName

InitialHandlerValidationClassName is an optional input parameter of type BSTR. When a handler is published, Designer will assign a value to its ClassName property. Each time the handler is published, the ClassName value will change.

So, if a class name is submitted to the BeginReplaySession method, Designer will check to make sure that the initial handler loaded in the replay session matches the class name specified. This is extremely helpful because you can ensure that the handler being displayed to the user matches the handler that IP was running for the replay session that is being run.

NewSession

An II3IDReplaySession object is returned.

II3ID2::CreateHandlerFromXMLFile Method

Synopsis

Reads the XML file specified in HandlerXMLFilePath and creates the handler from that.

IDL Function Prototype

HRESULT CreateHandlerFromXMLFile(

[in] BSTR HandlerXMLFilePath,

[out, retval] II3IDHandler ** NewHandler

);

C/C++ Syntax

HRESULT CreateHandlerFromXMLFile(BSTR HandlerXMLFilePath, II3IDHandler ** NewHandler);

Parameters

HandlerXMLFilePath

Fully qualified path to the XML file, including its filename.

NewHandler

The return value is an II3IDHandler handler object.

II3ID2::IsRunning Property

get_IsRunning

This property returns whether or not Designer is past the initialization that occurs at startup.

IDL Function Prototype

HRESULT IsRunning(

  [out, retval] VARIANT_BOOL * Running

);

C/C++ Syntax

HRESULT get_IsRunning(VARIANT_BOOL * Running);

Parameters

Running

The return value is a Boolean that indicates whether Interaction Designer is ready to interact with your application.

II3ID2::NotifierConnectionIsOpen Property

get_NotifierConnectionIsOpen

This property indicates whether or not the Notifier connection is currently open.

IDL Function Prototype

HRESULT NotifierConnectionIsOpen(

  [out, retval] VARIANT_BOOL * connOpen

);

C/C++ Syntax

HRESULT get_NotifierConnectionIsOpen(VARIANT_BOOL * connOpen);

Parameters

connOpen

The return value is True if the Notifier connection is open; otherwise False.

II3IDDebugStepEvents Interface

II3IDDebugStepEvents::ViewStepProperties Method

Synopsis

This is similar to II3IDStepEvents::EditStepProperties except that this method is called when the user views step properties in a debug session. Do not allow the user modify step parameters for this callout.

IDL Function Prototype

HRESULT ViewStepProperties(

  [in] II3IDStep * StepToDisplay,

  [in] long ParentHWND,

[out, retval] VARIANT_BOOL * BeenHandled

);

C/C++ Syntax

HRESULT ViewStepProperties(II3IDStep * StepToDisplay, long ParentHWND, VARIANT_BOOL * BeenHandled);

Parameters

StepToDisplay

The II3IDStep object whose properties will be viewed.

ParentHWND

The handle of the parent window.

BeenHandled

The return value is a Boolean flag that indicates whether or not the callback that has been called by Interaction Designer has handled this event. When this value is True, Interaction Designer does not need to perform its default event processing.

II3IDEvents Interface

II3IDEvents::HandlerClose Method

Synopsis

Called when a handler is being closed.

IDL Function Prototype

HRESULT HandlerClose(

[in] II3IDHandler * Handler

);

C/C++ Syntax

HRESULT HandlerClose(II3IDHandler * Handler);

Parameters

Handler

The II3IDHandler object that is being closed.

II3IDEvents::HandlerOpen Method

Synopsis

Called when a handler has been opened which occurs either from opening an existing handler or creating a new one.

IDL Function Prototype

HRESULT HandlerOpen(

[in] II3IDHandler * Handler

);

C/C++ Syntax

HRESULT HandlerOpen(II3IDHandler * Handler);

Parameters

Handler

The II3IDHandler object that has been opened.

II3IDEvents::HandlerPublish Method

Synopsis

Called before a handler is about to be published. This callout occurs before the callout to II3IDStepEvents::Publish. This event is fired **only** when a handler is published from Interaction Designer. It is not fired when you publish through the EICPublisher or another external publishing mechanism.

IDL Function Prototype

HRESULT HandlerPublish(

[in] II3IDHandler * Handler

);

C/C++ Syntax

HRESULT HandlerPublish(II3IDHandler * Handler);

Parameters

Handler

The II3IDHandler object that is about to be published.

II3IDEvents::Shutdown Method

Synopsis

Called when Interaction Designer is about to shut down.

IDL Function Prototype

HRESULT Shutdown(

[in] II3ID * Designer

);

C/C++ Syntax

HRESULT Shutdown(II3ID * Designer);

Parameters

Designer

The interface pointer to the instance of Interaction Designer that is about to shut down.

II3IDEvents::Startup Method

Synopsis

Called when Interaction Designer is starting up before any handlers are loaded but after InitializeTools/InitializeTypes/InitializeEnvironment methods have been called.

IDL Function Prototype

HRESULT Startup(

[in] II3ID * Designer

);

C/C++ Syntax

HRESULT Startup(II3ID * Designer);

Parameters

Designer

Interface pointer to the instance of Interaction Designer that is starting up.

II3IDEvents2 Interface

II3IDEvents2::HandlerAboutToSave Method

Synopsis

Called before Designer saves a handler. Returning a failure from this callout does not stop Designer from attempting to save the handler.

IDL Function Prototype

HRESULT HandlerAboutToSave(

[in] BSTR Path,

[in] II3IDHandler * Handler

);

C/C++ Syntax

HRESULT HandlerAboutToSave(BSTR Path, II3IDHandler * Handler);

Parameters

Path

Path to the handler file.

Handler

The II3IDHandler object being saved.

II3IDEvents2::HandlerAfterEditStepProperties Method

Synopsis

Called after a step has been edited and AFTER any II3IDStepEvents::StepUpdated call has been made by Designer.

IDL Function Prototype

HRESULT HandlerAfterEditStepProperties(

[in] II3IDStep2 * EditStep

);

C/C++ Syntax

HRESULT HandlerAfterEditStepProperties(II3IDStep2 * EditStep);

Parameters

EditStep

An II3IDStep2 object, which is an instance of a tool, subroutine or initiator within a handler.

II3IDEvents2::HandlerBeforeEditStepProperties Method

Synopsis

Called before the edit step properties dialog is displayed for a step. Returning a faulure from this callout does not stop Designer from displaying the edit step properties dialog.

IDL Function Prototype

HRESULT HandlerBeforeEditStepProperties(

[in] II3IDStep2 * EditStep

);

C/C++ Syntax

HRESULT HandlerBeforeEditStepProperties(II3IDStep2 * EditStep);

Parameters

EditStep

An II3IDStep2 object, which is an instance of a tool, subroutine or initiator within a handler.

II3IDEvents2::HandlerClose Method

Synopsis

Called when a handler is being closed. Returning a failure from this callout does not cause the handler not to close.

IDL Function Prototype

HRESULT HandlerClose(

[in] II3IDHandler * Handler

);

C/C++ Syntax

HRESULT HandlerClose(II3IDHandler * Handler);

Parameters

Handler

The II3IDHandler object that is being closed.

II3IDEvents2::HandlerOpen Method

Synopsis

Called when a handler has been opened which occurs either from opening an existing handler or creating a new one. Returning a failure from this callout does not cause the handler not to open.

IDL Function Prototype

HRESULT HandlerOpen(

[in] II3IDHandler * Handler

);

C/C++ Syntax

HRESULT HandlerOpen(II3IDHandler * Handler);

Parameters

Handler

The II3IDHandler object that was been opened.

II3IDEvents2::HandlerPublish Method

Synopsis

Called before a handler is about to be published. This callout occurs before the callout to II3IDStepEvents::Publish. Returning a failure from this call does not cause the publish to stop.

IDL Function Prototype

HRESULT HandlerPublish(

[in] II3IDHandler * Handler

);

C/C++ Syntax

HRESULT HandlerPublish(II3IDHandler * Handler);

Parameters

Handler

The II3IDHandler handler object that is about to be published.

II3IDEvents2::HandlerStepInserted Method

Synopsis

Called after a step has been inserted AND after any II3IDStepEvents::StepInerted callout has been made. Returning a failure from this callout does not cause the newly inserted step to go away.

IDL Function Prototype

HRESULT HandlerStepInserted(

[in] II3IDStep2 * InsertedStep

);

C/C++ Syntax

HRESULT HandlerStepInserted(II3IDStep2 * InsertedStep);

Parameters

InsertedStep

The II3IDStep2 step object that was inserted.

II3IDEvents2::Shutdown Method

Synopsis

Called when Interaction Designer is about to shut down.

IDL Function Prototype

HRESULT Shutdown(

[in] II3ID * Designer

);

C/C++ Syntax

HRESULT Shutdown(II3ID * Designer);

Parameters

Designer

The II3ID interface pointer to the instance of Interaction Designer that is about to shut down.

II3IDEvents2::Startup Method

Synopsis

Called when Interaction Designer is starting up before any handlers are loaded but after InitializeTools/InitializeTypes/InitializeEnvironment methods have been called.

IDL Function Prototype

HRESULT Startup(

[in] II3ID * Designer

);

C/C++ Syntax

HRESULT Startup(II3ID * Designer);

Parameters

Designer

The II3ID interface pointer to the running instance of Interaction Designer.

II3IDExitPath Interface

II3IDExitPath Interface

Overview

Derived From: IDispatch

Interface ID: {F9BD5BFB-42F5-4330-85D7-8B06E2EF224D}

II3IDExitPath exposes the functionality of an exit path defined for a tool, subroutine or initiator. Initiators and subroutines only have 1 exit path. Tool steps can have two or more exit paths. Exit Paths are paths that a step can take as a result of the step's action. Exit paths link steps to other steps. For example, a Condition step has two exit paths: True and False. If the condition evaluated by the Condition step is true, the step will exit along the True exit path. If the condition evaluates to False, the step will exit along the False exit path.

Methods

LinkToStep	Links this exit path to the step specified in the NextStep parameter. This method is used to link one step in a handler to another step.

Properties

Designer	Returns an Interaction Designer interface pointer.
ExitPathType	Returns what type of entity the exit path is coming from. Call II3IDExitPath::SourceTool if the ExitPath is of type Tool. If the ExitPath is for a step, call II3IDExitPath::SourceStep.
IsCustom	Indicates whether or not this exit path was added externally by the tool.
IsException	Returns whether or not the exit path was registered as being an exception (error) path.
Label	Returns the label associated with the exit path. This text is localized.
ReturnValue	Returns the return value that the tool returns to exit through this exit path.
SourceInitiator	Return an initiator object that wraps the initiator used to create this exit path.
SourceStep	Returns a step object that wraps the step for this exit path.
SourceSubroutine	Return a subroutine object that wraps the subroutine used to create this exit path.
SourceTool	Returns a tool object that wraps the tool definition for this exit path.

II3IDExitPath::Designer Property

get_Designer

Returns an Interaction Designer interface pointer.

IDL Function Prototype

HRESULT Designer(

[out, retval] II3ID ** ID

);

C/C++ Syntax

HRESULT get_Designer(II3ID ** ID);

Parameters

ID

The return value is a pointer to an II3ID object, which is Interaction Designer itself.

II3IDExitPath::ExitPathType Property

get_ExitPathType

Returns what type of entity the exit path is coming from. Call II3IDExitPath::SourceTool if the ExitPath is of type Tool. If the ExitPath is for a step, call II3IDExitPath::SourceStep.

IDL Function Prototype

HRESULT ExitPathType(

[out, retval] I3IDEntityType * StepType

);

C/C++ Syntax

HRESULT get_ExitPathType(I3IDEntityType * StepType);

Parameters

StepType

An I3IDEntityType value that identifies the type of entity this exit path is coming from.

II3IDExitPath::IsCustom Property

get_IsCustom

Indicates whether or not this exit path was added externally by the tool.

IDL Function Prototype

HRESULT IsCustom(

[out, retval] VARIANT_BOOL * ExitPathIsCustom

);

C/C++ Syntax

HRESULT get_IsCustom(VARIANT_BOOL * ExitPathIsCustom);

Parameters

ExitPathIsCustom

True if this exit path was added externally by the tool; otherwise False.

II3IDExitPath::IsException Property

get_IsException

Returns whether or not the exit path was registered as being an exception (error) path.

IDL Function Prototype

HRESULT IsException(

  [out, retval] VARIANT_BOOL * ExitPathIsException

);

C/C++ Syntax

HRESULT get_IsException(VARIANT_BOOL * ExitPathIsException);

Parameters

ExitPathIsException

True if the exit path was registered as being an exception; otherwise False.

II3IDExitPath::Label Property

get_Label

Returns the label associated with the exit path. This text is localized.

IDL Function Prototype

HRESULT Label(

  [out, retval] BSTR * ExitPathName

);

C/C++ Syntax

HRESULT get_Label(BSTR * ExitPathName);

Parameters

ExitPathName

A string containing the localized label of the exit path (e.g. Success).

II3IDExitPath::ReturnValue Property

get_ReturnValue

Returns the return value that the tool returns to exit through this exit path.

IDL Function Prototype

HRESULT ReturnValue(

[out, retval] long * ExitPathReturnValue

);

C/C++ Syntax

HRESULT get_ReturnValue(long * ExitPathReturnValue);

Parameters

ExitPathReturnValue

The value returned by the tool when it exits through this exit path.

---

put_ReturnValue

Sets the return value that the tool returns to exit through this exit path.

IDL Function Prototype

HRESULT ReturnValue(

[in] long ExitPathReturnValue

);

C/C++ Syntax

HRESULT put_ReturnValue(long ExitPathReturnValue);

Parameters

ExitPathReturnValue

A long number that the tool will assign as its return value when it exits through this exit path.

II3IDExitPath::SourceInitiator Property

get_SourceInitiator

Return an initiator object that wraps the initiator used to create this exit path.

IDL Function Prototype

HRESULT SourceInitiator(

[out, retval] II3IDInitiator ** Initiator

);

C/C++ Syntax

HRESULT get_SourceInitiator(II3IDInitiator ** Initiator);

Parameters

Initiator

An II3IDInitiator object is returned.

II3IDExitPath::SourceStep Property

get_SourceStep

Returns a step object that wraps the step for this exit path.

IDL Function Prototype

HRESULT SourceStep(

  [out, retval] II3IDStep ** Step

);

C/C++ Syntax

HRESULT get_SourceStep(II3IDStep ** Step);

Parameters

Step

An II3IDStep object is returned.

II3IDExitPath::SourceSubroutine Property

get_SourceSubroutine

Return a subroutine object that wraps the subroutine used to create this exit path.

IDL Function Prototype

HRESULT SourceSubroutine(

  [out, retval] II3IDSubroutine ** Subroutine

);

C/C++ Syntax

HRESULT get_SourceSubroutine(II3IDSubroutine ** Subroutine);

Parameters

Subroutine

An II3IDSubroutine subroutine object is returned.

II3IDExitPath::SourceTool Property

get_SourceTool

Returns a tool object that wraps the tool definition for this exit path.

IDL Function Prototype

HRESULT SourceTool(

  [out, retval] II3IDTool ** Tool

);

C/C++ Syntax

HRESULT get_SourceTool(II3IDTool ** Tool);

Parameters

Tool

An II3IDTool object is returned.

II3IDExitPaths Interface
II3IDExitPaths::Add Method

Synopsis

Adds an exit path to a tool. This should only be performed on tools that have NOT been committed with the II3IDTool::Commit method or on a step's exit paths collection.

IDL Function Prototype

HRESULT Add(

[in] BSTR PathLabel,

[in] long nReturnCode,

[in] VARIANT_BOOL bShowAsException,

[in, optional, defaultvalue(-1)] long Index,

[out, retval] II3IDExitPath ** TheExitPath

);

C/C++ Syntax

HRESULT Add(BSTR PathLabel, long nReturnCode, VARIANT_BOOL bShowAsException, long Index, II3IDExitPath ** TheExitPath);

Parameters

PathLabel

The name you want to assign to this exit path.

nReturnCode

The return code generated when this exit path is taken.

bShowAsException

Specify True to show this exit path as an exception.

Index

An index number can be optionally supplied. The default value is -1.

TheExitPath

The return value is an II3IDExitPath object.

II3IDExitPaths::AddFromExitPath Method

Synopsis

Lets you add an exit path from an existing exit path.

IDL Function Prototype

```
HRESULT AddFromExitPath(
  [in] II3IDExitPath * ExitPath,
  [in, optional, defaultvalue(-1)] long Index,
  [out, retval] II3IDExitPath ** NewExitPath
);
```

C/C++ Syntax

```
HRESULT AddFromExitPath( II3IDExitPath * ExitPath, long Index, II3IDExitPath ** NewExitPath);
```

Parameters

ExitPath

An II3IDExitPath object.

Index

Optional index number.The default value is -1.

NewExitPath

An II3IDExitPath object is returned.

II3IDExitPaths::Item Method

Synopsis

Returns an exit path object by it's index in the exit path collection.

IDL Function Prototype

```
HRESULT Item(
  [in] long ExitPathIndex,
  [out, retval] II3IDExitPath ** theExitPath
);
```

C/C++ Syntax

HRESULT Item(long ExitPathIndex, II3IDExitPath ** theExitPath);

Parameters

ExitPathIndex

The index number of the exit path that you wish to retrieve from the collection.

theExitPath

An II3IDExitPath object is returned.

II3IDExitPaths::QueryByExitPathReturnValue Method

Synopsis

Returns an exit path object by its return value in the exit path collection.

IDL Function Prototype

HRESULT QueryByExitPathReturnValue(

[in] long ExitPathReturnValue,

[out, retval] II3IDExitPath ** theExitPath

);

C/C++ Syntax

HRESULT QueryByExitPathReturnValue(long ExitPathReturnValue, II3IDExitPath ** theExitPath);

Parameters

ExitPathReturnValue

The return value used to select the exit path.

theExitPath

An II3IDExitPath exit path object is returned.

II3IDExitPaths::QueryByLabel Method

Synopsis

Returns an exit path object by it's name in the exit path collection.

IDL Function Prototype

HRESULT QueryByLabel(

[in] BSTR ExitPathLabel,

[out, retval] II3IDExitPath ** theExitPath

);

C/C++ Syntax

HRESULT QueryByLabel(BSTR ExitPathLabel, II3IDExitPath ** theExitPath);

Parameters

ExitPathLabel

The label of the exit path object in the collection.

theExitPath

The return value is an II3IDExitPath exit path object.

II3IDExitPaths::Remove Method

Synopsis

Removes an exit path from the exit paths collection. This should only be called on an exit path collection generated by II3IDStep::get_ExitPaths.

IDL Function Prototype

HRESULT Remove(

[in] long ExitPathIndex

);

C/C++ Syntax

HRESULT Remove(long ExitPathIndex);

Parameters

ExitPathIndex

The index of the item that will be removed from the collection.

II3IDExitPaths::RemoveByReturnValue Method

Synopsis

Removes an exit path from the exit paths collection by its return value. This should only be called on an exit path collection generated by II3IDStep::get_ExitPaths.

IDL Function Prototype

HRESULT RemoveByReturnValue(

[in] long ExitPathReturnValue

);

C/C++ Syntax

HRESULT RemoveByReturnValue(long ExitPathReturnValue);

Parameters

ExitPathReturnValue

An exit path return value.

II3IDExitPaths::Count Property

get_Count

Returns the number of items in the collection.

IDL Function Prototype

HRESULT Count(

  [out, retval] long * returnCount

);

C/C++ Syntax

HRESULT get_Count(long * returnCount);

Parameters

returnCount

The total number of exit path items in the collection.

II3IDExitPaths::IsModifiable Property

get_IsModifiable

Returns whether or not the exit path collection is modifiable.

IDL Function Prototype

HRESULT IsModifiable(

  [out, retval] VARIANT_BOOL * ExitPathCollectionIsModifiable

);

C/C++ Syntax

HRESULT get_IsModifiable(VARIANT_BOOL * ExitPathCollectionIsModifiable);

Parameters

ExitPathCollectionIsModifiable

True if the collection is modifiable; False if the collection is read-only.

II3IDExitPaths::RegisteredCount Property

get_RegisteredCount

The number of exit paths in the collection that were not dynamically added.

IDL Function Prototype

HRESULT RegisteredCount(

  [out, retval] long * returnCount

);

C/C++ Syntax

HRESULT get_RegisteredCount(long * returnCount);

Parameters

returnCount

The return value is the count of exit paths not dynamically added.

II3IDExpression Interface
II3IDExpression::Designer Property

get_Designer

Returns an Interaction Designer interface pointer.

IDL Function Prototype

HRESULT Designer(

[out, retval] II3ID ** ID

);

C/C++ Syntax

HRESULT get_Designer(II3ID ** ID);

Parameters

ID

The return value is a pointer to an II3ID object.

II3IDExpression::ExpressionString Property

get_ExpressionString

This property returns the expression associated with this parameter. Expressions are formulas that process literal values, variables, and operators to create a new value. Refer to Designer online help for comprehensive coverage concerning expressions.

IDL Function Prototype

HRESULT ExpressionString(

[out, retval] BSTR * theExpression

);

C/C++ Syntax

HRESULT get_ExpressionString(BSTR * theExpression);

Parameters

theExpression

The return value is a string that contains the expression.

II3IDExpression::IsVariable Property

get_IsVariable

This property returns True if this expression contains a variable.

IDL Function Prototype

HRESULT IsVariable(

[out, retval] VARIANT_BOOL * exprIsVariable

);

C/C++ Syntax

HRESULT get_IsVariable(VARIANT_BOOL * exprIsVariable);

Parameters

exprIsVariable

True if the expression contains a variable; otherwise False.

II3IDExpression::Variable Property

get_Variable

Returns the variable associated with this expression.

IDL Function Prototype

HRESULT Variable(

[out, retval] II3IDVariable ** theVariable

);

C/C++ Syntax

HRESULT get_Variable(II3IDVariable ** theVariable);

Parameters

theVariable

The return value is an II3IDVariable object.

II3IDExternal Interface

II3IDExternal::Designer Property

get_Designer

Returns an interface pointer to Interaction Designer.

IDL Function Prototype

HRESULT Designer(

[out, retval] II3ID ** ID

);

C/C++ Syntax

HRESULT get_Designer(II3ID ** ID);

Parameters

ID

The return value is an II3ID interface pointer.

II3IDHandler Interface
II3IDHandler::ChangeInitiator Method

Synopsis

This method changes the Initiator of the handler. An initiator is always the first step in a handler. It tells Interaction Processor which event starts an instance of that handler.

When one of the modules in CIC, such as Telephony Services, generates an event, Notifier sees the event and tells other modules about that event. One of these modules is Interaction Processor, where the handlers are registered. When the Notifier tells the Interaction Processor about an event, Interaction Processor starts an instance of a handler.

When you publish a handler, the handler's initiator tells Interaction Processor which event to watch for. An event is something that happens to an object. For example, a call (object) can be sent to voice mail (event). If you configure an initiator in a handler to start when calls are sent to voice mail, then Interaction Processor starts that handler any time it is notified of that event.

Note: *Subroutin*e initiators are different from other initiators because they are started by a call from another handler instead of an event that occurs on the CIC system. See the Subroutine initiator documentation for more information.

IDL Function Prototype

HRESULT ChangeInitiator(

  [in] VARIANT NewInitiator

);

C/C++ Syntax

HRESULT ChangeInitiator(VARIANT NewInitiator);

Parameters

NewInitiator

NewInitiator can be a VT_UNKNOWN or VT_DISPATCH interface pointer to an existing II3IDInitiator object or a VT_BSTR identifying the initiator by name.

II3IDHandler::Close Method

Synopsis

Closes the handler and all associated view (windows) for the handler.

IDL Function Prototype

HRESULT Close(

  [in] VARIANT_BOOL CloseEvenIfHandlerIsModified

);

C/C++ Syntax

HRESULT Close(VARIANT_BOOL CloseEvenIfHandlerIsModified);

Parameters

CloseEvenIfHandlerIsModified

Specify True to close the handler, losing unsaved changes, if any.

II3IDHandler::GenerateI3PUB Method

Synopsis

Generates a .i3pub file in the specified path. An .i3pub file packages information about a handler, so that the handler can be saved to disk and published later.

This intermediate file format contains the handler's .class file, configuration data needed by Directory Services, audio prompt data, and a copy of the .ihd file, along with everything else needed for publish. An .i3pub file can be transferred and published on an entirely different server.

IDL Function Prototype

HRESULT GenerateI3PUB(

[in] BSTR DestFilePath,

[in, optional, defaultvalue(0)] VARIANT_BOOL Activate,

[in, optional, defaultvalue(0)] VARIANT_BOOL Primary,

[in, optional] BSTR Category,

[in, optional] VARIANT_BOOL OverwriteIfFileExists

);

C/C++ Syntax

HRESULT GenerateI3PUB(BSTR DestFilePath, VARIANT_BOOL Activate, VARIANT_BOOL Primary, BSTR Category, VARIANT_BOOL OverwriteIfFileExists);

Parameters

DestFilePath

The fully qualified path and filename of the .i3pub file.

Activate

Specify True to activate the handler after it is published.

Primary

Specify True if the handler is a Primary handler; or False if it is a Monitor handler.

Category

This optional input parameter permits you to categorize the handler. Typical subroutine category entries are: Email Queuing, EMS, Generic Object, Interaction Attendant, IWeb, Reports, Support, System, Telephony, Voicemail, or Web.

OverwriteIfFileExists

To overwrite an existing .i3pub file if one exists, specify True.

II3IDHandler::InsertSubroutineStep Method

Synopsis

This method inserts a subroutine step into a handler.

IDL Function Prototype

HRESULT InsertSubroutineStep(

[in] VARIANT SubroutineToInsert,

[in, defaultvalue(0)] long XPos,

[in, defaultvalue(0)] long YPos,

[out, retval] II3IDStep ** NewStep

);

C/C++ Syntax

HRESULT InsertSubroutineStep(VARIANT SubroutineToInsert, long XPos, long YPos, II3IDStep ** NewStep);

Parameters

SubroutineToInsert

SubroutineToInsert can be a VT_UNKNOWN or VT_DISPATCH interface pointer to an existing II3IDSubroutine. You may also submit a VT_BSTR identifying the subroutine by name.

XPos

The horizontal position of the subroutine step. The default value is 0.

YPos

The vertical position of the subroutine step. The default value is 0.

NewStep

The return value is an II3IDStep object.

II3IDHandler::InsertToolStep Method

Synopsis

This method inserts a tool step into a handler.

IDL Function Prototype

HRESULT InsertToolStep(

[in] VARIANT ToolToInsert,

[in, defaultvalue(0)] long XPos,

[in, defaultvalue(0)] long YPos,

[out, retval] II3IDStep ** NewStep

);

C/C++ Syntax

HRESULT InsertToolStep(VARIANT ToolToInsert, long XPos, long YPos, II3IDStep ** NewStep);

Parameters

ToolToInsert

ToolToInsert can be a VT_UNKNOWN or VT_DISPATCH interface pointer to an existing II3IDTool. You may also submit a VT_BSTR with the following format: <ToolModule>::<ToolName>

XPos

The horizontal position of the tool step. The default value is 0.

YPos

The vertical position of the tool step. The default value is 0.

NewStep

The return value is an II3IDStep object.

II3IDHandler::IsEqualTo Method

Synopsis

This property indicates whether or not the inbound handler interface pointer points to the same handler.

IDL Function Prototype

HRESULT IsEqualTo(

[in] II3IDHandler * HandlerPtr,

[out, retval] VARIANT_BOOL * Equal

);

C/C++ Syntax

HRESULT IsEqualTo(II3IDHandler * HandlerPtr, VARIANT_BOOL * Equal);

Parameters

HandlerPtr

An II3IDHandler object.

Equal

True if the pointer points to the same handler; otherwise False.

II3IDHandler::Publish Method

Synopsis

Publishes the handler to the default IC server.

IDL Function Prototype

HRESULT Publish(

  [in, optional, defaultvalue(0)] VARIANT_BOOL Activate,

  [in, optional, defaultvalue(0)] VARIANT_BOOL Primary,

  [in, optional] BSTR Category

);

C/C++ Syntax

HRESULT Publish(VARIANT_BOOL Activate, VARIANT_BOOL Primary, BSTR Category);

Parameters

Activate

Specify True to activate the handler after it is published.

Primary

Specify True if this handler is a Primary handler; False if it is a Monitor handler.

Category

This optional input parameter permits you to categorize the handler. Typical category entries are: Email Queuing, EMS, Generic Object, Interaction Attendant, IWeb, Reports, Support, System, Telephony, Voicemail, or Web.

II3IDHandler::RemoveStep Method

Synopsis

Removes a step from a handler.

IDL Function Prototype

HRESULT RemoveStep(

  [in] long StepIDToRemove

);

C/C++ Syntax

HRESULT RemoveStep(long StepIDToRemove);

Parameters

StepIDToRemove

The ID number of the step to remove.

II3IDHandler::Save Method

Synopsis

Saves the handler.

IDL Function Prototype

HRESULT Save();

C/C++ Syntax

HRESULT Save();

Parameters

None.

II3IDHandler::SaveAs Method

Synopsis

Saves the handler using the filename and path specified.

IDL Function Prototype

```
HRESULT SaveAs(
  [in] VARIANT_BOOL OverwriteIfFileExists,
  [in] BSTR FilePath
);
```

C/C++ Syntax

HRESULT SaveAs(VARIANT_BOOL OverwriteIfFileExists, BSTR FilePath);

Parameters

OverwriteIfFileExists

To overwrite an existing file, specify True.

FilePath

The fully qualified path and filename. Specify .IHD as the file extension.

II3IDHandler::SetModified Method

Synopsis

Sets the modified flag on a handler.

IDL Function Prototype

HRESULT SetModified();

C/C++ Syntax

HRESULT SetModified();

Parameters

None.

II3IDHandler::Validate Method

Synopsis

This method validates a handler to see if it is ready for publish. If it is not ready, the ValidationMessages parameter will contain a list of messages indicating why the handler is not ready to be published.

IDL Function Prototype

HRESULT Validate(

  [out, retval] II3IDMessages ** ValidationMessages

);

C/C++ Syntax

HRESULT Validate(II3IDMessages ** ValidationMessages);

Parameters

ValidationMessages

The return value is an II3IDMessages object that contains a list of messages indicating why the handler cannot be published.

II3IDHandler::Category Property

get_Category

Returns the category of this handler (applicable for handlers with subroutine initiators).

IDL Function Prototype

HRESULT Category(

  [out, retval] BSTR * HandlerCategory

);

C/C++ Syntax

HRESULT get_Category(BSTR * HandlerCategory);

Parameters

HandlerCategory

The string returned is the category assigned to this handler.

II3IDHandler::Description Property

get_Description

Returns the description text for the handler.

IDL Function Prototype

HRESULT Description(

  [out, retval] BSTR * HandlerDescription

);

C/C++ Syntax

HRESULT get_Description(BSTR * HandlerDescription);

Parameters

HandlerDescription

This handler's description.

---

put_Description

Sets the description text for the handler.

IDL Function Prototype

HRESULT Description(

  [in] BSTR HandlerDescription

);

C/C++ Syntax

HRESULT put_Description(BSTR HandlerDescription);

Parameters

HandlerDescription

This handler's description.

II3IDHandler::Designer Property

get_Designer

Returns an Interaction Designer interface pointer.

IDL Function Prototype

HRESULT Designer(

  [out, retval] II3ID ** ID

);

C/C++ Syntax

HRESULT get_Designer(II3ID ** ID);

Parameters

ID

The return value is an II3ID interface pointer.

II3IDHandler::FilePath Property

get_FilePath

Returns the file path where the handler is located.

IDL Function Prototype

HRESULT FilePath(

  [out, retval] BSTR * FilePath

);

C/C++ Syntax

HRESULT get_FilePath(BSTR * FilePath);

Parameters

FilePath

The fully qualified path and filename are returned.

II3IDHandler::InitiatorStep Property

get_InitiatorStep

Retrieves the initiator step for this handler. An initiator step identifies the event that initiates an instance of this handler.

IDL Function Prototype

HRESULT InitiatorStep(

  [out, retval] II3IDStep ** theInitiator

);

C/C++ Syntax

HRESULT get_InitiatorStep(II3IDStep ** theInitiator);

Parameters

theInitiator

The return value is the II3IDStep object that is the initiator step for this handler.

II3IDHandler::IsModified Property

get_IsModified

Whether or not the handler has been modified.

IDL Function Prototype

HRESULT IsModified(

  [out, retval] VARIANT_BOOL * Modified

);

C/C++ Syntax

HRESULT get_IsModified(VARIANT_BOOL * Modified);

Parameters

Modified

True if the handler has been modified; otherwise False.

II3IDHandler::IsPublishable Property

get_IsPublishable

This property indicates whether or not a handler is publishable.

IDL Function Prototype

HRESULT IsPublishable(

  [out, retval] VARIANT_BOOL * CanBePublished

);

C/C++ Syntax

HRESULT get_IsPublishable(VARIANT_BOOL * CanBePublished);

Parameters

CanBePublished

True if the handler is publishable; False is the handler contains errors that will prevent it from being published.

II3IDHandler::IsReadOnly Property

get_IsReadOnly

Returns whether the handler is read-only. Handlers that have been opened for debugging are read only.

IDL Function Prototype

HRESULT IsReadOnly(

  [out, retval] VARIANT_BOOL * HandlerIsReadOnly

);

C/C++ Syntax

HRESULT get_IsReadOnly(VARIANT_BOOL * HandlerIsReadOnly);

Parameters

HandlerIsReadOnly

True if the handler is read-only.

II3IDHandler::Name Property

get_Name

Returns the name of the handler, without path information.

IDL Function Prototype

HRESULT Name(

  [out, retval] BSTR * TheName

);

C/C++ Syntax

HRESULT get_Name(BSTR * TheName);

Parameters

TheName

The name of the handler, including its .IHD extension. To retrieve the path with the filename, use the II3IDHandler::FilePath property.

II3IDHandler::Steps Property

get_Steps

Returns a collection of steps contained in this handler.

IDL Function Prototype

HRESULT Steps(

[out, retval] II3IDSteps ** Steps

);

C/C++ Syntax

HRESULT get_Steps(II3IDSteps ** Steps);

Parameters

Steps

The return value is an II3IDSteps collection.

II3IDHandler::ValidationMessages Property

get_ValidationMessages

Retrieve the validation messages for the handler.

IDL Function Prototype

HRESULT ValidationMessages(

[out, retval] II3IDMessages ** HandlerValidationMessages

);

C/C++ Syntax

HRESULT get_ValidationMessages(II3IDMessages ** HandlerValidationMessages);

Parameters

HandlerValidationMessages

The return value is a collection of II3IDMessages messages.

II3IDHandler::Variables Property

get_Variables

Retrieves a collection of all variables used by this handler.

IDL Function Prototype

HRESULT Variables(

[out, retval] II3IDVariables ** theVariables

);

C/C++ Syntax

HRESULT get_Variables(II3IDVariables ** theVariables);

Parameters

theVariables

The return value is an II3IDVariables collection.

II3IDHandler::XML Property

get_XML

Retrieves XML step and variable information from a handler.

IDL Function Prototype

HRESULT XML(

[out, retval] VARIANT * handlerXML

);

C/C++ Syntax

HRESULT get_XML(VARIANT * handlerXML);

Parameters

handlerXML

The return value is a VARIANT that contains step and variable information.

II3IDHandler2 Interface

II3IDHandler2::IsExpressionValid Method

Synopsis

Returns whether or not the expression is valid for the specified type and whether or not the expression is an input or not.

IDL Function Prototype

HRESULT IsExpressionValid(

[in] BSTR Expression,

[in] VARIANT TypeSpecifier,

[in] VARIANT_BOOL ExpressionIsInput,

[in,out] BSTR * ErrorString,

[out, retval] VARIANT_BOOL * IsValid

);

C/C++ Syntax

HRESULT IsExpressionValid(BSTR Expression, VARIANT TypeSpecifier, VARIANT_BOOL ExpressionIsInput, BSTR * ErrorString, VARIANT_BOOL * IsValid);

Parameters

Expression

The conditional expression whose formula is to be validated.

TypeSpecifier

TypeSpecifier denotes the type for the input parameter. It is a VARIANT that should be one of the three following types:

- VT_BSTR - A string of the format "<TypeModule>::<TypeName>" that identifies the type for this parameter. For example, the string "::Integer" could be used to specify a type of Integer. In this case, the module name for the type is empty.
- VT_UNKNOWN - The punkVal member should contain an IUnknown pointer to a COM object that implements the II3IDType interface.
- VT_DISPATCH - The pdispVal member should contain an IDispatch pointer to a COM object that implements the II3IDType interface.

For VT_UNKNOWN and VT_DISPATCH, "the COM object that implements the II3IDType interface" means that it is a type object that was retrieved from a method off of an II3IDTypes call such as the II3IDTypes::Item method.

ExpressionIsInput

Specify True if this expression is an input, False for output expressions.

ErrorString

String that describes error condition.

IsValid

The return value is True if the expression is valid; otherwise False is returned.

II3IDHandler2::StepsForSubroutine Method

Synopsis

Returns a collection of steps in the handler that use the specified subroutine.

IDL Function Prototype

HRESULT StepsForSubroutine(

[in] VARIANT SubroutineSpecifier,

[out, retval] II3IDSteps ** SubroutineSteps

);

C/C++ Syntax

HRESULT StepsForSubroutine(VARIANT SubroutineSpecifier, II3IDSteps ** SubroutineSteps);

Parameters

SubroutineSpecifier

The name of the subroutine to search for.

SubroutineSteps

The return value is a collection of II3IDSteps step objects.

II3IDHandler2::StepsForTool Method

Synopsis

Returns a collection of steps in the handler that use the specified tool.

IDL Function Prototype

HRESULT StepsForTool(

[in] VARIANT ToolSpecifier,

[out, retval] II3IDSteps ** ToolSteps

);

C/C++ Syntax

HRESULT StepsForTool(VARIANT ToolSpecifier, II3IDSteps ** ToolSteps);

Parameters

ToolSpecifier

Name of the tool to search for.

ToolSteps

The return value is a collection of II3IDSteps step objects.

II3IDHandler2::ClassName Property

get_ClassName

Returns the name of the .class file that was used when the handler was last published.

IDL Function Prototype

HRESULT ClassName(

[out, retval] BSTR * ClassFileName

);

C/C++ Syntax

HRESULT get_ClassName(BSTR * ClassFileName);

Parameters

ClassFileName

The return value is a class

II3IDHandler2::NameUsedForResourceCallouts Property

get_NameUsedForResourceCallouts

Returns the handler name as it is passed out to a resource-type callout.

IDL Function Prototype

HRESULT NameUsedForResourceCallouts(

[out, retval] BSTR * HandlerName

);

C/C++ Syntax

HRESULT get_NameUsedForResourceCallouts(BSTR * HandlerName);

Parameters

HandlerName

The return value is a string that contains the name of the handler.

II3IDHandlers Interface

II3IDHandlers::Item Method

Synopsis

Retrieves a handler by its index within the handler collection.

IDL Function Prototype

HRESULT Item(

[in] long HandlerIndex,

[out, retval] II3IDHandler ** theHandler

);

C/C++ Syntax

HRESULT Item(long HandlerIndex, II3IDHandler ** theHandler);

Parameters

HandlerIndex

The 0-based index number of the handler you wish to retrieve.

theHandler

The return value is an II3IDHandler handler object.

II3IDHandlers::Count Property

get_Count

Returns the number of items in the collection.

IDL Function Prototype

HRESULT Count(

  [out, retval] long * returnCount

);

C/C++ Syntax

HRESULT get_Count(long * returnCount);

Parameters

returnCount

The return value is the total number of handlers that are currently opened in Interaction Designer.

II3IDICServer Interface

II3IDICServer::ActivateHandler Method

Synopsis

Activates the handler specified by HandlerName. If ActivatePrimary is VARIANT_TRUE, the handler is activated primary. Otherwise, the handler is activated monitor.

IDL Function Prototype

HRESULT ActivateHandler(

  [in] BSTR HandlerName,

  [in] VARIANT_BOOL Primary

);

C/C++ Syntax

HRESULT ActivateHandler(BSTR HandlerName, VARIANT_BOOL Primary);

Parameters

HandlerName

The name of the handler to activate.

Primary

Specify True to activate the handler as a Primary handler, or False if the handler is a Monitor handler.

II3IDICServer::DeactivateHandler Method

Synopsis

Deactivates the handler specified by HandlerName.

IDL Function Prototype

HRESULT DeactivateHandler(

  [in] BSTR HandlerName

);

C/C++ Syntax

HRESULT DeactivateHandler(BSTR HandlerName);

Parameters

HandlerName

The name of the handler to deactivate.

II3IDICServer::DebugHandler Method

Synopsis

Launches a debugging session in Interaction Designer for the specified handler. Debugging allows you to monitor a running handler in Interaction Designer.

IDL Function Prototype

HRESULT DebugHandler(

  [in] BSTR HandlerName

);

C/C++ Syntax

HRESULT DebugHandler(BSTR HandlerName);

Parameters

HandlerName

The name of the handler to debug.

II3IDICServer::GetPublishedHandler Method

Synopsis

Retrieves the published IHD file for the handler specified by HandlerName from the IC server and saves it to DestinationPath.

IDL Function Prototype

HRESULT GetPublishedHandler(

  [in] BSTR HandlerName,

[in] VARIANT_BOOL OverwriteIfDestinationPathExists,

[in] BSTR DestinationPath

);

C/C++ Syntax

HRESULT GetPublishedHandler(BSTR HandlerName, VARIANT_BOOL OverwriteIfDestinationPathExists, BSTR DestinationPath);

Parameters

HandlerName

The name of the handler whose published file you wish to retrieve.

OverwriteIfDestinationPathExists

Specify True to overwrite an existing file, if one exists.

DestinationPath

The path where the IHD file will be saved.

II3IDICServer::QueryHandlerInfo Method

Synopsis

Returns an XML DOM pointer that contains information about the published handlers on the server.Pass NULL (or zero length string) in for HandlerName to retrieve information for all of the published handlers.

IDL Function Prototype

HRESULT QueryHandlerInfo(

[in, optional] BSTR HandlerName,

[out, retval] VARIANT * HandlerInfo

);

C/C++ Syntax

HRESULT QueryHandlerInfo(BSTR HandlerName, VARIANT * HandlerInfo);

Parameters

HandlerName

The name of a handler. If you specify an empty string, information about all published handlers will be returned.

HandlerInfo

The return value is an XML DOM pointer that contains information about the published handler(s) on the server.

II3IDICServer::Designer Property

get_Designer

Returns an Interaction Designer interface pointer.

IDL Function Prototype

HRESULT Designer(

[out, retval] II3ID ** ID

);

C/C++ Syntax

HRESULT get_Designer(II3ID ** ID);

Parameters

ID

The return value is a pointer to the II3ID interface.

II3IDICServer::Name Property

get_Name

The name of the IC server.

IDL Function Prototype

HRESULT Name(

[out, retval] BSTR * ServerName

);

C/C++ Syntax

HRESULT get_Name(BSTR * ServerName);

Parameters

ServerName

An IC server name is returned.

II3IDICServer2 Interface

II3IDICServer2::StartDebugSession Method

Synopsis

Starts a Debug Session. The initial breakpoint will be set on the initiator. EventNotifier specifies an object that implements the II3IDDebugSessionEvents interface.

IDL Function Prototype

HRESULT StartDebugSession(

[in] BSTR HandlerName,

[in] VARIANT EventSink,

[out, retval] II3IDDebugSession ** DebugSession

);

C/C++ Syntax

HRESULT StartDebugSession(BSTR HandlerName, VARIANT EventSink, II3IDDebugSession ** DebugSession);

Parameters

HandlerName

The handler to debug.

EventSink

The object that Interaction Designer should call back on with events specified in the II3IDICServer2 interface. The variant should be one of the following:

VT_EMPTY / VT_NULL - specify when you don't want any callbacks to occur.

VT_BSTR - bstrVal contains a ProgID of a COM object that Designer should create that implements the interface.

VT_UNKNOWN - an IUnknown interface pointer that Designer can query for an II3IDICServer2 interface pointer.

VT_DISPATCH - an IDispatch interface pointer that Designer can query for an II3IDICServer2 interface pointer.

DebugSession

The return value is an II3IDDebugSession object.

II3IDICServer2::ServerIsLicensedForPublish Property

get_ServerIsLicensedForPublish

This property indicates whether or not publishing is allowed on the server.

IDL Function Prototype

HRESULT ServerIsLicensedForPublish(

  [out, retval] VARIANT_BOOL * publishLicensed

);

C/C++ Syntax

HRESULT get_ServerIsLicensedForPublish(VARIANT_BOOL * publishLicensed);

Parameters

publishLicensed

The return value is True if publishing is allowed on the server; otherwise False.

II3IDInitiator Interface

II3IDInitiator::Commit Method

Synopsis

Commits an initiator object so that it can be used by handler developers.

IDL Function Prototype

HRESULT Commit();

C/C++ Syntax

HRESULT Commit();

Parameters

None.

II3IDInitiator::RegisterForStepEvents Method

Synopsis

Registers for step events from ID for this initiator.

IDL Function Prototype

HRESULT RegisterForStepEvents(

[in] VARIANT EventNotifier

);

C/C++ Syntax

HRESULT RegisterForStepEvents(VARIANT EventNotifier);

Parameters

EventNotifier

EventNotifier can be a VT_UNKNOWN or VT_DISPATCH pointing to an existing COM object or a VT_BSTR ProgID of an object that implements the II3IDStepEvents interface.

II3IDInitiator::Description Property

get_Description

Returns the description of this initiator.

IDL Function Prototype

HRESULT Description(

[out, retval] BSTR * InitDescription

);

C/C++ Syntax

HRESULT get_Description(BSTR * InitDescription);

Parameters

InitDescription

The description of this initiator.

II3IDInitiator::Designer Property

get_Designer

Returns an Interaction Designer interface pointer.

IDL Function Prototype

HRESULT Designer(

  [out, retval] II3ID ** ID

);

C/C++ Syntax

HRESULT get_Designer(II3ID ** ID);

Parameters

ID

An II3ID interface pointer is returned.

II3IDInitiator::Events Property

get_Events

Returns a collection of the currently selected steps for the active view of the handler (if it's active).

IDL Function Prototype

HRESULT Events(

  [out, retval] II3IDInitiatorEvents ** InitEvents

);

C/C++ Syntax

HRESULT get_Events(II3IDInitiatorEvents ** InitEvents);

Parameters

InitEvents

The return value is an II3IDInitiatorEvents collection.

II3IDInitiator::ExitPath Property

get_ExitPath

Returns the exit path for the initiator.

IDL Function Prototype

HRESULT ExitPath(

  [out, retval] II3IDExitPath ** InitExitPath

);

C/C++ Syntax

HRESULT get_ExitPath(II3IDExitPath ** InitExitPath);

Parameters

InitExitPath

An II3IDExitPath exit path object is returned.

II3IDInitiator::HelpContext Property

get_HelpContext

Returns the WinHelp context string associated with this initiator.

IDL Function Prototype

HRESULT HelpContext(

  [out, retval] long * InitHelpCntxt

);

C/C++ Syntax

HRESULT get_HelpContext(long * InitHelpCntxt);

Parameters

InitHelpCntxt

The Winhelp context string returned is a number that identifies a topic in a help file.

II3IDInitiator::HelpFile Property

get_HelpFile

Returns the name of the WinHelp help file that describes this initiator.

IDL Function Prototype

HRESULT HelpFile(

  [out, retval] BSTR * InitHelpFile

);

C/C++ Syntax

HRESULT get_HelpFile(BSTR * InitHelpFile);

Parameters

InitHelpFile

A Windows help filename is returned.

II3IDInitiator::II3IDInitiatorAddOnInstance Property

get_II3IDInitiatorAddOnInstance

Returns a COM Interface pointer to the initiator registered in Interaction Designer if the InitiatorSpecifier parameter to II3IDInitaitors::RegisterInitiator identified a COM object that implemented the II3IDInitiatorAddOn interface.

IDL Function Prototype

HRESULT II3IDInitiatorAddOnInstance(

  [out, retval] II3IDInitiatorAddOn ** InitiatorInstance

);

C/C++ Syntax

HRESULT get_II3IDInitiatorAddOnInstance(II3IDInitiatorAddOn ** InitiatorInstance);

Parameters

InitiatorInstance

The return value is a COM interface pointer to the registered initiator.

II3IDInitiator::InitNotificationObjectType Property

get_InitNotificationObjectType

Returns the notification event associated with this initiator.

IDL Function Prototype

HRESULT InitNotificationObjectType(

  [out, retval] BSTR * InitNotifObjType

);

C/C++ Syntax

HRESULT get_InitNotificationObjectType(BSTR * InitNotifObjType);

Parameters

InitNotifObjType

The return value is the notification event associated with this initiator.

II3IDInitiator::IsCommitted Property

get_IsCommitted

Lets you know whether or not this initiator has been committed.

IDL Function Prototype

HRESULT IsCommitted(

  [out, retval] VARIANT_BOOL * InitIsCommitted

);

C/C++ Syntax

HRESULT get_IsCommitted(VARIANT_BOOL * InitIsCommitted);

Parameters

InitIsCommitted

True if the initiator has been committed; otherwise False.

II3IDInitiator::IsExternal Property

get_IsExternal

Indicates whether or not this initiator is an external initiator.

IDL Function Prototype

HRESULT IsExternal(

  [out, retval] VARIANT_BOOL * InitIsExternal

);

C/C++ Syntax

HRESULT get_IsExternal(VARIANT_BOOL * InitIsExternal);

Parameters

InitIsExternal

True if this initiator is external; False if it is internal.

II3IDInitiator::Label Property

get_Label

Returns the label for this initiator.

IDL Function Prototype

HRESULT Label(

  [out, retval] BSTR * InitLabel

);

C/C++ Syntax

HRESULT get_Label(BSTR * InitLabel);

Parameters

InitLabel

The text of the label associated with this initiator.

II3IDInitiator::ModuleName Property

get_ModuleName

Returns the module name for this initiator.

IDL Function Prototype

HRESULT ModuleName(

  [out, retval] BSTR * InitModuleName

);

C/C++ Syntax

HRESULT get_ModuleName(BSTR * InitModuleName);

Parameters

InitModuleName

This initiator's module name.

II3IDInitiator::Name Property

get_Name

Returns the name of this initiator.

IDL Function Prototype

HRESULT Name(

  [out, retval] BSTR * InitName

);

C/C++ Syntax

HRESULT get_Name(BSTR * InitName);

Parameters

InitName

The name of this initiator.

II3IDInitiator::ObjectIDs Property

get_ObjectIDs

Returns a collection of the currently selected steps for the active view of the handler. The handler must be active.

IDL Function Prototype

HRESULT ObjectIDs(

  [out, retval] II3IDInitiatorObjectIDs ** InitiatorObjectIDs

);

C/C++ Syntax

HRESULT get_ObjectIDs(II3IDInitiatorObjectIDs ** InitiatorObjectIDs);

Parameters

InitiatorObjectIDs

The return value is an II3IDInitiatorObjectIDs collection.

II3IDInitiator::ParameterDefinitions Property

get_ParameterDefinitions

Returns a collection of parameter definitions for this initiator. Calls to this property to retrieve parameter definitions for an internal intiator or internal tool will fail, rather than return an empty collection.

IDL Function Prototype

HRESULT ParameterDefinitions(

  [out, retval] II3IDParameterDefinitions ** InitParmList

);

C/C++ Syntax

HRESULT get_ParameterDefinitions(II3IDParameterDefinitions ** InitParmList);

Parameters

InitParmList

The return value is an II3IDParameterDefinitions collection.

II3IDInitiator::RuntimeDLLName Property

get_RuntimeDLLName

Returns the runtime DLL name for the initiator.

IDL Function Prototype

HRESULT RuntimeDLLName(

  [out, retval] BSTR * InitRuntimeDLLName

);

C/C++ Syntax

HRESULT get_RuntimeDLLName(BSTR * InitRuntimeDLLName);

Parameters

InitRuntimeDLLName

The return value is a string containing the name of the runtime DLL for this initiator.

II3IDInitiator::RuntimeFunctionName Property

get_RuntimeFunctionName

Returns the runtime function name for the initiator.

IDL Function Prototype

HRESULT RuntimeFunctionName(

  [out, retval] BSTR * InitRuntimeFunctionName

);

C/C++ Syntax

HRESULT get_RuntimeFunctionName(BSTR * InitRuntimeFunctionName);

Parameters

InitRuntimeFunctionName

The initiator's runtime function name.

II3IDInitiator::Version Property

get_Version

Returns the registered version of this initiator.

IDL Function Prototype

HRESULT Version(

  [out, retval] BSTR * InitVersion

);

C/C++ Syntax

HRESULT get_Version(BSTR * InitVersion);

Parameters

InitVersion

The return value is a string that contains version information about this initiator.

II3IDInitiator2 Interface

II3IDInitiator2::RegisterForDebugStepEvents Method

Synopsis

Registers an object to receive debug step events from Interaction Designer for this initiator. These step events are fired during debug sessions in Interaction Designer.

IDL Function Prototype

HRESULT RegisterForDebugStepEvents(

  [in] VARIANT DebugEventNotifier

);

C/C++ Syntax

HRESULT RegisterForDebugStepEvents(VARIANT DebugEventNotifier);

Parameters

DebugEventNotifier

The object to register to receive debug step events.

II3IDInitiator2::RegisterForXMLStepEvents Method

Synopsis

Registers an object to receive XML step events from Interaction Designer for this initiator. These step events are fired when configuring steps from XML.

IDL Function Prototype

HRESULT RegisterForXMLStepEvents(

  [in] VARIANT XMLEventNotifier

);

C/C++ Syntax

HRESULT RegisterForXMLStepEvents(VARIANT XMLEventNotifier);

Parameters

XMLEventNotifier

The object that you wish to register for XML step events.

II3IDInitiator2::InitiatorHandle Property

get_InitiatorHandle

This property returns the initiator handle for the initiator.

IDL Function Prototype

```
HRESULT InitiatorHandle(
  [out, retval] long * InitHdl
);
```

C/C++ Syntax

HRESULT get_InitiatorHandle(long * InitHdl);

Parameters

InitHdl

InitHdl will hold the initiator handle for the initiator. The initiator handle is not needed if you are using the Designer COM API but it is needed if you want to call initiator functions defined in the Designer C API - i3idtoolreg.h. Most initiator methods in the Designer C API require the caller to pass in a handle to identify the initiator. The example is an initiator method defined in the Designer C API. The handle returned from this property would be used for the I3IDInitHandle parameter.

C++ Example

```
IdToolRetCode_t REG_DLL I3IDSetInitRequiresLicenseComponent(
    I3IDInitHandle_t initHdl,
    const wchar_t * LicenseComponentName
);
```

II3IDInitiator2::IsConfigurableFromXML Property

get_IsConfigurableFromXML

Returns whether or not the initiator supports configuration from XML.

IDL Function Prototype

```
HRESULT IsConfigurableFromXML(
  [out, retval] VARIANT_BOOL * ConfigurableFromXML
);
```

C/C++ Syntax

HRESULT get_IsConfigurableFromXML(VARIANT_BOOL * ConfigurableFromXML);

Parameters

ConfigurableFromXML

The return value is True if the initiator can be configured from XML; otherwise False.

II3IDInitiator2::IsParameterCollectionAvailable Property

get_IsParameterCollectionAvailable

Returns whether or not the initiator can return parameter collections.

IDL Function Prototype

HRESULT IsParameterCollectionAvailable(

  [out, retval] VARIANT_BOOL * ParametersAvailable

);

C/C++ Syntax

HRESULT get_IsParameterCollectionAvailable(VARIANT_BOOL * ParametersAvailable);

Parameters

ParametersAvailable

Returns True if the inditiator can return parameter collections; otherwise False.

II3IDInitiator2::LicenseComponentName Property

get_LicenseComponentName

Returns the license component name that the initiator requires.

IDL Function Prototype

HRESULT LicenseComponentName(

  [out, retval] BSTR * ComponentName

);

C/C++ Syntax

HRESULT get_LicenseComponentName(BSTR * ComponentName);

Parameters

ComponentName

Both tools and initiators can register that they require certain licensed components for users to be able to use them in Designer. This method returns the license components that are registered for the initiator.

---

put_LicenseComponentName

Sets the license object component to use to verify that an initiator is licensed. You need to call this during the registration phase of the initiator and before it is committed.

IDL Function Prototype

HRESULT LicenseComponentName(

  [in] BSTR ComponentName

);

C/C++ Syntax

HRESULT put_LicenseComponentName(BSTR ComponentName);

Parameters

ComponentName

The ComponentName is the license feature / component that needs to be installed on the server for the handlers with this initiator to be publishable.

Interaction Designer and EICPublisher will not let users publish a handler if the handler contains non-licensed tools or initiator.

Note: License checks can only be done when Designer users have a valid Notifier connection. In the event that there is no Notifier connection, Designer will consider the initiator to be licensed.

II3IDInitiatorAddOn Interface

II3IDInitiatorAddOn::Register Method

Synopsis

Called when an instance of a initiator is being registered by Interaction Designer.

IDL Function Prototype

HRESULT Register(

  [in] II3IDInitiator * Initiator

);

C/C++ Syntax

HRESULT Register(II3IDInitiator * Initiator);

Parameters

Initiator

Initiator is an input parameter of type II3IDInitiator.

II3IDInitiatorAddOn::Unregister Method

Synopsis

Called when an instance of a initiator is being unregistered by Interaction Designer.

IDL Function Prototype

HRESULT Unregister(

  [in] II3IDInitiator * Initiator

);

C/C++ Syntax

HRESULT Unregister(II3IDInitiator * Initiator);

Parameters

Initiator

Initiator is an input parameter of type II3IDInitiator.

II3IDInitiatorEvent Interface

II3IDInitiatorEvent::Designer Property

get_Designer

Returns an Interaction Designer interface pointer.

IDL Function Prototype

HRESULT Designer(

  [out, retval] II3ID ** ID

);

C/C++ Syntax

HRESULT get_Designer(II3ID ** ID);

Parameters

ID

The return value is an II3ID interface pointer.

II3IDInitiatorEvent::Label Property

get_Label

Returns the label for the initiator event.

IDL Function Prototype

HRESULT Label(

  [out, retval] BSTR * EventLabel

);

C/C++ Syntax

HRESULT get_Label(BSTR * EventLabel);

Parameters

EventLabel

The string returned is the label for the initiator event.

II3IDInitiatorEvent::Name Property

get_Name

Returns the name of the initiator event.

IDL Function Prototype

HRESULT Name(

  [out, retval] BSTR * EventName

);

C/C++ Syntax

HRESULT get_Name(BSTR * EventName);

Parameters

EventName

The return value is a string that contains the name of the initiator event.

II3IDInitiatorEvents Interface
II3IDInitiatorEvents::Add Method

Synopsis

Adds an Event to the initiator events collection.

IDL Function Prototype

HRESULT Add(

  [in] BSTR Name,

  [in] BSTR Label,

  [in, optional, defaultvalue(-1)] long Index,

  [out, retval] II3IDInitiatorEvent ** InitiatorEvent

);

C/C++ Syntax

HRESULT Add(BSTR Name, BSTR Label, long Index, II3IDInitiatorEvent ** InitiatorEvent);

Parameters

Name

The object type name.

Label

The label associated with this initiator.

Index

Optional index number used to insert the event into the collection.

InitiatorEvent

An II3IDInitiatorEvent event notification object is returned.

II3IDInitiatorEvents::Item Method

Synopsis

Retrieves a step by its index in the events collection.

IDL Function Prototype

```
HRESULT Item(
  [in] long EventIndex,
  [out, retval] II3IDInitiatorEvent ** InitiatorEvent
);
```

C/C++ Syntax

```
HRESULT Item(long EventIndex, II3IDInitiatorEvent ** InitiatorEvent);
```

Parameters

EventIndex

The index number of the item in the collection to be retrieved.

InitiatorEvent

An II3IDInitiatorEvent object is returned.

II3IDInitiatorEvents::Remove Method

Synopsis

Removes an initiator object from the events collection using the specified index number.

IDL Function Prototype

```
HRESULT Remove(
  [in] long InitiatorEventIndex
);
```

C/C++ Syntax

```
HRESULT Remove(long InitiatorEventIndex );
```

Parameters

InitiatorEventIndex

The index number of the item to remove from the collection.

II3IDInitiatorEvents::Count Property

get_Count

Returns the total number of items in the events collection.

IDL Function Prototype

```
HRESULT Count(
  [out, retval] long * returnCount
);
```

C/C++ Syntax

```
HRESULT get_Count(long * returnCount);
```

Parameters

returnCount

The number of items in the collection.

II3IDInitiatorObjectID Interface

II3IDInitiatorObjectID::Designer Property

get_Designer

Returns an Interaction Designer interface pointer.

IDL Function Prototype

```
HRESULT Designer(
  [out, retval] II3ID ** ID
);
```

C/C++ Syntax

```
HRESULT get_Designer( II3ID ** ID);
```

Parameters

ID

The return value is an II3ID interface pointer.

II3IDInitiatorObjectID::ID Property

get_ID

Returns the ID for the initiator object ID.

IDL Function Prototype

```
HRESULT ID(
  [out, retval] BSTR * ObjectID
);
```

C/C++ Syntax

```
HRESULT get_ID(BSTR * ObjectID);
```

Parameters

ObjectID

The string returned contains the initiator's object ID.

II3IDInitiatorObjectID::Label Property

get_Label

Returns the label for the initiator event.

IDL Function Prototype

HRESULT Label(

  [out, retval] BSTR * ObjectLabel

);

C/C++ Syntax

HRESULT get_Label(BSTR * ObjectLabel);

Parameters

ObjectLabel

The return value is a string containing the initiator object label.

II3IDInitiatorObjectIDs Interface

II3IDInitiatorObjectIDs::Add Method

Synopsis

Adds an ObjectID to the initiator object ID's list at the specified index position. Items are appended to the list if the Index contains a value of -1.

IDL Function Prototype

HRESULT Add(

  [in] BSTR ID,

  [in] BSTR Label,

  [in, optional, defaultvalue(-1)] LONG Index,

  [out, retval] II3IDInitiatorObjectID ** InitiatorObject

);

C/C++ Syntax

HRESULT Add(BSTR ID, BSTR Label, LONG Index, II3IDInitiatorObjectID ** InitiatorObject);

Parameters

ID

This string should contain an Object ID.

Label

This string contains the label for this initiator object.

Index

This index number determines where the item is added to the collection. Specify -1 to append the record to the end of the list.

InitiatorObject

The return value is an II3IDInitiatorObjectID initiator ID object.

II3IDInitiatorObjectIDs::Item Method

Synopsis

Retrieves a step by its index in the object ID's collection.

IDL Function Prototype

```
HRESULT Item(
  [in] long IDIndex,
  [out, retval] II3IDInitiatorObjectID ** InitiatorObject
);
```

C/C++ Syntax

```
HRESULT Item(long IDIndex, II3IDInitiatorObjectID ** InitiatorObject);
```

Parameters

IDIndex

Index number of the item to retrieve from the collection.

InitiatorObject

The return value is an II3IDInitiatorObjectID object.

II3IDInitiatorObjectIDs::Remove Method

Synopsis

Removes an initiator object ID from the list.

IDL Function Prototype

```
HRESULT Remove(
  [in] long InitiatorObjectIDIndex
);
```

C/C++ Syntax

```
HRESULT Remove(long InitiatorObjectIDIndex );
```

Parameters

InitiatorObjectIDIndex

The index number of the item to remove from the collection.

II3IDInitiatorObjectIDs::Count Property

get_Count

Returns the number of items in the collection.

IDL Function Prototype

HRESULT Count(

  [out, retval] long * returnCount

);

C/C++ Syntax

HRESULT get_Count(long * returnCount);

Parameters

returnCount

The count of items in the collection.

II3IDInitiators Interface

II3IDInitiators::Item Method

Synopsis

Returns an initiator from the Initiator collection specified by its index.

IDL Function Prototype

HRESULT Item(

  [in] long TypeIndex,

  [out, retval] II3IDInitiator ** theInitiator

);

C/C++ Syntax

HRESULT Item(long TypeIndex, II3IDInitiator ** theInitiator);

Parameters

TypeIndex

The index number of the item to retrieve from the collection.

theInitiator

The return value is an II3IDInitiator object.

II3IDInitiators::QueryByName Method

Synopsis

Returns an initiator from the Initiator collection specified by the initiator's name and Module

IDL Function Prototype

HRESULT QueryByName(

[in] BSTR InitiatorName,

[in, optional] BSTR ModuleName,

[out, retval] II3IDInitiator ** theInitiator

);

C/C++ Syntax

HRESULT QueryByName(BSTR InitiatorName, BSTR ModuleName, II3IDInitiator ** theInitiator);

Parameters

InitiatorName

The name of the initiator

ModuleName

ModuleName is an optional string input parameter that identifies the module associated with the named initiator.

theInitiator

The return value is an II3IDInitiator object.

II3IDInitiators::RegisterInitiator Method

Synopsis

Add a COM based tool to the collection and return its interface pointer. The ProgID will be used as the tool's Module name.

IDL Function Prototype

HRESULT RegisterInitiator(

[in] VARIANT InitiatorAddOnEventSink,

[in] BSTR InitLabel,

[in] BSTR InitModule,

[in] BSTR InitName,

[in] BSTR InitDescription,

[in] BSTR ObjectType,

[in] BSTR ObjectTypeLabel,

[in] BSTR RuntimeDLLName,

[in] BSTR RuntimeDLLFuncNme,

[in] long nbrParms,

[in] VARIANT_BOOL AllowAllEvents,

[in] VARIANT_BOOL AllowAllObjectIDs,

[in, optional] BSTR InitVersion,

[in, optional, defaultvalue(0)] BSTR HelpFileName,

[in, optional, defaultvalue(0)] long HelpFileContext,

[out, retval] II3IDIni

C/C++ Syntax

HRESULT RegisterInitiator(VARIANT InitiatorAddOnEventSink, BSTR InitLabel, BSTR InitModule, BSTR InitName, BSTR InitDescription, BSTR ObjectType, BSTR ObjectTypeLabel, BSTR RuntimeDLLName, BSTR RuntimeDLLFuncNme, long nbrParms, VARIANT_BOOL AllowAllEvents, VARIANT_BOOL AllowAllObjectIDs, BSTR InitVersion, BSTR HelpFileName, long HelpFileContext, II3IDIni

Parameters

InitiatorAddOnEventSink

InitiatorAddOnEventSink specifies the object that Interaction Designer should call back on with events specified in the II3IDInitiatorAddOn interface. The variant should be one of the following:

VT_EMPTY / VT_NULL - specify when you don't want any II3IDInitiatorAddOn callbacks to occur.

VT_BSTR - the bstrVal contains a ProgID of a COM object that Designer should create that implements the II3IDInitiatorAddOn interface.

VT_UNKNOWN - punkVal should be an IUnknown interface pointer that Designer can query for an II3IDInitiatorAddOn interface pointer.

VT_DISPATCH - pdispVal should be an IDispatch interface pointer that Designer can query for an II3IDInitiatorAddOn interface pointer.

InitLabel

The label for the initiator. The label should be localized because it is displayed to the Interaction Designer user. For instance, when a user clicks on the File | Change Initiator menu item, the label is displayed in the list of initiator choices.

InitModule

The label for the initiator. The label should be localized because it is displayed to the Interaction Designer user. For instance, when a user clicks on the File | Change Initiator menu item, the label is displayed in the list of initiator choices.

InitName

The unchanging name of the initiator.

Note: The module name/initiator pair name needs to be unique among all initiators that are registered in Designer. Interaction Designer will fail to register an initiator if there is another initiator already registered with the same module/name combination.

InitDescription

This is a localized description of what your initiator is used for.

ObjectType

Type of entity causing notification. This is the Notifier object type string and should not be localized.

ObjectTypeLabel

The GUI label for the object that should be internationalized.

RuntimeDLLName

The name of the DLL that contains the 'RuntimeDLLFuncName' function.

RuntimeDLLFuncName

The name of the function inside of the 'RuntimeDLLName' DLL to call when the notification is received.

nbrParms

The number of output parameters for this initiator. For an initiator, all outputs parameters are variables.

AllowAllEvents

This Boolean value determines whether or not Interaction Designer will allow a handler author who is editing the initiator's properties to select '{all}' as a notification event.

AllowAllObjectIDs

This Boolean determines whether or not Interaction Designer will allow a handler author who is editing the initiator's properties to select '{all}' as an object ID.

InitVersion

A string that you can set to indicate the version of the initiator.

HelpFileName

This optional input parameter specifies the name of the Windows help file that contains a topic describing this initiator.

HelpFileContext

This optional parameter is Help Context ID number of the topic in the Windows help file that describes this initiator.

NewInit

The return value is an II3IDInitiator object.

II3IDInitiators::Count Property

get_Count

Returns the number of items in the collection of currently loaded initiators.

IDL Function Prototype

```
HRESULT Count(
  [out, retval] long * returnCount
);
```

C/C++ Syntax

```
HRESULT get_Count(long * returnCount);
```

Parameters

returnCount

The total count of items in the collection.

II3IDMenuItem Interface

II3IDMenuItem::AddMenuItem Method

Synopsis

Adds a child menu to this menu item in Interaction Designer's Utility Menu.

IDL Function Prototype

```
HRESULT AddMenuItem(
  [in] BSTR MenuItemText,
  [in, optional, defaultvalue(0)] VARIANT_BOOL Checked,
  [in, optional, defaultvalue(0)] VARIANT_BOOL Enabled,
  [in, optional] VARIANT MenuEventNotifier,
  [out, retval] II3IDMenuItem ** NewMenu
);
```

C/C++ Syntax

HRESULT AddMenuItem(BSTR MenuItemText, VARIANT_BOOL Checked, VARIANT_BOOL Enabled, VARIANT MenuEventNotifier, II3IDMenuItem ** NewMenu);

Parameters

MenuItemText

The text that will appear in the new menu item.

Checked

Set this Boolean value True to check the menu item. Specify False to uncheck the item.

Enabled

Set this Boolean value True to enable the menu item. Specify False to disable.

MenuEventNotifier

Specifies what object Interaction Designer should call to process a menu event. This can be a variant that contains an IDispatch pointer, an IUnknown pointer, or a BSTR. If you specify a BSTR in the VARIANT, Interaction Designer will treat that string as a ProgId, create the object using that ProgId and QueryInterface the created object for an II3IDMenuEvents interface pointer. For an IDispatch or IUnknown pointer, Interaction Designer will call QueryInterface on that pointer for the II3IDMenuEvents interface.

NewMenu

The return value is a new II3IDMenuItem object.

II3IDMenuItem::AddSeparator Method

Synopsis

Adds a separator child menu to this menu item in Interaction Designer's Utility Menu.

IDL Function Prototype

HRESULT AddSeparator(

  [out, retval] II3IDMenuItem ** NewMenu

);

C/C++ Syntax

HRESULT AddSeparator(II3IDMenuItem ** NewMenu);

Parameters

NewMenu

The return value is an II3IDMenuItem object.

II3IDMenuItem::Designer Property

get_Designer

Returns an Interaction Designer interface pointer.

IDL Function Prototype

HRESULT Designer(

  [out, retval] II3ID ** ID

);

C/C++ Syntax

HRESULT get_Designer(II3ID ** ID);

Parameters

ID

The return value is an II3ID interface pointer.

II3IDMenuItem::ID Property

get_ID

The ID assigned to the menu item by Interaction Designer.

IDL Function Prototype

HRESULT ID(

  [out, retval] LONG * MenuID

);

C/C++ Syntax

HRESULT get_ID(LONG * MenuID);

Parameters

MenuID

The return value is the ID number that Interaction Designer assigned to the menu item.

II3IDMenuItem::IsChecked Property

get_IsChecked

Indicates whether or not the menu item is checked.

IDL Function Prototype

HRESULT IsChecked(

  [out, retval] VARIANT_BOOL * MenuIsChecked

);

C/C++ Syntax

HRESULT get_IsChecked(VARIANT_BOOL * MenuIsChecked);

Parameters

MenuIsChecked

True if the menu item is checked; otherwise False.

---

put_IsChecked

This property checks or unchecks a menu item.

IDL Function Prototype

HRESULT IsChecked(

  [in] VARIANT_BOOL MenuIsChecked

);

C/C++ Syntax

HRESULT put_IsChecked(VARIANT_BOOL MenuIsChecked);

Parameters

MenuIsChecked

To check the menu item, specify True. To uncheck the menu item, specify False.

II3IDMenuItem::IsEnabled Property

get_IsEnabled

Indicates whether or not the menu item is enabled.

IDL Function Prototype

HRESULT IsEnabled(

  [out, retval] VARIANT_BOOL * MenuIsEnabled

);

C/C++ Syntax

HRESULT get_IsEnabled(VARIANT_BOOL * MenuIsEnabled);

Parameters

MenuIsEnabled

This property returns True if the menu item is enabled, or False if the item is disabled.

---

put_IsEnabled

Sets whether or not the menu item is enabled.

IDL Function Prototype

HRESULT IsEnabled(

[in] VARIANT_BOOL MenuIsEnabled

);

C/C++ Syntax

HRESULT put_IsEnabled(VARIANT_BOOL MenuIsEnabled);

Parameters

MenuIsEnabled

Specify True to enable the menu item, or False to disable it.

II3IDMenuItem::IsSeparator Property

get_IsSeparator

Indicates whether or not the menu item is a separator.

IDL Function Prototype

HRESULT IsSeparator(

[out, retval] VARIANT_BOOL * ItemIsASeparator

);

C/C++ Syntax

HRESULT get_IsSeparator(VARIANT_BOOL * ItemIsASeparator);

Parameters

ItemIsASeparator

The return value is True if the menu item is a separator; otherwise False.

II3IDMenuItem::SubItems Property

get_SubItems

Returns a collection of menu items that are children for that menu item in Interaction Designer's Utility Menu.

IDL Function Prototype

HRESULT SubItems(

[out, retval] II3IDMenuItems ** TheSubItems

);

C/C++ Syntax

HRESULT get_SubItems(II3IDMenuItems ** TheSubItems);

Parameters

TheSubItems

The return value is an II3IDMenuItems collection.

II3IDMenuItem::Text Property

get_Text

Returns the text of the menu item in Interaction Designer's Utility Menu.

IDL Function Prototype

HRESULT Text(

  [out, retval] BSTR * MenuText

);

C/C++ Syntax

HRESULT get_Text(BSTR * MenuText);

Parameters

MenuText

The return value is the text displayed in Interaction Designer's pull-down menu.

II3IDMenuItemEvents Interface

II3IDMenuItemEvents::MenuClicked Method

Synopsis

The MenuClicked event gets fired when a user selects a menu item.

IDL Function Prototype

HRESULT MenuClicked(

  [in] II3IDMenuItem * MenuItem,

  [in] II3ID * Designer

);

C/C++ Syntax

HRESULT MenuClicked(II3IDMenuItem * MenuItem, II3ID * Designer);

Parameters

MenuItem

The II3IDMenuItem menu item that was clicked.

Designer

An II3ID interface pointer to Interaction Designer.

II3IDMenuItemEvents::UpdateMenuUI Method

Synopsis

This event gets fired when a user selects a menu item.

IDL Function Prototype

HRESULT UpdateMenuUI(

[in] II3IDMenuItem * MenuItem,

[in] II3ID * Designer

);

C/C++ Syntax

HRESULT UpdateMenuUI(II3IDMenuItem * MenuItem, II3ID * Designer);

Parameters

MenuItem

The II3IDMenuItem menu item selected.

Designer

An II3ID interface pointer to Interaction Designer.

II3IDMenuItems Interface

II3IDMenuItems::Item Method

Synopsis

Retrieves a menu item by its index in the menu items collection.

IDL Function Prototype

HRESULT Item(

[in] long MenuIndex,

[out, retval] II3IDMenuItem ** theMenuItem

);

C/C++ Syntax

HRESULT Item(long MenuIndex, II3IDMenuItem ** theMenuItem);

Parameters

MenuIndex

The index of the menu item to retrieve from the collection.

theMenuItem

The return value is an II3IDMenuItem object.

II3IDMenuItems::Count Property

get_Count

Returns the number of items in the collection of menu items.

IDL Function Prototype

HRESULT Count(

[out, retval] long * returnCount

);

C/C++ Syntax

HRESULT get_Count(long * returnCount);

Parameters

returnCount

The return value is the number of items in the menu collection.

II3IDMenuManager Interface
II3IDMenuManager::AddMenuItem Method

Synopsis

This method adds a menu item to Interaction Designer's Utility Menu.

IDL Function Prototype

HRESULT AddMenuItem(

[in] BSTR MenuItemText,

[in, optional, defaultvalue(0)] VARIANT_BOOL Checked,

[in, optional, defaultvalue(0)] VARIANT_BOOL Enabled,

[in, optional] VARIANT MenuEventNotifier,

[out, retval] II3IDMenuItem ** NewMenu

);

C/C++ Syntax

HRESULT AddMenuItem(BSTR MenuItemText, VARIANT_BOOL Checked, VARIANT_BOOL Enabled, VARIANT MenuEventNotifier, II3IDMenuItem ** NewMenu);

Parameters

MenuItemText

The text that will appear as the menu item.

Checked

This Boolean determines whether or not the menu item is checked.

Enabled

If this value is False, the menu item appears, but is dimmed out.

MenuEventNotifier

Specifies what object Interaction Designer should call to process a menu event. This can be a variant that contains an IDispatch pointer, an IUnknown pointer, or a BSTR. If you specify a BSTR in the VARIANT, Interaction Designer will treat that string as a ProgId, create the object using that ProgId and QueryInterface the created object for an II3IDMenuEvents interface pointer. For an IDispatch or IUnknown pointer, Interaction Designer will call QueryInterface on that pointer for the II3IDMenuEvents interface.

NewMenu

The return value is an II3IDMenuItem object.

II3IDMenuManager::AddSeparator Method

Synopsis

Appends a separator to the Utilities menu.

IDL Function Prototype

```
HRESULT AddSeparator(
  [out, retval] II3IDMenuItem ** NewMenu
);
```

C/C++ Syntax

```
HRESULT AddSeparator( II3IDMenuItem ** NewMenu);
```

Parameters

NewMenu

An II3IDMenuItem object is returned.

II3IDMenuManager::MenuItems Property

get_MenuItems

Retrieves a collection of all of the custom menu items that have been created in Interaction Designer's Utility Menu.

IDL Function Prototype

```
HRESULT MenuItems(
  [out, retval] II3IDMenuItems ** theMenuItems
);
```

C/C++ Syntax

```
HRESULT get_MenuItems( II3IDMenuItems ** theMenuItems);
```

Parameters

theMenuItems

The return value is an II3IDMenuItems collection.

II3IDMenuManager2 Interface

II3IDMenuManager2::AddMenuItem2 Method

Synopsis

Adds a new menu item below the Preferences menu instead of the Utilities menu.

IDL Function Prototype

HRESULT AddMenuItem2(

[in] I3IDMenuLocation ParentMenu,

[in] BSTR MenuItemText,

[in, optional, defaultvalue(0)] VARIANT_BOOL Checked,

[in, optional, defaultvalue(0)] VARIANT_BOOL Enabled,

[in, optional] VARIANT MenuEventNotifier,

[out, retval] II3IDMenuIem ** NewMenu

);

C/C++ Syntax

HRESULT AddMenuItem2(I3IDMenuLocation ParentMenu, BSTR MenuItemText, VARIANT_BOOL Checked, VARIANT_BOOL Enabled, VARIANT MenuEventNotifier, II3IDMenuIem ** NewMenu);

Parameters

ParentMenu

An 3IDMenuLocation constant that identifies which menu will host the new menu item.

MenuItemText

Text that you want the user to see as the menu selection.

Checked

Boolean that determines whether or not this menu item has a checkbox next to it.

Enabled

Boolean that determines whether or not the new menu item is enabled or disabled.

MenuEventNotifier

Specifies what object Interaction Designer should call to process a menu event. This can be a variant that contains an IDispatch pointer, an IUnknown pointer, or a BSTR. If you specify a BSTR in the VARIANT, Interaction Designer will treat that string as a ProgId, create the object using that ProgId and QueryInterface the created object for an II3IDMenuEvents interface pointer. For an IDispatch or IUnknown pointer, Interaction Designer will call QueryInterface on that pointer for the II3IDMenuEvents interface.

NewMenu

The return value is a new II3IDMenuItem object.

II3IDMenuManager2::AddSeparator2 Method

Synopsis

Appends a separator to the Utilities menu.

IDL Function Prototype

HRESULT AddSeparator2(

  [in] I3IDMenuLocation ParentMenu,

  [out, retval] II3IDMenuItem ** NewMenu

);

C/C++ Syntax

HRESULT AddSeparator2(I3IDMenuLocation ParentMenu, II3IDMenuItem ** NewMenu);

Parameters

ParentMenu

The I3IDMenuLocation parent menu object.

NewMenu

The return value is an II3IDMenuItem object.

II3IDMenuManager2::GetMenuItemsForMenu Method

Synopsis

Retrieves a collection of all of the custom menu items that have been created in Interaction Designer's Utility Menu.

IDL Function Prototype

HRESULT GetMenuItemsForMenu(

  [in] I3IDMenuLocation ParentMenu,

  [out, retval] II3IDMenuItems ** theMenuItems

);

C/C++ Syntax

HRESULT GetMenuItemsForMenu(I3IDMenuLocation ParentMenu, II3IDMenuItems ** theMenuItems);

Parameters

ParentMenu

The I3IDMenuLocation object whose items will be returned.

theMenuItems

The return value is a collection of II3IDmenuItems menu items.

II3IDMessage Interface
II3IDMessage::Category Property

get_Category

The category of the message.

IDL Function Prototype

HRESULT Category(

  [out, retval] I3IDMessageCategory * MsgCategory

);

C/C++ Syntax

HRESULT get_Category(I3IDMessageCategory * MsgCategory);

Parameters

MsgCategory

An I3IDMessageCategory value is returned.

II3IDMessage::Description Property

get_Description

Description of information for this message.

IDL Function Prototype

HRESULT Description(

  [out, retval] BSTR * MsgDescription

);

C/C++ Syntax

HRESULT get_Description(BSTR * MsgDescription);

Parameters

MsgDescription

The return value is a string that describes this message.

II3IDMessage::Designer Property

get_Designer

Returns an Interaction Designer interface pointer.

IDL Function Prototype

HRESULT Designer(

  [out, retval] II3ID ** ID

);

C/C++ Syntax

HRESULT get_Designer(II3ID ** ID);

Parameters

ID

The return value is an II3ID interface pointer.

II3IDMessage::Handler Property

get_Handler

Returns a handler pointer to which this message refers. NULL is returned if the handler is no longer available or if the message was not logged for a handler.

IDL Function Prototype

HRESULT Handler(

  [out, retval] II3IDHandler ** MsgHandler

);

C/C++ Syntax

HRESULT get_Handler(II3IDHandler ** MsgHandler);

Parameters

MsgHandler

The return value is a pointer to the II3IDHandler message.

II3IDMessage::HelpFile Property

get_HelpFile

Name of the Windows Help file contains information about this message.

IDL Function Prototype

HRESULT HelpFile(

  [out, retval] BSTR * MsgHelpFile

);

C/C++ Syntax

HRESULT get_HelpFile(BSTR * MsgHelpFile);

Parameters

MsgHelpFile

The name of a Windows help file.

II3IDMessage::HelpFileContext Property

get_HelpFileContext

The context ID number of a Windows help file topic that corresponds to this message.

IDL Function Prototype

HRESULT HelpFileContext(

  [out, retval] long * MsgHelpContext

);

C/C++ Syntax

HRESULT get_HelpFileContext(long * MsgHelpContext);

Parameters

MsgHelpContext

The return value is the context number of a topic in the help file.

II3IDMessage::HelpFileURL Property

get_HelpFileURL

URL where information can be found for this message.

IDL Function Prototype

HRESULT HelpFileURL(

  [out, retval] BSTR * MsgURL

);

C/C++ Syntax

HRESULT get_HelpFileURL(BSTR * MsgURL);

Parameters

MsgURL

The return value is a universal resource locator (web address).

II3IDMessage::Number Property

get_Number

Returns the message number associated with this message. If an error occurs, the error number is returned.

IDL Function Prototype

HRESULT Number(

  [out, retval] long * MsgNumber

);

C/C++ Syntax

HRESULT get_Number(long * MsgNumber);

Parameters

MsgNumber

A message number or error number.

II3IDMessage::Persist Property

get_Persist

Indicates whether or not this message should be persisted with the handler. This only applies to messages that have handler/step context.

IDL Function Prototype

HRESULT Persist(

  [out, retval] VARIANT_BOOL * PersistMessage

);

C/C++ Syntax

HRESULT get_Persist(VARIANT_BOOL * PersistMessage);

Parameters

PersistMessage

True if the message should be persisted with the handler; otherwise False.

II3IDMessage::Severity Property

get_Severity

A number that represents the severity of this message.

IDL Function Prototype

HRESULT Severity(

  [out, retval] long * MsgSeverity

);

C/C++ Syntax

HRESULT get_Severity(long * MsgSeverity);

Parameters

MsgSeverity

The return value is a long number that represents the severity of this message.

II3IDMessage::Source Property

get_Source

Source of this message.

IDL Function Prototype

HRESULT Source(

[out, retval] BSTR * MsgSource

);

C/C++ Syntax

HRESULT get_Source(BSTR * MsgSource);

Parameters

MsgSource

The return value is a string that contains the source for this message.

II3IDMessage::Step Property

get_Step

Returns a step pointer to which this message refers. NULL is returned if the step is no longer available or if the message was not logged for a step.

IDL Function Prototype

HRESULT Step(

[out, retval] II3IDStep ** MsgStep

);

C/C++ Syntax

HRESULT get_Step(II3IDStep ** MsgStep);

Parameters

MsgStep

The return value is an II3IDStep pointer to the step to which this message refers. NULL is returned if the message is no longer available, or if the message was not logged for a step.

II3IDMessage::Topic Property

get_Topic

The topic associated with this message.

IDL Function Prototype

HRESULT Topic(

[out, retval] BSTR * MsgTopic

);

C/C++ Syntax

HRESULT get_Topic(BSTR * MsgTopic);

Parameters

MsgTopic

A string containing the topic associated with this message.

II3IDMessage::Type Property

get_Type

The type of the message.

IDL Function Prototype

HRESULT Type(

  [out, retval] I3IDMessageType * MsgType

);

C/C++ Syntax

HRESULT get_Type(I3IDMessageType * MsgType);

Parameters

MsgType

An I3IDMessageType value.

II3IDMessage::UserName Property

get_UserName

Returns the user associated with the message.

IDL Function Prototype

HRESULT UserName(

  [out, retval] BSTR * UserName

);

C/C++ Syntax

HRESULT get_UserName(BSTR * UserName);

Parameters

UserName

The return value is a string that contains the name of the user.

II3IDMessages Interface

II3IDMessages::Add Method

Synopsis

Adds a message to the Interaction Designer messages collection.

IDL Function Prototype

HRESULT Add(

  [in] long MessageNumber,

[in, optional] long MessageSeverity,

[in, optional] BSTR MessageDescription,

[in, optional] BSTR MessageSource,

[in, optional] BSTR Topic,

[in, optional] IDispatch * MessageContext,

[in, optional, defaultvalue(0)] VAIANT_BOOL PersistMessage,

[in, optional, defaultvalue(0)] I3IDMessageType MessageType,

[in, optional, defaultvalue(0)] I3IDMessageCategory MessageCategory,

[in, optional] BSTR MessageHelpURL,

[in, optional] BSTR MessageHelpFile,

[in, optional] long Messa

C/C++ Syntax

HRESULT Add(long MessageNumber, long MessageSeverity, BSTR MessageDescription, BSTR MessageSource, BSTR Topic, Dispatch * MessageContext, VAIANT_BOOL PersistMessage, I3IDMessageType MessageType, I3IDMessageCategory MessageCategory, BSTR MessageHelpURL, BSTR MessageHelpFile, long Messa

Parameters

MessageNumber

The message number.

MessageSeverity

A number that represents the severity of this message.

MessageDescription

Description of information for this message.

MessageSource

Source of this message.

Topic

The topic of this message.

MessageContext

MessageContext is an optional input parameter of type IDispatch.

PersistMessage

This optional parameter is a Boolean that indicates whether or not this message should be persisted with the handler. It only applies to messages that have handler/step context. Specify True if the message should be persisted with the handler; otherwise False.

MessageType

An I3IDMessageType value

MessageCategory

An I3IDMessageCategory value.

MessageHelpURL

Optional URL pointing to more information about this message.

MessageHelpFile

Optional name of the Windows Help file contains information about this message.

MessageHelpFileContext

The optional context ID number of a Windows help file topic that corresponds to this message.

NewMessage

The return value is an II3IDMessage message object.

II3IDMessages::Item Method

Synopsis

Returns an II3IDMessage object from the messages collection specified by the message index.

IDL Function Prototype

```
HRESULT Item(
  [in] long MessageIndex,
  [out, retval] II3IDMessage ** TheMessage
);
```

C/C++ Syntax

```
HRESULT Item(long MessageIndex, II3IDMessage ** TheMessage);
```

Parameters

MessageIndex

The index number of the II3IDMessage message object to retrieve.

TheMessage

An II3IDMessage object is returned.

II3IDMessages::Remove Method

Synopsis

Removes the message at the specified index from Interaction Designer's messages collection.

IDL Function Prototype

HRESULT Remove(

  [in] long MessageIndex

);

C/C++ Syntax

HRESULT Remove(long MessageIndex);

Parameters

MessageIndex

Index number of the message item to be removed from the collection.

II3IDMessages::RemoveAll Method

Synopsis

Deletes all of the messages from the error collection.

IDL Function Prototype

HRESULT RemoveAll();

C/C++ Syntax

HRESULT RemoveAll();

Parameters

None.

II3IDMessages::Count Property

get_Count

Returns the number of items in the messages collection.

IDL Function Prototype

HRESULT Count(

  [out, retval] long * returnCount

);

C/C++ Syntax

HRESULT get_Count(long * returnCount);

Parameters

returnCount

The total number of items is returned.

II3IDMessages::FilterApplicationContext Property

get_FilterApplicationContext

Returns the II3ID interface pointer associated with this collection (if applicable) This means that the message is associated with Interaction Designer itself.

IDL Function Prototype

HRESULT FilterApplicationContext(

[out, retval] II3ID ** FilterApplicationContext

);

C/C++ Syntax

HRESULT get_FilterApplicationContext(II3ID ** FilterApplicationContext);

Parameters

FilterApplicationContext

The return value is an II3ID interface pointer.

II3IDMessages::FilterHandlerContext Property

get_FilterHandlerContext

Returns the II3IDHandler interface pointer associated with this collection (if applicable).

IDL Function Prototype

HRESULT FilterHandlerContext(

[out, retval] II3IDHandler ** FilterHandlerContext

);

C/C++ Syntax

HRESULT get_FilterHandlerContext(II3IDHandler ** FilterHandlerContext);

Parameters

FilterHandlerContext

An II3IDHandler interface pointer.

II3IDMessages::FilterMsgCategory Property

get_FilterMsgCategory

The message category filter for this collection.

IDL Function Prototype

HRESULT FilterMsgCategory(

[out, retval] I3IDMessageCategory * FilterCategory

);

C/C++ Syntax

HRESULT get_FilterMsgCategory(I3IDMessageCategory * FilterCategory);

Parameters

FilterCategory

An I3IDMessageCategory value is returned.

II3IDMessages::FilterMsgType Property

get_FilterMsgType

The message type filter for this collection.

IDL Function Prototype

HRESULT FilterMsgType(

[out, retval] I3IDMessageType * FilterType

);

C/C++ Syntax

HRESULT get_FilterMsgType(I3IDMessageType * FilterType);

Parameters

FilterType

An I3IDMesssageType value is returned.

II3IDMessages::FilterStepContext Property

get_FilterStepContext

Returns the II3IDStep interface pointer associated with this collection (if applicable).

IDL Function Prototype

HRESULT FilterStepContext(

[out, retval] II3IDStep ** FilterStepContext

);

C/C++ Syntax

HRESULT get_FilterStepContext(II3IDStep ** FilterStepContext);

Parameters

FilterStepContext

The return value is an II3IDStep interface pointer.

II3IDOldStepInfo Interface

II3IDOldStepInfo::Designer Property

get_Designer

Returns an Interaction Designer interface pointer.

IDL Function Prototype

HRESULT Designer(

  [out, retval] II3ID ** ID

);

C/C++ Syntax

HRESULT get_Designer(II3ID ** ID);

Parameters

ID

An II3ID interface pointer is returned.

II3IDOldStepInfo::ExitPaths Property

get_ExitPaths

Returns the exit paths as defined in the old step (if they are not in sync.)

IDL Function Prototype

HRESULT ExitPaths(

  [out, retval] II3IDExitPaths ** OldExitPaths

);

C/C++ Syntax

HRESULT get_ExitPaths(II3IDExitPaths ** OldExitPaths);

Parameters

OldExitPaths

An II3IDExitPaths collection of exit paths is returned.

II3IDOldStepInfo::ExitPathsNextSteps Property

get_ExitPathsNextSteps

Returns a collection of steps to which the exit paths from get_ExitPaths were connected. There is a 1 to 1 correlation here based on the index in the collection.

IDL Function Prototype

HRESULT ExitPathsNextSteps(

  [out, retval] II3IDSteps ** OldExitPathsNextSteps

);

C/C++ Syntax

HRESULT get_ExitPathsNextSteps(II3IDSteps ** OldExitPathsNextSteps);

Parameters

OldExitPathsNextSteps

An II3IDSteps collection is returned.

II3IDOldStepInfo::OldStepInfoType Property

get_OldStepInfoType

Returns which type of out of sync information is present (e.g. unknown, tool, subroutine, initiator, step, etc.)

IDL Function Prototype

HRESULT OldStepInfoType(

[out, retval] I3IDEntityType * OldInfoType

);

C/C++ Syntax

HRESULT get_OldStepInfoType(I3IDEntityType * OldInfoType);

Parameters

OldInfoType

An I3IDEntityType value is returned.

II3IDOldStepInfo::Parameters Property

get_Parameters

Returns the parameters of the step as they were stored in the IHD file.

IDL Function Prototype

HRESULT Parameters(

[out, retval] II3IDParameters ** Parms

);

C/C++ Syntax

HRESULT get_Parameters(II3IDParameters ** Parms);

Parameters

Parms

An II3IDParameters collection of parameter objects is returned.

II3IDOldStepInfo::SourceInitiator Property

get_SourceInitiator

Returns the initiator as it was defined when the step was saved.

IDL Function Prototype

HRESULT SourceInitiator(

  [out, retval] II3IDInitiator ** OldInitiator

);

C/C++ Syntax

HRESULT get_SourceInitiator(II3IDInitiator ** OldInitiator);

Parameters

OldInitiator

The return value is an II3IDInitiator object.

II3IDOldStepInfo::SourceTool Property

get_SourceTool

Returns the tool as it was defined when the step was saved.

IDL Function Prototype

HRESULT SourceTool(

  [out, retval] II3IDTool ** OldTool

);

C/C++ Syntax

HRESULT get_SourceTool(II3IDTool ** OldTool);

Parameters

OldTool

The return value is an II3IDTool object.

II3IDParameter Interface

II3IDParameter::AssignStringLiteralToStringParameter Method

Synopsis

Assign a string literal to a string parameter. This method performs the necessary insertion of backslashes where appropriate. Pass the literal value only without the quotes in the StringLiteral parameter.

IDL Function Prototype

HRESULT AssignStringLiteralToStringParameter(

  [in] BSTR StringLiteral

);

C/C++ Syntax

HRESULT AssignStringLiteralToStringParameter(BSTR StringLiteral);

Parameters

StringLiteral

A literal string value (without quotes) that is to be assigned to a string parameter.

II3IDParameter::ClearExpression Method

Synopsis

Clears the expression currently contained in this parameter.

IDL Function Prototype

HRESULT ClearExpression();

C/C++ Syntax

HRESULT ClearExpression();

Parameters

None.

II3IDParameter::GetStringLiteralFromStringParameter Method

Synopsis

For a string parameter, this method returns the literal value contained within the string parameter. Backslashes are automatically removed.

IDL Function Prototype

HRESULT GetStringLiteralFromStringParameter(

[out, retval] BSTR * StringLiteral

);

C/C++ Syntax

HRESULT GetStringLiteralFromStringParameter(BSTR * StringLiteral);

Parameters

StringLiteral

The literal value contained within the string, without backslashes.

II3IDParameter::Designer Property

get_Designer

Returns an Interaction Designer interface pointer.

IDL Function Prototype

HRESULT Designer(

[out, retval] II3ID ** ID

);

C/C++ Syntax

HRESULT get_Designer(II3ID ** ID);

Parameters

ID

The return value is an II3ID interface pointer.

II3IDParameter::Expression Property

get_Expression

Returns a VARIANT indicating how the value of the parameter is determined at runtime.

IDL Function Prototype

HRESULT Expression(

[out, retval] VARIANT * ExpressionValue

);

C/C++ Syntax

HRESULT get_Expression(VARIANT * ExpressionValue);

Parameters

ExpressionValue

The result of the expression.

---

put_Expression

Defines how the value of the parameter is determined at runtime. The value you supply should look as it does in the expression editor. For string parameters, we do have a II3ID::EscapeExpression method that escapes string values.

IDL Function Prototype

HRESULT Expression(

[in] VARIANT ExpressionValue

);

C/C++ Syntax

HRESULT put_Expression(VARIANT ExpressionValue);

Parameters

ExpressionValue

The value of the expression.

II3IDParameter::ExpressionIsEmpty Property

get_ExpressionIsEmpty

Indicates whether or not an expression exists for this parameter.

IDL Function Prototype

HRESULT ExpressionIsEmpty(

  [out, retval] VARIANT_BOOL * exprIsEmpty

);

C/C++ Syntax

HRESULT get_ExpressionIsEmpty(VARIANT_BOOL * exprIsEmpty);

Parameters

exprIsEmpty

True if an expression exists for this parameter; otherwise False.

II3IDParameter::ExpressionIsLiteral Property

get_ExpressionIsLiteral

Indicates whether the expression that is assigned to this parameter is a string literal.

IDL Function Prototype

HRESULT ExpressionIsLiteral(

  [out, retval] VARIANT_BOOL * exprIsLiteral

);

C/C++ Syntax

HRESULT get_ExpressionIsLiteral(VARIANT_BOOL * exprIsLiteral);

Parameters

exprIsLiteral

Returns True if the expression assigned to this parameter is a string literal; otherwise False.

II3IDParameter::ExpressionIsReadOnly Property

get_ExpressionIsReadOnly

Indicates whether the parameter contains a read-only expression.

IDL Function Prototype

HRESULT ExpressionIsReadOnly(

  [out, retval] VARIANT_BOOL * ExprIsReadOnly

);

C/C++ Syntax

HRESULT get_ExpressionIsReadOnly(VARIANT_BOOL * ExprIsReadOnly);

Parameters

ExprIsReadOnly

True of the expression is read-only; otherwise False.

II3IDParameter::ExpressionIsValid Property

get_ExpressionIsValid

Indicates whether or not the expression contained in this parameter is valid. If the expression is empty, this method returns FALSE.

IDL Function Prototype

HRESULT ExpressionIsValid(

  [out, retval] VARIANT_BOOL * exprIsValid

);

C/C++ Syntax

HRESULT get_ExpressionIsValid(VARIANT_BOOL * exprIsValid);

Parameters

exprIsValid

Returns True if the expression contained in this parameter is valid. False is returned if the express in invalid or empty.

II3IDParameter::ExpressionIsVariable Property

get_ExpressionIsVariable

Indicates whether or not the expression contained in this parameter is a variable.

IDL Function Prototype

HRESULT ExpressionIsVariable(

  [out, retval] VARIANT_BOOL * exprIsVariable

);

C/C++ Syntax

HRESULT get_ExpressionIsVariable(VARIANT_BOOL * exprIsVariable);

Parameters

exprIsVariable

Returns True if the expression is a variable; otherwise False.

II3IDParameter::ExpressionVariable Property

get_ExpressionVariable

If the expression contained in this parameter is a variable, this method returns the variable.

IDL Function Prototype

HRESULT ExpressionVariable(

  [out, retval] II3IDVariable ** exprVariable

);

C/C++ Syntax

HRESULT get_ExpressionVariable(II3IDVariable ** exprVariable);

Parameters

exprVariable

The return value is an II3IDVariable object.

II3IDParameter::IsClone Property

get_IsClone

Returns whether or not this is a parameter generated by the II3IDParameter::Clone method.

IDL Function Prototype

HRESULT IsClone(

  [out, retval] VARIANT_BOOL * ParmIsClone

);

C/C++ Syntax

HRESULT get_IsClone(VARIANT_BOOL * ParmIsClone);

Parameters

ParmIsClone

True if this parameter was generated using the II3IDParameter::Clone method; otherwise False.

II3IDParameter::IsCustom Property

get_IsCustom

This property indicates whether or not this parameter was added in addition to the creator's original parameter set definition.

IDL Function Prototype

HRESULT IsCustom(

  [out, retval] VARIANT_BOOL * ParmIsCustom

);

C/C++ Syntax

HRESULT get_IsCustom(VARIANT_BOOL * ParmIsCustom);

Parameters

ParmIsCustom

True if the original parameters have been amended; otherwise False.

II3IDParameter::ParameterDefinition Property

get_ParameterDefinition

Returns the data definition for the parameter. This can be used to retrieve meta data for the parameter.

IDL Function Prototype

HRESULT ParameterDefinition(

  [out, retval] II3IDParameterDefinition ** theDataDef

);

C/C++ Syntax

HRESULT get_ParameterDefinition(II3IDParameterDefinition ** theDataDef);

Parameters

theDataDef

An II3IDParameterDefinition object is returned.

II3IDParameter::ResourceInfo Property

get_ResourceInfo

Retrieves resource information associated with the parameter.

IDL Function Prototype

HRESULT ResourceInfo(

  [out, retval] VARIANT * ResInfo

);

C/C++ Syntax

HRESULT get_ResourceInfo(VARIANT * ResInfo);

Parameters

ResInfo

The return value is a VARIANT that contains resource information associated with this step.

---

put_ResourceInfo

Sets the resource information associated with the parameter.

IDL Function Prototype

HRESULT ResourceInfo(

  [in] VARIANT ResInfo

);

C/C++ Syntax

HRESULT put_ResourceInfo(VARIANT ResInfo);

Parameters

ResInfo

The input parameter is a VARIANT that contains new resource information to associate with this step.

II3IDParameter::SourceStep Property

get_SourceStep

Returns the step associated with this parameter.

IDL Function Prototype

HRESULT SourceStep(

  [out, retval] II3IDStep ** ParameterStep

);

C/C++ Syntax

HRESULT get_SourceStep(II3IDStep ** ParameterStep);

Parameters

ParameterStep

An II3IDStep object is returned.

II3IDParameterDefinition Interface

II3IDParameterDefinition::SetAsHiddenParameter Method

Synopsis

Implements functionality defined in I3IDToolReg.h's I3IDAddToolHiddenParameter method.

This method adds a run-time string parameter that is hidden from the user. The tool can use this as a way to "pass" itself extra information. Suppose, for example, that you want a single C function to support more than one external tool step at IP run-time. To do so, your C function must know the context in which it was used (i.e. which action the user wants to carry out).

This function can be used to achieve this capability. To do so, your DLL must register two different tools with Interaction Designer, but would specify the same C function in both cases.

Then for each of the registered tools, your DLL must add a hidden parameter that indicates which action the registered tool represents. At IP run-time, your C function will receive the value of that hidden parameter as a string input. Based on the value of that input, your function can perform the appropriate action.

IDL Function Prototype

HRESULT SetAsHiddenParameter(

  [in] BSTR HiddenValue

);

C/C++ Syntax

HRESULT SetAsHiddenParameter(BSTR HiddenValue);

Parameters

HiddenValue

HiddenValue contains a string that is passed to the tool's runtime function.

II3IDParameterDefinition::SetAsInputCheckBox Method

Synopsis

Implements functionality defined in I3IDToolReg.h's I3IDAddToolInputCheckBox method.

IDL Function Prototype

HRESULT SetAsInputCheckBox(

  [in, optional] BSTR UILabel

);

C/C++ Syntax

HRESULT SetAsInputCheckBox(BSTR UILabel);

Parameters

UILabel

This is the localized label that should be displayed next to the checkbox on a step properties page in Interaction Designer.

II3IDParameterDefinition::SetAsInputComboBox Method

Synopsis

Implements functionality defined in I3IDToolReg.h's I3IDAddToolInputComboBox method.

IDL Function Prototype

HRESULT SetAsInputComboBox(

  [in] VARIANT TypeSpecifier,

  [in] BSTR UILabel,

  [in, optional, defaultvalue(0)] VARIANT_BOOL Required

);

C/C++ Syntax

HRESULT SetAsInputComboBox(VARIANT TypeSpecifier, BSTR UILabel, VARIANT_BOOL Required);

Parameters

TypeSpecifier

TypeSpecifier denotes the type for the input parameter. TypeSpecifier is a VARIANT that should be one of the three following types:

VT_BSTR - A string of the format '<TypeModule>::<TypeName>' that identifies the type for this parameter. For example, the string "::Integer" could be used to specify a type of Integer. In this case, the module name for the type is empty.

VT_UNKNOWN - The punkVal member should contain an IUnknown pointer to a COM object that implements the II3IDType interface.

VT_DISPATCH - The pdispVal member should contain an IDispatch pointer to a COM object that implements the II3IDType interface.

For VT_UNKNOWN and VT_DISPATCH, 'the COM object that implements the II3IDType interface' means that it is a type object that was retrieved from a method off of an II3IDTypes call such as the II3IDTypes::Item method.

UILabel

This is the localized label that should be displayed next to the combo box on a step properties page in Interaction Designer.

Required

This Boolean tells Interaction Designer if the parameter is required for the tool step. If so, a valid entry must be given to the parameter before the step is published.

II3IDParameterDefinition::SetAsInputMultiLine Method

Synopsis

Implements functionality defined in I3IDToolReg.h's I3IDAddToolInputMultiLine method.

IDL Function Prototype

HRESULT SetAsInputMultiLine(

[in, optional] BSTR UILabel,

[in, optional, defaultvalue(0)] VARIANT_BOOL Required

);

C/C++ Syntax

HRESULT SetAsInputMultiLine(BSTR UILabel, VARIANT_BOOL Required);

Parameters

UILabel

This is the localized label that should be displayed next to the multi-line text edit box on a step properties page in Interaction Designer.

Required

This Boolean parameter tells Interaction Designer if the parameter is required for the tool step. If so, a valid entry must be given to the parameter before the step is publishable.

II3IDParameterDefinition::SetAsOutput Method

Synopsis

Implements functionality defined in I3IDToolReg.h's I3IDAddToolOutput method.

IDL Function Prototype

HRESULT SetAsOutput(

[in] VARIANT TypeSpecifier,

[in] BSTR UILabel,

[in, optional, defaultvalue(0)] VARIANT_BOOL Required

);

C/C++ Syntax

HRESULT SetAsOutput(VARIANT TypeSpecifier, BSTR UILabel, VARIANT_BOOL Required);

Parameters

TypeSpecifier

TypeSpecifier denotes the type for the output parameter. It is a VARIANT that should be one of the three following types:

VT_BSTR - A string of the format <TypeModule>::<TypeName> that identifies the type for this parameter. For example, the string '::Integer' could be used to specify a type of Integer. In this case, the module name for the type is empty.

VT_UNKNOWN - The punkVal member should contain an IUnknown pointer to a COM object that implements the II3IDType interface.

VT_DISPATCH - The pdispVal member should contain an IDispatch pointer to a COM object that implements the II3IDType interface.

For VT_UNKNOWN and VT_DISPATCH, 'the COM object that implements the II3IDType interface' means that it is a type object that was retrieved from a method off of an II3IDTypes call such as the II3IDTypes::Item method.

UILabel

This is the localized label that should be displayed next to the output parameter on a step properties page in Interaction Designer.

Required

This parameter is used to tell Interaction Designer if the parameter is required for the tool step. If so, a valid entry must be given to the parameter before the step is publishable.

II3IDParameterDefinition::DefaultValue Property

get_DefaultValue

Returns the default value for the parameter.

IDL Function Prototype

HRESULT DefaultValue(

[out, retval] BSTR * DefValue

);

C/C++ Syntax

HRESULT get_DefaultValue(BSTR * DefValue);

Parameters

DefValue

This parameter's default value.

---

put_DefaultValue

Sets the default value for the parameter.

IDL Function Prototype

HRESULT DefaultValue(

[in] BSTR DefValue

);

C/C++ Syntax

HRESULT put_DefaultValue(BSTR DefValue);

Parameters

DefValue

New default value for the parameter, which must be convertible to a VT_BSTR.

II3IDParameterDefinition::DefaultValueIsVariable Property

get_DefaultValueIsVariable

Indicates whether or not the default value of the parameter specifies a variable.

IDL Function Prototype

HRESULT DefaultValueIsVariable(

[out, retval] VARIANT_BOOL * DefValIsVariable

);

C/C++ Syntax

HRESULT get_DefaultValueIsVariable(VARIANT_BOOL * DefValIsVariable);

Parameters

DefValIsVariable

True if the default value specifies a variable; otherwise False.

II3IDParameterDefinition::DefaultVariableName Property

put_DefaultVariableName

Assigns the default variable name. May specify BSTR of variable name or IUnknown/IDispatch pointer to existing variable in the handler.

IDL Function Prototype

HRESULT DefaultVariableName(

[in] BSTR DefaultVarName

);

C/C++ Syntax

HRESULT put_DefaultVariableName(BSTR DefaultVarName);

Parameters

DefaultVarName

A string containing the default variable name or an IUnknown or IDispatch pointer to an existing variable in the handler.

II3IDParameterDefinition::Designer Property

get_Designer

Returns an Interaction Designer interface pointer.

IDL Function Prototype

HRESULT Designer(

[out, retval] II3ID ** ID

);

C/C++ Syntax

HRESULT get_Designer(II3ID ** ID);

Parameters

ID

The return value is an II3ID interface pointer.

II3IDParameterDefinition::IsInput Property

get_IsInput

Indicates whether or not the parameter is defined to be input-only.

IDL Function Prototype

HRESULT IsInput(

[out, retval] VARIANT_BOOL * ParmIsInput

);

C/C++ Syntax

HRESULT get_IsInput(VARIANT_BOOL * ParmIsInput);

Parameters

ParmIsInput

True if the parameter is input-only; otherwise False.

II3IDParameterDefinition::IsRequired Property

get_IsRequired

Returns whether or not the parameter is required.

IDL Function Prototype

HRESULT IsRequired(

  [out, retval] VARIANT_BOOL * ParmIsRequired

);

C/C++ Syntax

HRESULT get_IsRequired(VARIANT_BOOL * ParmIsRequired);

Parameters

ParmIsRequired

True if this parameter is a required parameter; otherwise False.

II3IDParameterDefinition::Label Property

get_Label

Returns the label for the parameter definition. This string is localized.

IDL Function Prototype

HRESULT Label(

  [out, retval] BSTR * DefinitionLabel

);

C/C++ Syntax

HRESULT get_Label(BSTR * DefinitionLabel);

Parameters

DefinitionLabel

The localized label for this parameter definition.

---

put_Label

Assigns label text to the parameter definition. This string is localized.

IDL Function Prototype

HRESULT Label(

  [in] BSTR DefinitionLabel

);

C/C++ Syntax

HRESULT put_Label(BSTR DefinitionLabel);

Parameters

DefinitionLabel

Assigns a new label for the parameter definition.

II3IDParameterDefinition::ParameterDefinitionType Property

get_ParameterDefinitionType

Returns what type of entity this parameter definition points to. You can use this information to get to the SourceTool or SourceInitiator for the definition.

IDL Function Prototype

HRESULT ParameterDefinitionType(

  [out, retval] I3IDEntityType * ParmDefType

);

C/C++ Syntax

HRESULT get_ParameterDefinitionType(I3IDEntityType * ParmDefType);

Parameters

ParmDefType

The return value is an II3IDEntityType object.

II3IDParameterDefinition::SourceInitiator Property

get_SourceInitiator

Return an initiator object that wraps the initiator used to create this step.

IDL Function Prototype

HRESULT SourceInitiator(

  [out, retval] II3IDInitiator ** TheInitiator

);

C/C++ Syntax

HRESULT get_SourceInitiator(II3IDInitiator ** TheInitiator);

Parameters

TheInitiator

The return value is an II3IDInitiator object.

II3IDParameterDefinition::SourceTool Property

get_SourceTool

Returns the tool associated with this parameter.

IDL Function Prototype

HRESULT SourceTool(

  [out, retval] II3IDTool ** theTool

);

C/C++ Syntax

HRESULT get_SourceTool(II3IDTool ** theTool);

Parameters

theTool

The return value is an II3IDTool object that is associated with this parameter.

II3IDParameterDefinition::TypeDefinition Property

get_TypeDefinition

Returns more detailed data type information for the parameter definition.

IDL Function Prototype

HRESULT TypeDefinition(

  [out, retval] II3IDType ** DataType

);

C/C++ Syntax

HRESULT get_TypeDefinition(II3IDType ** DataType);

Parameters

DataType

The return value is an II3IDType data type object.

II3IDParameterDefinition2 Interface

II3IDParameterDefinition2::DesignerUIControlInfo Method

Synopsis

This method returns User Interface information about a parameter.

IDL Function Prototype

HRESULT DesignerUIControlInfo(

  [in, out, optional] BSTR * ControlTypeName,

  [in, out, optional] long * ControlHeight,

  [in, out, optional] long * ControlWidth

);

C/C++ Syntax

HRESULT DesignerUIControlInfo(BSTR * ControlTypeName, long * ControlHeight, long * ControlWidth);

Parameters

ControlTypeName

Parameter type.

ControlHeight

Parameter height.

ControlWidth

ControlWidth is reserved for future use.

II3IDParameterDefinition2::SupportsMultipleTypeDefinitions Property

get_SupportsMultipleTypeDefinitions

Returns whether or not the parameter supports multiple types.

IDL Function Prototype

HRESULT SupportsMultipleTypeDefinitions(

  [out, retval] VARIANT_BOOL * SupportsMultipleTypes

);

C/C++ Syntax

HRESULT get_SupportsMultipleTypeDefinitions(VARIANT_BOOL * SupportsMultipleTypes);

Parameters

SupportsMultipleTypes

True if the parameter supports more than one type; otherwise False.

II3IDParameterDefinition2::TabName Property

get_TabName

Returns the tab name associated with this parameter.

IDL Function Prototype

HRESULT TabName(

  [out, retval] BSTR * TheTabName

);

C/C++ Syntax

HRESULT get_TabName(BSTR * TheTabName);

Parameters

TheTabName

The return value is a string containing the name of the tab.

---

put_TabName

Assigns a tab name to the parameter. The tab name should be internationalized. You may not change the tab name for a parameter once the associated tool/initiator has been registered. Assigning a blank string reverts to the default Input/Output tab.

IDL Function Prototype

HRESULT TabName(

  [in] BSTR TheTabName

);

C/C++ Syntax

HRESULT put_TabName(BSTR TheTabName);

Parameters

TheTabName

The string to assign to the parameter's tab name property.

II3IDParameterDefinition2::TypeDefinitions Property

get_TypeDefinitions

Returns a collection of data types that are supported by this parameter.

IDL Function Prototype

HRESULT TypeDefinitions(

  [out, retval] II3IDTypes ** TheTypes

);

C/C++ Syntax

HRESULT get_TypeDefinitions(II3IDTypes ** TheTypes);

Parameters

TheTypes

The return value is a collection of II3IDTypes types.

II3IDParameterDefinitions Interface
II3IDParameterDefinitions::Item Method

Synopsis

Retrieves a parameter definition its index in the II3IDParameterDefinitions collection.

IDL Function Prototype

HRESULT Item(

  [in] long ParameterIndex,

[out, retval] II3IDParameterDefinition ** theParameter

);

C/C++ Syntax

HRESULT Item(long ParameterIndex, II3IDParameterDefinition ** theParameter);

Parameters

ParameterIndex

The index number of the item to retrieve from the collection.

theParameter

The return value is an II3IDParameterDefinition object.

II3IDParameterDefinitions::Count Property

get_Count

Returns the number of items in the collection.

IDL Function Prototype

HRESULT Count(

[out, retval] long * returnCount

);

C/C++ Syntax

HRESULT get_Count(long * returnCount);

Parameters

returnCount

The count of items in the parameter definitions collection.

II3IDParameters Interface
II3IDParameters::AddHiddenParameter Method

Synopsis

Implements functionality defined in I3IDToolReg.h's I3IDAddToolHiddenParameter method.

IDL Function Prototype

HRESULT AddHiddenParameter(

[in] BSTR HiddenValue,

[in, optional, defaultvalue(-1)] long Index,

[out, retval] II3IDParameter ** NewParm

);

C/C++ Syntax

HRESULT AddHiddenParameter(BSTR HiddenValue, long Index, II3IDParameter ** NewParm);

Parameters

HiddenValue

HiddenValue contains a string that is be passed to the tool's runtime function.

Index

The optional index number of the item to be added.

NewParm

The return value is an II3IDParameter object.

II3IDParameters::AddInputCheckBox Method

Synopsis

Implements functionality defined in I3IDToolReg.h's I3IDAddToolInputCheckBox method.

IDL Function Prototype

HRESULT AddInputCheckBox(

[in, optional] BSTR UILabel,

[in, optional, defaultvalue(-1)] long Index,

[out, retval] II3IDParameter ** NewParm

);

C/C++ Syntax

HRESULT AddInputCheckBox(BSTR UILabel, long Index, II3IDParameter ** NewParm);

Parameters

UILabel

The label for the checkbox.

Index

Optional index number of this item in the collection.

NewParm

The return value is an II3IDParameter object.

II3IDParameters::AddInputComboBox Method

Synopsis

Implements functionality defined in I3IDToolReg.h's I3IDAddToolInputComboBox method.

IDL Function Prototype

HRESULT AddInputComboBox(

[in] VARIANT TypeSpecifier,

[in] BSTR UILabel,

[in, optional, defaultvalue(0)] VARIANT_BOOL Required,

[in, optional, defaultvalue(-1)] long Index,

[out, retval] II3IDParameter ** NewParm

);

C/C++ Syntax

HRESULT AddInputComboBox(VARIANT TypeSpecifier, BSTR UILabel, VARIANT_BOOL Required, long Index, II3IDParameter ** NewParm);

Parameters

TypeSpecifier

TypeSpecifier denotes the type for the input parameter. It is a VARIANT that should be one of the three following types:

VT_BSTR - A string of the format "<TypeModule>::<TypeName>" that identifies the type for this parameter. For example, the string "::Integer" could be used to specify a type of Integer. In this case, the module name for the type is empty.

VT_UNKNOWN - The punkVal member should contain an IUnknown pointer to a COM object that implements the II3IDType interface.

VT_DISPATCH - The pdispVal member should contain an IDispatch pointer to a COM object that implements the II3IDType interface.

For VT_UNKNOWN and VT_DISPATCH, "the COM object that implements the II3IDType interface" means that it is a type object that was retrieved from a method off of an II3IDTypes call such as the II3IDTypes::Item method.

UILabel

The localized label that should be displayed next to the combo box on a step properties page in Interaction Designer.

Required

This Boolean tells Interaction Designer if the parameter is required for the tool step. If so, a valid entry must be given to the parameter before the step is publishable.

Index

Index indicates where in the parameter list you would like to insert the parameter. The new parameter will be inserted BEFORE the specified index To append to the current parameter list, specify -1.

The index must be greater than or equal to the registered number of parameters in the parameter list (or -1). The registered count is the number of parameters that are registered by the tool or initiator during its initial registration. That number can be found by checking II3IDParaemters::get_RegisteredCount.

NewParm

The II3IDParameter parameter object that has been created by Interaction Designer.

II3IDParameters::AddInputMultiLine Method

Synopsis

Implements functionality defined in I3IDToolReg.h's I3IDAddToolInputMultiLine method.

IDL Function Prototype

HRESULT AddInputMultiLine(

[in, optional] BSTR UILabel,

[in, optional, defaultvalue(0)] VARIANT_BOOL Required,

[in, optional, defaultvalue(-1)] long Index,

[out, retval] II3IDParameter ** NewParm

);

C/C++ Syntax

HRESULT AddInputMultiLine(BSTR UILabel, VARIANT_BOOL Required, long Index, II3IDParameter ** NewParm);

Parameters

UILabel

The localized label displayed next to the multi-line text box on a step properties page in Interaction Designer.

Required

This Boolean tells Interaction Designer if the parameter is required for the tool step. If so, a valid entry must be given to the parameter before the step is publishable.

Index

Index indicates where in the parameter list you would like to insert the parameter. The new parameter will be inserted BEFORE the specified index. To append to the current parameter list, specify -1.

The index must be greater than or equal to the registered number of parameters in the parameter list (or -1). The registered count is the number of parameters that are registered by the tool/initiator during its initial registration. That number can be found by checking II3IDParaemters::get_RegisteredCount.

NewParm

The return value is an II3IDParameter parameter created by Interaction Designer.

II3IDParameters::AddOutput Method

Synopsis

Implements functionality defined in I3IDToolReg.h's I3IDAddToolOutput method.

IDL Function Prototype

HRESULT AddOutput(

[in] VARIANT TypeSpecifier,

[in, optional] BSTR UILabel,

[in, optional, defaultvalue(0)] VARIANT_BOOL Required,

[in, optional, defaultvalue(-1)] long Index,

[out, retval] II3IDParameter ** NewParm

);

C/C++ Syntax

HRESULT AddOutput(VARIANT TypeSpecifier, BSTR UILabel, VARIANT_BOOL Required, long Index, II3IDParameter ** NewParm);

Parameters

TypeSpecifier

TypeSpecifier denotes the type for the input parameter. It is a VARIANT that should be one of the three following types:

VT_BSTR - A string of the format "<TypeModule>::<TypeName>" that identifies the type for this parameter. For example, the string "::Integer" could be used to specify a type of Integer. In this case, the module name for the type is empty.

VT_UNKNOWN - The punkVal member should contain an IUnknown pointer to a COM object that implements the II3IDType interface.

VT_DISPATCH - The pdispVal member should contain an IDispatch pointer to a COM object that implements the II3IDType interface.

For VT_UNKNOWN and VT_DISPATCH, "the COM object that implements the II3IDType interface" means that it is a type object that was retrieved from a method off of an II3IDTypes call such as the II3IDTypes::Item method.

UILabel

The localized label displayed next to the multi-line text box on a step properties page in Interaction Designer.

Required

This parameter is used to tell Interaction Designer if the parameter is required for the tool step. If so, a valid entry must be given to the parameter before the step is publishable.

Index

Index indicates where in the parameter list you would like to insert the parameter. The new parameter will be inserted BEFORE the specified index. To append to the current parameter list, specify -1.

The index must be greater than or equal to the registered number of parameters in the parameter list (or -1). The registered count is the number of parameters that are registered by the tool or initiator during its initial registration. That number can be found by checking II3IDParaemters::get_RegisteredCount.

NewParm

The return value is the II3IDParameter object created by Interaction Designer.

II3IDParameters::AddParameterFromParameterDefinition Method

Synopsis

Adds a parameter to the collection based off of the parameter definition.

IDL Function Prototype

HRESULT AddParameterFromParameterDefinition(

[in] II3IDParameterDefinition * ParmDef,

[in, optional, defaultvalue(-1)] long Index,

[out, retval] II3IDParameter ** NewParm

);

C/C++ Syntax

HRESULT AddParameterFromParameterDefinition(II3IDParameterDefinition * ParmDef, long Index, II3IDParameter ** NewParm);

Parameters

ParmDef

Meta data about the parameter passed as an II3IDParameterDefinition object.

Index

The optional index number for this item in the collection.

NewParm

The return value is an II3IDParameter parameter object.

II3IDParameters::ApplyToStep Method

Synopsis

Takes a cloned parameter collection and applies it to the original step where the collection came from.

IDL Function Prototype

HRESULT ApplyToStep(

[out, retval] VARIANT_BOOL * ParmsWereModified

);

C/C++ Syntax

HRESULT ApplyToStep(VARIANT_BOOL * ParmsWereModified);

Parameters

ParmsWereModified

The return value is a Boolean that indicates whether parameters were actually changed.

II3IDParameters::Clone Method

Synopsis

Returns a cloned set of parameters for the step.

IDL Function Prototype

HRESULT Clone(

  [out, retval] II3IDParameters ** ClonedParms

);

C/C++ Syntax

HRESULT Clone(II3IDParameters ** ClonedParms);

Parameters

ClonedParms

The return value is an II3IDParameters collection.

II3IDParameters::Item Method

Synopsis

Retrieves a parameter by its index in the collection.

IDL Function Prototype

HRESULT Item(

  [in] long ParameterIndex,

  [out, retval] II3IDParameter ** theParameter

);

C/C++ Syntax

HRESULT Item(long ParameterIndex, II3IDParameter ** theParameter);

Parameters

ParameterIndex

The index number of a parameter in the parameters collection.

theParameter

The return value is an II3IDParameter object.

II3IDParameters::Remove Method

Synopsis

Removes a parameter from the step.

IDL Function Prototype

HRESULT Remove(

  [in] long ParameterIndex

);

C/C++ Syntax

HRESULT Remove(long ParameterIndex);

Parameters

ParameterIndex

The index number of the parameter to remove from the collection.

II3IDParameters::Count Property

get_Count

Returns the number of items in the collection.

IDL Function Prototype

HRESULT Count(

  [out, retval] long * returnCount

);

C/C++ Syntax

HRESULT get_Count(long * returnCount);

Parameters

returnCount

The total count of parameters in the collection.

II3IDParameters::IsClone Property

get_IsClone

Indicates whether or not the parameter collection is a cloned collection.

IDL Function Prototype

HRESULT IsClone(

  [out, retval] VARIANT_BOOL * ParmCollectionIsClone

);

C/C++ Syntax

HRESULT get_IsClone(VARIANT_BOOL * ParmCollectionIsClone);

Parameters

ParmCollectionIsClone

True if the collection was cloned from another; otherwise False.

II3IDParameters::IsModifiable Property

get_IsModifiable

Returns whether or not the parameter collection is modifiable.

IDL Function Prototype

HRESULT IsModifiable(

[out, retval] VARIANT_BOOL * ParmCollectionIsModifiable

);

C/C++ Syntax

HRESULT get_IsModifiable(VARIANT_BOOL * ParmCollectionIsModifiable);

Parameters

ParmCollectionIsModifiable

True if the parameter collection is modifiable; otherwise False.

II3IDParameters::RegisteredCount Property

get_RegisteredCount

Number of parameters defined by the tool. This number may be different than the total number of parameters available in the step.

IDL Function Prototype

HRESULT RegisteredCount(

[out, retval] long * returnCount

);

C/C++ Syntax

HRESULT get_RegisteredCount(long * returnCount);

Parameters

returnCount

The number of parameters defined by the tool, which may be different than the total number of parameters available in the step.

II3IDStep Interface

II3IDStep::ClearInfoStr Method

Synopsis

Clears the info string associated with this step.

IDL Function Prototype

HRESULT ClearInfoStr();

C/C++ Syntax

HRESULT ClearInfoStr();

Parameters

None.

II3IDStep::LogMessage Method

Synopsis

Logs a message for the step in Interaction Designer's messages collection.

IDL Function Prototype

```
HRESULT LogMessage(
  [in] BSTR Message,
  [in] I3IDMessageCategory MessageCategory,
  [in] I3IDMessageType MessageType
);
```

C/C++ Syntax

HRESULT LogMessage(BSTR Message, I3IDMessageCategory MessageCategory, I3IDMessageType MessageType);

Parameters

Message

The text of the message.

MessageCategory

A valid I3IDMessageCategory message category.

MessageType

An I3IDMessageType value that identifies the type of messages logged to the collection.

II3IDStep::ResizeToFitLabel Method

Synopsis

Resizes a step so that the label text will fit.

IDL Function Prototype

HRESULT ResizeToFitLabel(

[in,optional,defaultvalue(512)] long MaxStepWidth

);

C/C++ Syntax

HRESULT ResizeToFitLabel(long MaxStepWidth);

Parameters

MaxStepWidth

This optional parameter is a number that sets the width (in pixels) of this step in the user interface.

II3IDStep::Validate Method

Synopsis

This method returns NULL (or Nothing) if the node is publishable. If the node is not publishable, it returns a collection of II3IDMessages validation messages.

IDL Function Prototype

HRESULT Validate(

[out, retval] II3IDMessages ** ValidationMessages

);

C/C++ Syntax

HRESULT Validate(II3IDMessages ** ValidationMessages);

Parameters

ValidationMessages

If this return value is NULL or Nothing, then the node is publishable. If the node is not publishable, a collection of II3IDMessages validation messages is returned.

II3IDStep::Bottom Property

get_Bottom

Returns the bottom pixel location of this step in the handler.

IDL Function Prototype

HRESULT Bottom(

[out, retval] long * BottomPos

);

C/C++ Syntax

HRESULT get_Bottom(long * BottomPos);

Parameters

BottomPos

The bottom pixel location of this tool step.

II3IDStep::Designer Property

get_Designer

Returns an Interaction Designer interface pointer.

IDL Function Prototype

HRESULT Designer(

  [out, retval] II3ID ** ID

);

C/C++ Syntax

HRESULT get_Designer(II3ID ** ID);

Parameters

ID

The return value is an II3ID interface pointer.

II3IDStep::ExitPaths Property

get_ExitPaths

Returns a collection of the exit paths leaving this step or initiator.

IDL Function Prototype

HRESULT ExitPaths(

  [out, retval] II3IDExitPaths ** theExitPaths

);

C/C++ Syntax

HRESULT get_ExitPaths(II3IDExitPaths ** theExitPaths);

Parameters

theExitPaths

The return value is an II3IDExitPaths collection.

II3IDStep::Handler Property

get_Handler

Returns the handler object that contains this step.

IDL Function Prototype

HRESULT Handler(

  [out, retval] II3IDHandler ** TheHandler

);

C/C++ Syntax

HRESULT get_Handler(II3IDHandler ** TheHandler);

Parameters

TheHandler

The return value is the II3IDHandler object that contains this step.

II3IDStep::Height Property

get_Height

Returns the height of the node in pixels.

IDL Function Prototype

HRESULT Height(

[out, retval] long * NodeHeight

);

C/C++ Syntax

HRESULT get_Height(long * NodeHeight);

Parameters

NodeHeight

The height in pixels.

---

put_Height

Sets the height of the node to the number of pixels specified.

IDL Function Prototype

HRESULT Height(

[in] long NodeHeight

);

C/C++ Syntax

HRESULT put_Height(long NodeHeight);

Parameters

NodeHeight

The new height of the node, in pixels.

II3IDStep::ID Property

get_ID

Returns the Step ID number associated with this step.

IDL Function Prototype

HRESULT ID(

  [out, retval] long * StepID

);

C/C++ Syntax

HRESULT get_ID(long * StepID);

Parameters

StepID

The step's ID number.

II3IDStep::InfoStr Property

get_InfoStr

Returns the information string for the step. This string is passed out to the DLL that registers for the IntermediatePublish callback when EICPublisher is launched.

IDL Function Prototype

HRESULT InfoStr(

  [out, retval] BSTR * InfoString

);

C/C++ Syntax

HRESULT get_InfoStr(BSTR * InfoString);

Parameters

InfoString

The return value is a string containing the information passed to the DLL that registers for the IntermediatePublish callback when EICPublisher is launched.

---

put_InfoStr

Sets the information string on this step. This string is passed out to the DLL that registers for the IntermediatePublish callback when EICPublisher is launched.

IDL Function Prototype

HRESULT InfoStr(

  [in] BSTR InfoString

);

C/C++ Syntax

HRESULT put_InfoStr(BSTR InfoString);

Parameters

InfoString

Assigns a new information string to this step.

II3IDStep::InitNotificationEvent Property

get_InitNotificationEvent

Returns the currently assigned notification event if the step type is an initiator. If the step is not an initiator, an empty string is returned.

IDL Function Prototype

HRESULT InitNotificationEvent(

  [out, retval] BSTR * InitNotifEvent

);

C/C++ Syntax

HRESULT get_InitNotificationEvent(BSTR * InitNotifEvent);

Parameters

InitNotifEvent

Returns the currently assigned notification event if the step is an initiator, or an empty string if the step is not an initiator.

---

put_InitNotificationEvent

When the step type is an initiator, this sets the notification event on this step.

IDL Function Prototype

HRESULT InitNotificationEvent(

  [in] BSTR InitNotifEvent

);

C/C++ Syntax

HRESULT put_InitNotificationEvent(BSTR InitNotifEvent);

Parameters

InitNotifEvent

The notification event you wish to assign to this initiator step.

II3IDStep::InitObjectID Property

get_InitObjectID

Returns the initiator's object ID if the step type is an initiator.

IDL Function Prototype

HRESULT InitObjectID(

  [out, retval] BSTR * InitObjID

);

C/C++ Syntax

HRESULT get_InitObjectID(BSTR * InitObjID);

Parameters

InitObjID

An object ID is returned if this step is an initiator; otherwise an empty string.

---

put_InitObjectID

When the step type is an initiator, this sets the initiator's object ID.

IDL Function Prototype

HRESULT InitObjectID(

  [in] BSTR InitObjID

);

C/C++ Syntax

HRESULT put_InitObjectID(BSTR InitObjID);

Parameters

InitObjID

The object ID number that you wish to assign to this initiator step.

II3IDStep::Label Property

get_Label

Returns the label associated with this step.

IDL Function Prototype

HRESULT Label(

  [out, retval] BSTR * LabelStr

);

C/C++ Syntax

HRESULT get_Label(BSTR * LabelStr);

Parameters

LabelStr

The label for this step.

---

put_Label

Changes the label displayed for this step in the user interface.

IDL Function Prototype

HRESULT Label(

  [in] BSTR LabelStr

);

C/C++ Syntax

HRESULT put_Label(BSTR LabelStr);

Parameters

LabelStr

Changes the value of the label for this step in the user interface.

II3IDStep::Left Property

get_Left

Returns the x-coordinate (left) pixel location of this step in the handler document.

IDL Function Prototype

>HRESULT Left(

  [out, retval] long * LeftPos

);

C/C++ Syntax

HRESULT get_Left(long * LeftPos);

Parameters

LeftPos

The x-coordinate of this step in pixels.

---

put_Left

Assigns the x-coordinate (left) pixel location of this step in the handler document.

IDL Function Prototype

>HRESULT Left(

[in] long LeftPos

);

C/C++ Syntax

HRESULT put_Left(long LeftPos);

Parameters

LeftPos

The new x-coordinate of this step in pixels.

II3IDStep::NextSteps Property

get_NextSteps

Returns a collection of all steps that this step can branch to.

IDL Function Prototype

HRESULT NextSteps(

[out, retval] II3IDStepLinks ** NextSteps

);

C/C++ Syntax

HRESULT get_NextSteps(II3IDStepLinks ** NextSteps);

Parameters

NextSteps

The return value is an II3IDStepLinks collection of step links.

II3IDStep::Notes Property

get_Notes

Returns the description associated with this step.

IDL Function Prototype

HRESULT Notes(

[out, retval] BSTR * DescStr

);

C/C++ Syntax

HRESULT get_Notes(BSTR * DescStr);

Parameters

DescStr

Text from this step's description field.

put_Notes

Assigns text to the description associated with this step.

IDL Function Prototype

HRESULT Notes(

  [in] BSTR DescStr

);

C/C++ Syntax

HRESULT put_Notes(BSTR DescStr);

Parameters

DescStr

Description of this step.

II3IDStep::Parameters Property

get_Parameters

Returns a collection of parameters that are instantiated for this step. You may optionally use this property to return parameters for an internal subroutine step.

IDL Function Prototype

HRESULT Parameters(

  [out, retval] II3IDParameters ** Parms

);

C/C++ Syntax

HRESULT get_Parameters(II3IDParameters ** Parms);

Parameters

Parms

The return value is an II3IDParameters collection of parameters that are instantiated for this step.

II3IDStep::PreviousSteps Property

get_PreviousSteps

Returns an II3IDStepLinks collection of all steps that can branch to this step.

IDL Function Prototype

HRESULT PreviousSteps(

  [out, retval] II3IDStepLinks ** PreviousSteps

);

C/C++ Syntax

HRESULT get_PreviousSteps(II3IDStepLinks ** PreviousSteps);

Parameters

PreviousSteps

The return value is an II3IDStepLinks collection of steps that can branch to this step.

II3IDStep::Right Property

get_Right

Returns the rightmost pixel of this step as it is rendered within the handler document.

IDL Function Prototype

HRESULT Right(

  [out, retval] long * RightPos

);

C/C++ Syntax

HRESULT get_Right(long * RightPos);

Parameters

RightPos

The rightmost location of this step, in pixels.

II3IDStep::SourceInitiator Property

get_SourceInitiator

Returns the initiator object used to create this step.

IDL Function Prototype

HRESULT SourceInitiator(

  [out, retval] II3IDInitiator ** TheInitiator

);

C/C++ Syntax

HRESULT get_SourceInitiator(II3IDInitiator ** TheInitiator);

Parameters

TheInitiator

The return value is an II3IDInitiator initiator object that wraps the initiator used to create this step.

II3IDStep::SourceSubroutine Property

get_SourceSubroutine

Return a subroutine object that wraps the subroutine used to create this step.

IDL Function Prototype

HRESULT SourceSubroutine(

[out, retval] II3IDSubroutine ** TheSubroutine

);

C/C++ Syntax

HRESULT get_SourceSubroutine(II3IDSubroutine ** TheSubroutine);

Parameters

TheSubroutine

The return value is an II3IDSubroutine subroutine used to create this step.

II3IDStep::SourceTool Property

get_SourceTool

Return the tool object used to create this step.

IDL Function Prototype

HRESULT SourceTool(

[out, retval] II3IDTool ** Tool

);

C/C++ Syntax

HRESULT get_SourceTool(II3IDTool ** Tool);

Parameters

Tool

The return value is the II3IDTool tool object used to create this step.

II3IDStep::StepType Property

get_StepType

Returns the step type (Tool, Subroutine, Initiator, etc.) of this step. If the step type is a Tool, you can call get_SourceTool to retrieve the tool object that created this tool step. Similar handling can be used for Initiators and Subroutines.

IDL Function Prototype

HRESULT StepType(

[out, retval] I3IDEntityType * StepEntityType

);

C/C++ Syntax

HRESULT get_StepType(I3IDEntityType * StepEntityType);

Parameters

StepEntityType

An I3IDEntityType value is returned.

II3IDStep::Top Property

get_Top

Returns the y-coordinate (top) pixel location of this step in the handler document.

IDL Function Prototype

HRESULT Top(

  [out, retval] long * TopPos

);

C/C++ Syntax

HRESULT get_Top(long * TopPos);

Parameters

TopPos

The y-coordinate location of this step in pixels.

---

put_Top

Assigns a y-coordinate (Top) pixel location of this step within the handler document. This does not change the height of the node.

IDL Function Prototype

HRESULT Top(

  [in] long TopPos

);

C/C++ Syntax

HRESULT put_Top(long TopPos);

Parameters

TopPos

The y-coordinate location of this step in pixels.

II3IDStep::Width Property

get_Width

Returns the width of this step as it is rendered within the handler document.

IDL Function Prototype

HRESULT Width(

  [out, retval] long * NodeWidth

);

C/C++ Syntax

HRESULT get_Width(long * NodeWidth);

Parameters

NodeWidth

The width of this step in pixels.

---

put_Width

Sets the width of this step as it is rendered within the handler document.

IDL Function Prototype

HRESULT Width(

[in] long NodeWidth

);

C/C++ Syntax

HRESULT put_Width(long NodeWidth);

Parameters

NodeWidth

The width of this step in pixels.

II3IDStep::XML Property

get_XML

Retrieves XML step and variable information from a handler.

Tip: How to access the Variable and Value parameters of an Assignment step

To access the XML property you have to reference MSXML, assign it to a variable and declared it as IXMLDOMElement.

IDL Function Prototype

HRESULT XML(

[out, retval] VARIANT * stepXML

);

C/C++ Syntax

HRESULT get_XML(VARIANT * stepXML);

Parameters

stepXML

The return value is a VARIANT containing XML step and variable information about the step.

II3IDStep2 Interface
II3IDStep2::ConfigureFromXML Method

Synopsis

Configures the contents of the tool step from an XML string. Unused parts of the XML string are returned from the function. The XML format should match the format of XML files exported from Designer.

IDL Function Prototype

HRESULT ConfigureFromXML(

[in] VARIANT XML,

[out, retval] VARIANT * UnprocessedXML

);

C/C++ Syntax

HRESULT ConfigureFromXML(VARIANT XML, VARIANT * UnprocessedXML);

Parameters

XML

An XML string containing statements used to configure the tool step.

UnprocessedXML

Unused parts of the XML string, if any.

II3IDStep2::IsConfigurableFromXML Property

get_IsConfigurableFromXML

Returns whether or not the step supports being able to be configured from XML.

IDL Function Prototype

HRESULT IsConfigurableFromXML(

[out, retval] VARIANT_BOOL * ConfigurableFromXML

);

C/C++ Syntax

HRESULT get_IsConfigurableFromXML(VARIANT_BOOL * ConfigurableFromXML);

Parameters

ConfigurableFromXML

True if this step is XML-configurable; otherwise False.

II3IDStep2::IsMarkedForDeletion Property

get_IsMarkedForDeletion

Returns whether or not the step is marked for deletion. Steps are marked for deletion if II3IDHandler::RemoveStep is called when a handler is still loading or re-synching with the registered tools.

IDL Function Prototype

HRESULT IsMarkedForDeletion(

[out, retval] VARIANT_BOOL * MarkedForDeletion

);

C/C++ Syntax

HRESULT get_IsMarkedForDeletion(VARIANT_BOOL * MarkedForDeletion);

Parameters

MarkedForDeletion

True if the step is marked for deletion; otherwise False.

II3IDStep2::IsParameterCollectionAvailable Property

get_IsParameterCollectionAvailable

Returns whether or not the step is able to return a parameters collection.

IDL Function Prototype

HRESULT IsParameterCollectionAvailable(

[out, retval] VARIANT_BOOL * ParametersAvailable

);

C/C++ Syntax

HRESULT get_IsParameterCollectionAvailable(VARIANT_BOOL * ParametersAvailable);

Parameters

ParametersAvailable

Returns True if this step can return a parameters collection; otherwise False.

II3IDStepEvents Interface

II3IDStepEvents::EditStepProperties Method

Synopsis

This step event is called when a request is made to open the properties dialog for a step. This makes it possible to display a custom property dialog.

IDL Function Prototype

HRESULT EditStepProperties(

[in] II3IDStep * StepToModify,

[in] long ParentHWND,

[out, retval] VARIANT_BOOL * BeenHandled

);

C/C++ Syntax

HRESULT EditStepProperties(II3IDStep * StepToModify, long ParentHWND, VARIANT_BOOL * BeenHandled);

Parameters

StepToModify

The II3IDStep object whose properties will be edited.

ParentHWND

The handle of the parent window.

BeenHandled

The return value is a Boolean flag that indicates whether or not the callback that has been called by Interaction Designer has handled this event. When this value is True, Interaction Designer does not need to perform it's default event processing.

When BeenHandled is False, Interaction Designer displays the standard Edit Properties dialog. When BeenHandled is set to True by the callout, Interaction Designer does not attempt display a dialog to the user, since it is assumed that the callback has already displayed an edit properties dialog for the user.

II3IDStepEvents::GenerateI3Pub Method

Synopsis

Called prior to a step being written to an Intermediate Publish (.i3pub) file.

IDL Function Prototype

HRESULT GenerateI3Pub(

[in] II3IDStep * Step

);

C/C++ Syntax

HRESULT GenerateI3Pub(II3IDStep * Step);

Parameters

Step

An II3IDStep object.

II3IDStepEvents::Publish Method

Synopsis

This step event method is called prior to a step being published.

IDL Function Prototype

HRESULT Publish(

[in] II3IDStep * StepToPublish,

[in] II3IDICServer * Server

);

C/C++ Syntax

HRESULT Publish(II3IDStep * StepToPublish, II3IDICServer * Server);

Parameters

StepToPublish

The II3IDStep object that will be published.

Server

The II3IDICServer server object that the step will be published to.

II3IDStepEvents::StepInserted Method

Synopsis

This step event method is called when an instance of a tool is added to a handler.

IDL Function Prototype

HRESULT StepInserted(

[in] II3IDStep * InsertedStep

);

C/C++ Syntax

HRESULT StepInserted(II3IDStep * InsertedStep);

Parameters

InsertedStep

An II3IDStep step object is returned.

II3IDStepEvents::StepLinked Method

Synopsis

This method is called when the exit path of one step is assigned to another step.

IDL Function Prototype

HRESULT StepLinked(

[in] II3IDStepLink * LinkStepInfo,

[in] VARIANT_BOOL ToolIsPreviousStep

);

C/C++ Syntax

HRESULT StepLinked(II3IDStepLink * LinkStepInfo, VARIANT_BOOL ToolIsPreviousStep);

Parameters

LinkStepInfo

The II3IDStepLink object linked from the exit path.

ToolIsPreviousStep

True if the tool step linked to is a previous step in the handler.

II3IDStepEvents::StepOutOfSync Method

Synopsis

This method is called when inconsistencies are found between the registered tool and the data stored in the .IHD file.

IDL Function Prototype

HRESULT StepOutOfSync(

[in] I3IDOutOfSyncReason ReasonCode,

[in] II3IDStep * CurrentStep,

[in] II3IDOldStepInfo * OutOfSyncInfo,

[out, retval] VARIANT_BOOL * BeenHandled

);

C/C++ Syntax

HRESULT StepOutOfSync(I3IDOutOfSyncReason ReasonCode, II3IDStep * CurrentStep, II3IDOldStepInfo * OutOfSyncInfo, VARIANT_BOOL * BeenHandled);

Parameters

ReasonCode

An I3IDOutOfSyncReason reason code that indicates which part of the step is not in sync with the currently registered tool.

CurrentStep

The II3IDStep step object that is out of synch.

OutOfSyncInfo

This object indicates how the step looked when it was in synch with the currently registered tool.

BeenHandled

The return value is a Boolean flag that indicates whether or not the callback that has been called by Interaction Designer has handled this event. The callback sets BeenHandled to True if the callback has handled the parameter migration for that step. This tells Interaction Designer not to perform its generic parameter migration functions. Likewise, if the callback did not handle the parameter migration, it sets the value of BeenHandled to False so that Interaction Designer will perform its generic parameter migration functions.

II3IDStepEvents::StepToBeRemoved Method

Synopsis

This method is called when a request is made to delete an instance of a tool.

IDL Function Prototype

HRESULT StepToBeRemoved(

  [in] II3IDStep * StepToRemove

);

C/C++ Syntax

HRESULT StepToBeRemoved(II3IDStep * StepToRemove);

Parameters

StepToRemove

The II3IDStep instance of a tool that is to be removed.

II3IDStepEvents::StepUnlinked Method

Synopsis

This method is called when the link between two steps has been removed.

IDL Function Prototype

HRESULT StepUnlinked(

  [in] II3IDStep * PreviousStep,

  [in] II3IDExitPath * PreviousStepExitPath, II3IDStep * NextStep,

  [in] VARIANT_BOOL ToolIsPreviousStepInStepInfo

);

C/C++ Syntax

HRESULT StepUnlinked(II3IDStep * PreviousStep, II3IDExitPath * PreviousStepExitPath, II3IDStep * NextStep, VARIANT_BOOL ToolIsPreviousStepInStepInfo);

Parameters

PreviousStep

The previous II3IDStep step object.

PreviousStepExitPath

The II3IDExitPath exit path of the previous step.

NextStep

The next II3IDStep step object.

ToolIsPreviousStepInStepInfo

The II3IDExitPath exit path of the next step.

II3IDStepEvents::StepUpdated Method

Synopsis

This method is called when a step has been updated.

IDL Function Prototype

HRESULT StepUpdated(

  [in] II3IDStep * UpdatedStep

);

C/C++ Syntax

HRESULT StepUpdated(II3IDStep * UpdatedStep);

Parameters

UpdatedStep

The return value is the II3IDStep object that was updated.

II3IDStepEvents::Validate Method

Synopsis

This method is called by Interaction Designer to validate the step to see if it is publishable. If the step is not publishable, problems are logged to the messages collection of the II3ID interface.

IDL Function Prototype

HRESULT Validate(

  [in] II3IDStep * StepToValidate

);

C/C++ Syntax

HRESULT Validate(II3IDStep * StepToValidate);

Parameters

StepToValidate

The II3IDStep object that you wish to validate.

II3IDStepLink Interface

II3IDStepLink::Designer Property

get_Designer

Returns an Interaction Designer interface pointer.

IDL Function Prototype

HRESULT Designer(

  [out, retval] II3ID ** ID

);

C/C++ Syntax

HRESULT get_Designer(II3ID ** ID);

Parameters

ID

The return value is an II3ID interface pointer.

II3IDStepLink::Handler Property

get_Handler

Returns the handler associated with this step link.

IDL Function Prototype

HRESULT Handler(

  [out, retval] II3IDHandler ** TheHandler

);

C/C++ Syntax

HRESULT get_Handler(II3IDHandler ** TheHandler);

Parameters

TheHandler

The return value is an II3IDHandler handler object.

II3IDStepLink::NextStep Property

get_NextStep

Returns the step that is acting as the 'target' in this step link. The previous step's exit path will visually hook up to this step in Interaction Designer.

IDL Function Prototype

HRESULT NextStep(

  [out, retval] II3IDStep ** NextStep

);

C/C++ Syntax

HRESULT get_NextStep(II3IDStep ** NextStep);

Parameters

NextStep

The return value is the targeted II3IDStep object.

II3IDStepLink::PreviousStep Property

get_PreviousStep

Returns the step that is the 'source' step in this step link.

IDL Function Prototype

HRESULT PreviousStep(

  [out, retval] II3IDStep ** PreviousStep

);

C/C++ Syntax

HRESULT get_PreviousStep(II3IDStep ** PreviousStep);

Parameters

PreviousStep

The return value is the source II3IDStep object.

II3IDStepLink::PreviousStepExitPath Property

get_PreviousStepExitPath

Returns the exit path for the previous step associated in this step link.

IDL Function Prototype

HRESULT PreviousStepExitPath(

  [out, retval] II3IDExitPath ** ExitPath

);

C/C++ Syntax

HRESULT get_PreviousStepExitPath(II3IDExitPath ** ExitPath);

Parameters

ExitPath

The return value is an II3IDExitPath exit path object.

II3IDStepLinks Interface

II3IDStepLinks::Item Method

Synopsis

Retrieves a step link by its index in the step links collection.

IDL Function Prototype

HRESULT Item(

  [in] long StepLinkIndex,

  [out, retval] II3IDStepLink ** theLink

);

C/C++ Syntax

HRESULT Item(long StepLinkIndex, II3IDStepLink ** theLink);

Parameters

StepLinkIndex

The index number of the item to retrieve from the collection.

theLink

The return value is an II3IDStepLink object.

II3IDStepLinks::Count Property

get_Count

Returns the number of items in the collection of step links.

IDL Function Prototype

HRESULT Count(

  [out, retval] long * returnCount

);

C/C++ Syntax

HRESULT get_Count(long * returnCount);

Parameters

returnCount

The number of items in the collection.

II3IDSteps Interface

II3IDSteps::Item Method

Synopsis

Retrieves a step by its index in the steps collection.

IDL Function Prototype

HRESULT Item(

  [in] long StepIndex,

[out, retval] II3IDStep ** theStep

);

C/C++ Syntax

HRESULT Item(long StepIndex, II3IDStep ** theStep);

Parameters

StepIndex

The index number of the item to retrieve from the collection.

theStep

The return value is an II3IDStep step object.

II3IDSteps::QueryByID Method

Synopsis

Retrieves a step by its ID in the steps collection.

IDL Function Prototype

HRESULT QueryByID(

[in] long StepID,

[out, retval] II3IDStep ** theStep

);

C/C++ Syntax

HRESULT QueryByID(long StepID, II3IDStep ** theStep);

Parameters

StepID

The ID of the step to retrieve from the collection.

theStep

The return value is an II3IDStep step object.

II3IDSteps::Count Property

get_Count

Returns the number of items in the collection.

IDL Function Prototype

HRESULT Count(

[out, retval] long * returnCount

);

C/C++ Syntax

HRESULT get_Count(long * returnCount);

Parameters

returnCount

The total number of items in the II3IDSteps collection.

II3IDSubroutine Interface

II3IDSubroutine::Category Property

get_Category

Returns the subroutine category for this subroutine. When the handler is published, this category is used to label a tab in the Subroutine palette.

IDL Function Prototype

HRESULT Category(

[out, retval] BSTR * TheCategory

);

C/C++ Syntax

HRESULT get_Category(BSTR * TheCategory);

Parameters

TheCategory

The return value is a string containing a subroutine category.

II3IDSubroutine::Designer Property

get_Designer

Returns an Interaction Designer interface pointer.

IDL Function Prototype

HRESULT Designer(

[out, retval] II3ID ** ID

);

C/C++ Syntax

HRESULT get_Designer(II3ID ** ID);

Parameters

ID

The return value is an II3ID interface pointer.

II3IDSubroutine::Name Property

get_Name

Returns the name of the subroutine.

IDL Function Prototype

HRESULT Name(

  [out, retval] BSTR * SubroutineName

);

C/C++ Syntax

HRESULT get_Name(BSTR * SubroutineName);

Parameters

SubroutineName

The return value is a string that contains the name of the subroutine.

II3IDSubroutine2 Interface

II3IDSubroutine2::IsConfigurableFromXML Property

get_IsConfigurableFromXML

Indicates whether or not subroutine steps can be configured from XML.

IDL Function Prototype

HRESULT IsConfigurableFromXML(

  [out, retval] VARIANT_BOOL * ConfigurableFromXML

);

C/C++ Syntax

HRESULT get_IsConfigurableFromXML(VARIANT_BOOL * ConfigurableFromXML);

Parameters

ConfigurableFromXML

This property is provided for sake of completeness. It will always return VARIANT_TRUE to indicate that subroutine steps can be configured from XML.

II3IDSubroutine2::IsParameterCollectionAvailable Property

get_IsParameterCollectionAvailable

This property indicates whether parameter collections can be obtained from subroutine steps.

IDL Function Prototype

HRESULT IsParameterCollectionAvailable(

  [out, retval] VARIANT_BOOL * ParametersAvailable

);

C/C++ Syntax

HRESULT get_IsParameterCollectionAvailable(VARIANT_BOOL * ParametersAvailable);

Parameters

ParametersAvailable

This property is included for the sake of completeness. It will always return VARIANT_TRUE to indicate that parameter collections can be obtained from subroutine steps.

II3IDSubroutines Interface
II3IDSubroutines::Item Method

Synopsis

Returns a subroutine from the collection by its index number.

IDL Function Prototype

HRESULT Item(

  [in] long SubroutineIndex,

  [out, retval] II3IDSubroutine ** theSubroutine

);

C/C++ Syntax

HRESULT Item(long SubroutineIndex, II3IDSubroutine ** theSubroutine);

Parameters

SubroutineIndex

The index number of the subroutine object to retrieve from the collection.

theSubroutine

The return value is an II3IDSubroutine subroutine object.

II3IDSubroutines::QueryByName Method

Synopsis

Returns a subroutine from the collection by name.

IDL Function Prototype

HRESULT QueryByName(

  [in] BSTR SubroutineName,

  [out, retval] II3IDSubroutine ** theSubroutine

);

C/C++ Syntax

HRESULT QueryByName(BSTR SubroutineName, II3IDSubroutine ** theSubroutine);

Parameters

SubroutineName

The name of the subroutine to retrieve from the collection.

theSubroutine

The return value is an II3IDSubroutine subroutine object.

II3IDSubroutines::Count Property

get_Count

Returns the number of items in the collection of subroutines.

IDL Function Prototype

HRESULT Count(

  [out, retval] long * returnCount

);

C/C++ Syntax

HRESULT get_Count(long * returnCount);

Parameters

returnCount

Total number of items in the collection.

II3IDSubroutines::IsFiltered Property

get_IsFiltered

Returns whether or not the subroutine collection is filtered. This collection can be filtered when obtaining a subroutine collection from an II3IDCategory pointer.

IDL Function Prototype

HRESULT IsFiltered(

  [out, retval] VARIANT_BOOL * CollectionIsFiltered

);

C/C++ Syntax

HRESULT get_IsFiltered(VARIANT_BOOL * CollectionIsFiltered);

Parameters

CollectionIsFiltered

True if this collection is filtered; otherwise False.

II3IDTool Interface

II3IDTool::Commit Method

Synopsis

Commits a tool object so that it becomes available in Interaction Designer to handler developers.

IDL Function Prototype

HRESULT Commit();

C/C++ Syntax

HRESULT Commit();

Parameters

None.

II3IDTool::RegisterForStepEvents Method

Synopsis

Registers an object to receive step events from Interaction Designer for this tool.

IDL Function Prototype

HRESULT RegisterForStepEvents(

  [in] VARIANT EventNotifier

);

C/C++ Syntax

HRESULT RegisterForStepEvents(VARIANT EventNotifier);

Parameters

EventNotifier

EventNotifier can be a VT_UNKNOWN or VT_DISPATCH pointing to an existing COM object or a VT_BSTR ProgID of an object that implements the II3IDStepEvents interface.

II3IDTool::CategoryName Property

get_CategoryName

Returns the category name associated with this tool. The category name is localized.

IDL Function Prototype

HRESULT CategoryName(

  [out, retval] BSTR * CategoryName

);

C/C++ Syntax

>HRESULT get_CategoryName(BSTR * CategoryName);

Parameters

CategoryName

This string contains the localized category name associated with this tool.

II3IDTool::Description Property

get_Description

Returns the description associated for the tool. This string is localized.

IDL Function Prototype

HRESULT Description(

  [out, retval] BSTR * ToolDescription

);

C/C++ Syntax

HRESULT get_Description(BSTR * ToolDescription);

Parameters

ToolDescription

The localized description of this tool.

II3IDTool::Designer Property

get_Designer

Returns an Interaction Designer interface pointer.

IDL Function Prototype

HRESULT Designer(

  [out, retval] II3ID ** ID

);

C/C++ Syntax

HRESULT get_Designer(II3ID ** ID);

Parameters

ID

The return value is an II3ID interface pointer.

II3IDTool::ExitPaths Property

get_ExitPaths

Returns a collection of the various exit paths defined for the tool.

IDL Function Prototype

HRESULT ExitPaths(

  [out, retval] II3IDExitPaths ** theExitPaths

);

C/C++ Syntax

HRESULT get_ExitPaths(II3IDExitPaths ** theExitPaths);

Parameters

theExitPaths

The return value is a collection of II3IDExitPaths exit paths for the tool.

II3IDTool::HelpContext Property

get_HelpContext

Returns the Windows help context ID number of the help topic that discusses this tool.

IDL Function Prototype

HRESULT HelpContext(

  [out, retval] long * ToolHelpCntxt

);

C/C++ Syntax

HRESULT get_HelpContext(long * ToolHelpCntxt);

Parameters

ToolHelpCntxt

The context ID number of a topic in a Windows help file.

II3IDTool::HelpFile Property

get_HelpFile

Returns the name of the Windows help file associated with the tool.

IDL Function Prototype

HRESULT HelpFile(

  [out, retval] BSTR * ToolHelpFile

);

C/C++ Syntax

HRESULT get_HelpFile(BSTR * ToolHelpFile);

Parameters

ToolHelpFile

The name of a Windows help file, without path information.

II3IDTool::II3IDToolAddOnInstance Property

get_II3IDToolAddOnInstance

Returns a COM interface pointer to the tool registered in Interaction Designer if the ToolSpecifier parameter to RegisterTool identified a COM object that implemented the II3IDToolAddOn interface.

IDL Function Prototype

HRESULT II3IDToolAddOnInstance(

[out, retval] II3IDToolAddOn ** ToolInstance

);

C/C++ Syntax

HRESULT get_II3IDToolAddOnInstance(II3IDToolAddOn ** ToolInstance);

Parameters

ToolInstance

The return value is a II3IDToolAddOn interface pointer if the above-mentioned conditions are met.

II3IDTool::IsCommitted Property

get_IsCommitted

Indicates whether or not this tool has been committed (made available to handler developers in Interaction Designer).

IDL Function Prototype

HRESULT IsCommitted(

[out, retval] VARIANT_BOOL * ToolIsCommitted

);

C/C++ Syntax

HRESULT get_IsCommitted(VARIANT_BOOL * ToolIsCommitted);

Parameters

ToolIsCommitted

True if the tool has been committed; otherwise False.

II3IDTool::IsExternal Property

get_IsExternal

Indicates whether or not this tool is an external tool. External tools are tools created by developers. Internal tools are tools created by Interaction Designer itself.

IDL Function Prototype

HRESULT IsExternal(

[out, retval] VARIANT_BOOL * ToolIsExternal

);

C/C++ Syntax

HRESULT get_IsExternal(VARIANT_BOOL * ToolIsExternal);

Parameters

ToolIsExternal

True if this tool is an external tool created by a developer; False if Designer created this tool.

II3IDTool::Label Property

get_Label

Returns the label for the tool. This string is localized.

IDL Function Prototype

HRESULT Label(

  [out, retval] BSTR * ToolLabel

);

C/C++ Syntax

HRESULT get_Label(BSTR * ToolLabel);

Parameters

ToolLabel

The return value is a string that contains the localized label for the tool.

II3IDTool::ModuleName Property

get_ModuleName

Returns the module name associated with the tool. The module name is not localized.

IDL Function Prototype

HRESULT ModuleName(

  [out, retval] BSTR * ToolModuleName

);

C/C++ Syntax

HRESULT get_ModuleName(BSTR * ToolModuleName);

Parameters

ToolModuleName

The non-localized module name of the tool.

II3IDTool::Name Property

get_Name

Returns the name of the tool. The name is not localized.

IDL Function Prototype

HRESULT Name(

[out, retval] BSTR * ToolName

);

C/C++ Syntax

HRESULT get_Name(BSTR * ToolName);

Parameters

ToolName

The non-localized tool name.

II3IDTool::ParameterDefinitions Property

get_ParameterDefinitions

Returns the parameter definitions (meta data) associated with the tool. Calls to this property to retrieve parameter definitions for an internal intiator or internal tool will fail, rather than return an empty collection.

IDL Function Prototype

HRESULT ParameterDefinitions(

[out, retval] II3IDParameterDefinitions ** ParmList

);

C/C++ Syntax

HRESULT get_ParameterDefinitions(II3IDParameterDefinitions ** ParmList);

Parameters

ParmList

The return value is an II3IDParameterDefinitions collection of parameter definition objects.

II3IDTool::RuntimeDLLName Property

get_RuntimeDLLName

Returns the runtime DLL name.

IDL Function Prototype

HRESULT RuntimeDLLName(

[out, retval] BSTR * ToolRuntimeDLLName

);

C/C++ Syntax

HRESULT get_RuntimeDLLName(BSTR * ToolRuntimeDLLName);

Parameters

ToolRuntimeDLLName

The return value is the runtime DLL name.

II3IDTool::RuntimeFunctionName Property

get_RuntimeFunctionName

Returns the runtime function name.

IDL Function Prototype

HRESULT RuntimeFunctionName(

  [out, retval] BSTR * ToolRuntimeFunctionName

);

C/C++ Syntax

HRESULT get_RuntimeFunctionName(BSTR * ToolRuntimeFunctionName);

Parameters

ToolRuntimeFunctionName

The return value is the runtime function name.

II3IDTool::Version Property

get_Version

Returns the registered version of this tool.

IDL Function Prototype

HRESULT Version(

  [out, retval] BSTR * TheVersion

);

C/C++ Syntax

HRESULT get_Version(BSTR * TheVersion);

Parameters

TheVersion

The return value is a string that contains the version number associated with the registered tool.

II3IDTool::Visible Property

get_Visible

Indicates whether or not the tool is visible to the user.

IDL Function Prototype

HRESULT Visible(

  [out, retval] VARIANT_BOOL * ShowTool

);

C/C++ Syntax

HRESULT get_Visible(VARIANT_BOOL * ShowTool);

Parameters

ShowTool

True if the tool is visible to the user; otherwise False.

---

put_Visible

Sets whether or not the tool is visible to the user

IDL Function Prototype

HRESULT Visible(

  [in] VARIANT_BOOL ShowTool

);

C/C++ Syntax

HRESULT put_Visible(VARIANT_BOOL ShowTool);

Parameters

ShowTool

True makes the tool visible to the user; False makes the tool invisible.

II3IDTool2 Interface
II3IDTool2::RegisterForDebugStepEvents Method

Synopsis

Registers an object to receive debug step events from Interaction Designer for this tool. These step events are fired during debug sessions in Interaction Designer.

IDL Function Prototype

HRESULT RegisterForDebugStepEvents(

  [in] VARIANT DebugEventNotifier

);

C/C++ Syntax

HRESULT RegisterForDebugStepEvents(VARIANT DebugEventNotifier);

Parameters

DebugEventNotifier

The object to register to receive debug step events.

II3IDTool2::RegisterForXMLStepEvents Method

Synopsis

Registers an object to receive XML step events from Interaction Designer for this tool. These step events are fired when configuring a step from XML.

IDL Function Prototype

```
HRESULT RegisterForXMLStepEvents(
  [in] VARIANT XMLEventNotifier
);
```

C/C++ Syntax

```
HRESULT RegisterForXMLStepEvents(VARIANT XMLEventNotifier);
```

Parameters

XMLEventNotifier

The object to be registered.

II3IDTool2::IsConfigurableFromXML Property

get_IsConfigurableFromXML

Returns whether or not the initiator supports configuration from XML.

IDL Function Prototype

```
HRESULT IsConfigurableFromXML(
  [out, retval] VARIANT_BOOL * ConfigurableFromXML
);
```

C/C++ Syntax

```
HRESULT get_IsConfigurableFromXML(VARIANT_BOOL * ConfigurableFromXML);
```

Parameters

ConfigurableFromXML

Returns True if this initiator supports configuration from XML; otherwise False.

II3IDTool2::IsLicensed Property

get_IsLicensed

Indicates whether or not the tool is licensed.

IDL Function Prototype

```
HRESULT IsLicensed(
  [out, retval] VARIANT_BOOL * LicensedState
);
```

C/C++ Syntax

```
HRESULT get_IsLicensed(VARIANT_BOOL * LicensedState);
```

Parameters

LicensedState

True if the tool is licensed; otherwise False.

II3IDTool2::IsParameterCollectionAvailable Property

get_IsParameterCollectionAvailable

Returns whether or not the initiator can return parameter collections.

IDL Function Prototype

HRESULT IsParameterCollectionAvailable(

  [out, retval] VARIANT_BOOL * ParametersAvailable

);

C/C++ Syntax

HRESULT get_IsParameterCollectionAvailable(VARIANT_BOOL * ParametersAvailable);

Parameters

ParametersAvailable

Returns True if the initiator can return parameter collections; otherwise False.

II3IDTool2::LicenseComponentName Property

get_LicenseComponentName

Returns the license component name that the tool requires.

IDL Function Prototype

HRESULT LicenseComponentName(

  [out, retval] BSTR * ComponentName

);

C/C++ Syntax

HRESULT get_LicenseComponentName(BSTR * ComponentName);

Parameters

ComponentName

Both tools and initiators can register that they require certain licensed components for users to be able to use them in Designer. This method returns the license components that are registered for the initiator.

---

put_LicenseComponentName

Sets the license object component to use to verify that a tool is licensed. You need to call this during the registration phase of the tool and before it is committed.

IDL Function Prototype

HRESULT LicenseComponentName(

[in] BSTR ComponentName

);

C/C++ Syntax

HRESULT put_LicenseComponentName(BSTR ComponentName);

Parameters

ComponentName

The ComponentName is the license feature / component that needs to be installed on the server for handlers that have tool steps created from this tool to be publishable.

Interaction Designer and EICPublisher will not let users publish a handler if the handler contains non-licensed tools or initiator.

Note: License checks can only be done when Designer users have a valid Notifier connection. In the event that there is no Notifier connection, Designer will consider the tool to be licensed.

II3IDTool2::ToolHandle Property

get_ToolHandle

Returns the tool handle for the tool

IDL Function Prototype

HRESULT ToolHandle(

[out, retval] long * ToolHdl

);

C/C++ Syntax

HRESULT get_ToolHandle(long * ToolHdl);

Parameters

ToolHdl

The return value is the tool handle, a long integer.

II3IDToolAddOn Interface

II3IDToolAddOn::Register Method

Synopsis

This method is called when an instance of a tool is being registered by Interaction Designer.

IDL Function Prototype

HRESULT Register(

[in] II3IDTool * Tool

);

C/C++ Syntax

HRESULT Register(II3IDTool * Tool);

Parameters

Tool

An II3IDTool tool object.

II3IDToolAddOn::Unregister Method

Synopsis

This method is called when an instance of a tool is being unregistered by Interaction Designer.

IDL Function Prototype

HRESULT Unregister(

[in] II3IDTool * Tool

);

C/C++ Syntax

HRESULT Unregister(II3IDTool * Tool);

Parameters

Tool

The II3IDTool tool object that is being unregistered.

II3IDToolSetAddOn Interface

II3IDToolSetAddOn::Initialize Method

Synopsis

Initializes a new tool object. This method is called immediately after Designer creates the tool set object, but before Designer calls the II3IDToolSetAddOn::InitializeTypes method to initialize data types used by the tool object.

IDL Function Prototype

HRESULT Initialize(

[in] II3ID * Designer

);

C/C++ Syntax

HRESULT Initialize(II3ID * Designer);

Parameters

Designer

An II3ID interface pointer.

II3IDToolSetAddOn::InitializeEnvironment Method

Synopsis

Initializes Designer's toolset environment. This method is called by Interaction Designer after the II3IDToolSetAddOn::InitializeTypes and II3IDToolSetAddOn::InitializeTools methods are called.

IDL Function Prototype

HRESULT InitializeEnvironment(

[in] II3ID * Designer

);

C/C++ Syntax

HRESULT InitializeEnvironment(II3ID * Designer);

Parameters

Designer

An II3ID interface pointer.

II3IDToolSetAddOn::InitializeTools Method

Synopsis

Interaction Designer calls this method after II3IDToolSetAddOn::InitializeTypes to register your Tools.

IDL Function Prototype

HRESULT InitializeTools(

[in] II3ID * Designer

);

C/C++ Syntax

HRESULT InitializeTools(II3ID * Designer);

Parameters

Designer

An II3ID interface pointer.

II3IDToolSetAddOn::InitializeTypes Method

Synopsis

Interaction Designer calls this method when your tool set should initialize its data types.

IDL Function Prototype

HRESULT InitializeTypes(

[in] II3ID * Designer

);

C/C++ Syntax

HRESULT InitializeTypes(II3ID * Designer);

Parameters

Designer

An II3ID interface pointer.

II3IDToolSetAddOn::ShutDown Method

Synopsis

Interaction Designer calls this method when it is preparing to release its interface pointer to the tool set.

IDL Function Prototype

HRESULT ShutDown(

[in] II3ID * Designer

);

C/C++ Syntax

HRESULT ShutDown(II3ID * Designer);

Parameters

Designer

An II3ID interface pointer.

II3IDTools Interface

II3IDTools::Item Method

Synopsis

Returns a tool object from the tool collection by its index.

IDL Function Prototype

HRESULT Item(

[in] long ToolIndex,

[out, retval] II3IDTool ** theTool

);

C/C++ Syntax

HRESULT Item(long ToolIndex, II3IDTool ** theTool);

Parameters

ToolIndex

The index of the item in the tool collection that you want to retrieve.

theTool

An II3IDTool object is returned.

II3IDTools::QueryByName Method

Synopsis

Returns a tool object from the tool collection by its module and name.

IDL Function Prototype

HRESULT QueryByName(

[in] BSTR ToolName,

[out, retval] II3IDTool ** theTool

);

C/C++ Syntax

HRESULT QueryByName(BSTR ToolName, II3IDTool ** theTool);

Parameters

ToolName

The name of the tool that you wish to retrieve.

theTool

An II3IDTool object is returned.

II3IDTools::RegisterTool Method

Synopsis

Add a COM based tool to the collection and return its interface pointer. The ProgID will be used as the tool's Module name.

IDL Function Prototype

HRESULT RegisterTool(

[in] VARIANT ToolAddOnEventSink,

[in] BSTR ToolLabel,

[in] BSTR ToolModuleName,

[in] BSTR ToolName,

[in] BSTR Description,

[in] BSTR Category,

[in] BSTR RuntimeDLLName,

[in] BSTR RuntimeDLLFuncName,long nbrParms,

[in, otional] BSTR ToolVersion,

[in, optional, defaultvalue(0)] BSTR HelpFileName,

[in, optional, defaultvalue(0)] long HelpFileContext,

[out, retval] II3IDTool ** NewTool

);

C/C++ Syntax

HRESULT RegisterTool(VARIANT ToolAddOnEventSink, BSTR ToolLabel, BSTR ToolModuleName, BSTR ToolName, BSTR Description, BSTR Category, BSTR RuntimeDLLName, BSTR RuntimeDLLFuncName,long nbrParms, BSTR ToolVersion, BSTR HelpFileName, long HelpFileContext, II3IDTool ** NewTool);

Parameters

ToolAddOnEventSink

ToolAddOnEventSink specifies the object that Interaction Designer should call back on with events specified in the II3IDToolAddOn interface. The variant should be one of the following:

VT_EMPTY / VT_NULL - specify when you don't want any II3IDToolAddOn callbacks to occur.

VT_BSTR - the bstrVal contains a ProgID of a COM object that Designer should create that implements the II3IDToolAddOn interface.

VT_UNKNOWN - punkVal should be an IUnknown interface pointer that Designer can query for an II3IDToolAddOn interface pointer.

VT_DISPATCH - pdispVal should be an IDispatch interface pointer that Designer can query for an II3IDToolAddOn interface pointer.

ToolLabel

The label for the tool. The label should be localized because it is displayed to the Interaction Designer user for the tool on the tool palette.

ToolModuleName

The unchanging module name for the tool.

ToolName

The unchanging name of the tool.

Note: The module name/initiator pair name needs to be unique among all tools that are registered in Designer. Interaction Designer will fail to register a tool if there is another tool already registered with the same module/name combination.

Description

The localized description for the tool.

Category

The category determines what tab you would like for the tool to appear under on the main Designer window.

RuntimeDLLName

The name of the DLL that contains the 'RuntimeDLLFuncName' function.

RuntimeDLLFuncName

The name of the function to be called in the runtime DLL to be called at runtime to perform the desired functionality of the tool.

nbrParms

The number of registered parameters for instances of this tool.

ToolVersion

A version string that you can assign to the tool. This string does not have to be a number. If you change the registered tool version for a tool, then the II3IDStepEvents::StepOutOfSync method will be called for handlers opened that were saved with earlier versions of your tool.

HelpFileName

The name of a Windows help file that contains information about this tool.

HelpFileContext

The context ID number of a topic in the help file that describes this tool.

NewTool

The return value is an II3IDTool tool object.

II3IDTools::Count Property

get_Count

Returns the number of items in the collection.

IDL Function Prototype

HRESULT Count(

  [out, retval] long * returnCount

);

C/C++ Syntax

HRESULT get_Count(long * returnCount);

Parameters

returnCount

Total number of items in the II3IDTools collection.

II3IDTools::IsFiltered Property

get_IsFiltered

Returns whether or not the tools collection is filtered. This collection can be filtered when obtaining a tool collection from an II3IDCategory pointer.

IDL Function Prototype

HRESULT IsFiltered(

  [out, retval] VARIANT_BOOL * CollectionIsFiltered

);

C/C++ Syntax

HRESULT get_IsFiltered(VARIANT_BOOL * CollectionIsFiltered);

Parameters

CollectionIsFiltered

True if the tools collection is filtered; otherwise False.

II3IDType Interface

II3IDType::Description Property

get_Description

Returns the description for the specified type

IDL Function Prototype

HRESULT Description(

  [out, retval] BSTR * ModuleName

);

C/C++ Syntax

HRESULT get_Description(BSTR * ModuleName);

Parameters

ModuleName

The type's module name is returned as a string.

II3IDType::Designer Property

get_Designer

Returns the module name for the specified data type.

IDL Function Prototype

HRESULT Designer(

  [out, retval] II3ID ** ID

);

C/C++ Syntax

HRESULT get_Designer(II3ID ** ID);

Parameters

ID

The return value is an II3ID interface pointer.

II3IDType::ModuleName Property

get_ModuleName

Returns the module used to define the type

IDL Function Prototype

HRESULT ModuleName(

  [out, retval] BSTR * ModuleName

);

C/C++ Syntax

HRESULT get_ModuleName(BSTR * ModuleName);

Parameters

ModuleName

The name of the module used to define the type.

II3IDType::NativeType Property

get_NativeType

Returns the data type for this parameter.

IDL Function Prototype

HRESULT NativeType(

  [out, retval] I3IDNativeDataType * NativeDataType

);

C/C++ Syntax

HRESULT get_NativeType(I3IDNativeDataType * NativeDataType);

Parameters

NativeDataType

An I3IDNativeDataType value is returned. ID_CUSTOM is returned for non-native data types.

II3IDType::TypeLabel Property

get_TypeLabel

Returns the data type label for this type object. Data type labels are localized.

IDL Function Prototype

HRESULT TypeLabel(

[out, retval] BSTR * LabelName

);

C/C++ Syntax

HRESULT get_TypeLabel(BSTR * LabelName);

Parameters

LabelName

The return value is a string that contains the localized label for this data type.

II3IDType::TypeName Property

get_TypeName

Returns the data type name for the specified type.

IDL Function Prototype

HRESULT TypeName(

[out, retval] BSTR * TypeName

);

C/C++ Syntax

HRESULT get_TypeName(BSTR * TypeName);

Parameters

TypeName

The return value is a string that contains the type name for the specified type.

II3IDTypes Interface

II3IDTypes::Add Method

Synopsis

Adds a new type to the type collection.

IDL Function Prototype

HRESULT Add(

[in] BSTR Label,

[in] BSTR Name,

[in] BSTR Module,

[in, optional] BSTR Description,

[in, optional] BSTR RefCountDLLName,

[in, optional] BSTR RefCountAttachFuncName,

[in, optional] BSTR RefCountDetachFuncName,

[out, retval] II3IDType ** theType

);

C/C++ Syntax

HRESULT Add(BSTR Label, BSTR Name, BSTR Module, BSTR Description, BSTR RefCountDLLName, BSTR RefCountAttachFuncName, BSTR RefCountDetachFuncName, II3IDType ** theType);

Parameters

Label

A label for the new data type.

Name

The name of the new data type.

Module

The module name associated with this data type.

Description

Description of this data type.

RefCountDLLName

Refernce Count DLL Name.

RefCountAttachFuncName

The attached reference count function name.

RefCountDetachFuncName

The detached reference count function name.

theType

A new II3IDType object is returned.

II3IDTypes::Item Method

Synopsis

Returns a type object specified by the index in the types collection.

IDL Function Prototype

HRESULT Item(

[in] long TypeIndex,

[out, retval] II3IDType ** theType

);

C/C++ Syntax

HRESULT Item(long TypeIndex, II3IDType ** theType);

Parameters

TypeIndex

The index number of the item to retrieve from the types collection.

theType

The return value is an II3IDType type object.

II3IDTypes::QueryByName Method

Synopsis

Returns a type object specified by the type name in the types collection.

IDL Function Prototype

HRESULT QueryByName(

[in] BSTR TypeName,

[out, retval] II3IDType ** theType

);

C/C++ Syntax

HRESULT QueryByName(BSTR TypeName, II3IDType ** theType);

Parameters

TypeName

The type name of the type object to retrieve.

theType

The return value is an II3IDType object.

II3IDTypes::Count Property

get_Count

Returns the total number of items in the type collection.

IDL Function Prototype

HRESULT Count(

[out, retval] long * returnCount

);

C/C++ Syntax

HRESULT get_Count(long * returnCount);

Parameters

returnCount

The total number of items in the collection.

II3IDVariable Interface

II3IDVariable::BoundGlobalVarID Property

get_BoundGlobalVarID

Returns the global variable ID that the variable is bound to.

IDL Function Prototype

HRESULT BoundGlobalVarID(

  [out, retval] BSTR * globVarID

);

C/C++ Syntax

HRESULT get_BoundGlobalVarID(BSTR * globVarID);

Parameters

globVarID

The return value is a string that contains the global variable ID.

II3IDVariable::Designer Property

get_Designer

Returns an Interaction Designer interface pointer.

IDL Function Prototype

HRESULT Designer(

  [out, retval] II3ID ** ID

);

C/C++ Syntax

HRESULT get_Designer(II3ID ** ID);

Parameters

ID

An II3ID interface pointer is returned.

II3IDVariable::InitialValue Property

get_InitialValue

Retrieves information about the default value for the variable.

IDL Function Prototype

HRESULT InitialValue(

  [out, retval] VARIANT * InitValue

);

C/C++ Syntax

HRESULT get_InitialValue(VARIANT * InitValue);

Parameters

InitValue

The default value of the variable (if applicable) is returned in a VARIANT.

---

put_InitialValue

Sets the default value for a variable.

IDL Function Prototype

HRESULT InitialValue(

  [in] VARIANT InitValue

);

C/C++ Syntax

HRESULT put_InitialValue(VARIANT InitValue);

Parameters

InitValue

A VARIANT that contains the variable's new default value.

II3IDVariable::IsBoundToGlobal Property

get_IsBoundToGlobal

Returns if the variable is bound to a global.

IDL Function Prototype

HRESULT IsBoundToGlobal(

  [out, retval] VARIANT_BOOL * boundToGlobalVar

);

C/C++ Syntax

HRESULT get_IsBoundToGlobal(VARIANT_BOOL * boundToGlobalVar);

Parameters

boundToGlobalVar

True if the variable is bound to a global; otherwise False.

II3IDVariable::Name Property

get_Name

Returns the name of the variable.

IDL Function Prototype

HRESULT Name(

  [out, retval] BSTR * theName

);

C/C++ Syntax

HRESULT get_Name(BSTR * theName);

Parameters

theName

The variable name.

II3IDVariable::TypeDefinition Property

get_TypeDefinition

Returns data type information about this variable.

IDL Function Prototype

HRESULT TypeDefinition(

  [out, retval] II3IDType ** TypeInfo

);

C/C++ Syntax

HRESULT get_TypeDefinition(II3IDType ** TypeInfo);

Parameters

TypeInfo

The return value is an II3IDType data type object.

II3IDVariable::Value Property

get_Value

Retrieves the value of a variable from a handler that is currently being debugged.

IDL Function Prototype

HRESULT Value(

  [out, retval] VARIANT * Value

);

C/C++ Syntax

HRESULT get_Value(VARIANT * Value);

Parameters

Value

The value of the variable is returned as a VARIANT.

II3IDVariable::XML Property

get_XML

Retrieves an XML doc pointer that contains information about the variable.

IDL Function Prototype

HRESULT XML(

  [out, retval] VARIANT * variableXML

);

C/C++ Syntax

HRESULT get_XML(VARIANT * variableXML);

Parameters

variableXML

The return value is a VARIANT XML document pointer to meta data about the variable.

II3IDVariable2 Interface

II3IDVariable2::RenameVariable Method

Synopsis

Renames a variable to the new name specified. Valid characters for a variable name are [a-zA-Z] for the first character and [0-9_a-zA-Z] for all other characters.

IDL Function Prototype

HRESULT RenameVariable(

  [in] BSTR NewVariableName

);

C/C++ Syntax

HRESULT RenameVariable(BSTR NewVariableName);

Parameters

NewVariableName

The variable's new name.

II3IDVariables Interface

II3IDVariables::AddNativeTypeVariable Method

Synopsis

Adds a native typed variable to the variables collection.

IDL Function Prototype

HRESULT AddNativeTypeVariable(

[in] BSTR NewVariableName,

[in] I3IDNativeDataType VariableType,

[out, retval] II3IDVariable ** TheVariable

);

C/C++ Syntax

HRESULT AddNativeTypeVariable(BSTR NewVariableName, I3IDNativeDataType VariableType, II3IDVariable ** TheVariable);

Parameters

NewVariableName

The name of the new variable.

VariableType

The new variable's data type.

TheVariable

The return value is an II3IDVariable variable object.

II3IDVariables::AddVariable Method

Synopsis

Add a variable of the specified type. VariableType can be either a VT_UNKNOWN or VT_DISPATCH pointer to an existing II3IDType or it can be a VT_BSTR with the following format: <ModuleName>::<TypeName>

IDL Function Prototype

HRESULT AddVariable(

[in] BSTR NewVariableName,

[in] VARIANT VariableType,

[out, retval] II3IDVariable ** TheVariable

);

C/C++ Syntax

HRESULT AddVariable(BSTR NewVariableName, VARIANT VariableType, II3IDVariable ** TheVariable);

Parameters

NewVariableName

The name of the new variable.

VariableType

VariableType can be either a VT_UNKNOWN or VT_DISPATCH pointer to an existing II3IDType or it can be a VT_BSTR with the following format: <ModuleName>::<TypeName>

TheVariable

The return value is an II3IDVariable variable object.

II3IDVariables::Item Method

Synopsis

Retrieves a variable by its index in the variable collection.

IDL Function Prototype

HRESULT Item(

[in] long VaraibleIndex,

[out, retval] II3IDVariable ** theVariable

);

C/C++ Syntax

HRESULT Item(long VaraibleIndex, II3IDVariable ** theVariable);

Parameters

VaraibleIndex

The index number of the item to retrieve from the variables collection.

theVariable

The return value is an II3IDVariable variable object.

II3IDVariables::QueryByName Method

Synopsis

Retrieves a variable by its name in the variable collection.

IDL Function Prototype

HRESULT QueryByName(

[in] BSTR VariableName,

[out, retval] II3IDVariable ** theVariable

);

C/C++ Syntax

HRESULT QueryByName(BSTR VariableName, II3IDVariable ** theVariable);

Parameters

VariableName

The name of the variable to retrieve from the collection.

theVariable

The return value is an II3IDVariable variable object.

II3IDVariables::Count Property

get_Count

Returns the number of items in the collection.

IDL Function Prototype

HRESULT Count(

  [out, retval] long * returnCount

);

C/C++ Syntax

HRESULT get_Count(long * returnCount);

Parameters

returnCount

The total number of items in the collection.

II3IDVariables::XML Property

get_XML

Retrieves an XML document pointer that contains information about the variables.

IDL Function Prototype

HRESULT XML(

  [out, retval] VARIANT * variablesXML

);

C/C++ Syntax

HRESULT get_XML(VARIANT * variablesXML);

Parameters

variablesXML

The return value is a VARIANT that contains an XML document pointer to this variable's meta data.

II3IDXMLStepEvents Interface
II3IDXMLStepEvents::ConfigureFromXML Method

Synopsis

Tools that use custom parameters or exit paths not registered by the tool should implement this method of supporting configuration from XML.

IDL Function Prototype

HRESULT ConfigureFromXML(

[in] II3IDStep * StepToConfigure,

[in] VARIANT XML,

[out, retval] VARIANT_BOOL * BeenHandled

);

C/C++ Syntax

HRESULT ConfigureFromXML(II3IDStep> * StepToConfigure, VARIANT XML, VARIANT_BOOL * BeenHandled);

Parameters

StepToConfigure

The II3IDStep object that you want to configure.

XML

Fully qualified path to the XML file used to configure the II3IDStep object.

BeenHandled

Set return value to VARIANT_TRUE if parameters, exit paths, etc. are set properly.

Data Type Definitions

I3IDEntityType TypeDef

I3IDEntityType is used to return entity type information. For example, the step object used this enumeration to let you know what type of step it is.

Unknown	0x0
Tool	0x1
Subroutine	0x2

Initiator	0x3
Step	0x4
I3IDInternal	0x10000

I3IDParameterDisplayMode TypeDef

Used for dynamically adding parameters to a step.

SetAsNone	0
SetAsOutput	1
SetAsInputComboBox	2
SetAsInputCheckBox	3
SetAsInputMultiLine	4
SetAsHiddenParameter	5

I3IDMessageType TypeDef

Used to indicate the type of message.

MSGTYPE_None	0x00000001	No type (generic). Not recommended for use.
MSGTYPE_Information	0x00000002	Indicates an informational message.
MSGTYPE_Warning	0x00000004	Warning. While not serious, something is possibly in error.
MSGTYPE_Error	0x00000008	There is definitely something wrong here.
MSGTYPE_ANY	0xFFFFFFFF	Used for retrieving collections

I3IDMessageCategory TypeDef

Used to indicate the message category.

MSGCAT_None	0x00000001	No category associated with this message.

MSGCAT_Validate	0x00000002	This error was generated when a II3IDStep::Validate()
MSGCAT_ChangeLog	0x00000004	Used to log a change that has been made
MSGCAT_Migration	0x00000008	Messages logged during a migration of a handler.
MSGCAT_ID	0x8000000	Errors that were generated by Interaction Designer when
MSGCAT_ANY_EXCEPT_ID	0x7FFFFFFF	Used for retrieving collections
MSGCAT_ANY	0xFFFFFFFF	Used for retrieving collections

I3IDOutOfSyncReason TypeDef

I3IDOutOfSyncReason is used when the II3IDStepEvents::StepOutOfSync method is invoked to tell developers what part of the step is not in sync with the currently registered tool.

COMPRESULT_EQUAL	0x1	The tools are equal
COMPRESULT_PARMCHANGED	0x2	At least 1 parm has changed
COMPRESULT_EXITPATHCHANGED	0x4	At least 2 exit paths have changed
COMPRESULT_OBJTYPEDIFF	0x8	The tool object is of a different
COMPRESULT_NOTIFCHANGED	0x10	The notification entries
COMPRESULT_VERSIONCHANGED	0x40	The version for the tool has changed.

I3IDProcessXML TypeDef

Used to tell tools that support constitution from XML what they should configure.

XML_PROCESS_LABEL	0x1
XML_PROCESS_NOTES	0x2
XML_PROCESS_PARAMETERS	0x4

XML_PROCESS_COORDS	0x8
XML_PROCESS_EXITPATHS	0x10
XML_PROCESS_ALL	0xFFFFFFFF

I3IDNativeDataType TypeDef

The internal data types defined in Interaction Designer are listed below. These are used to indicate a variable's typeversion(1.0).

ID_CUSTOM	0x00000000
ID_STRING	0x00000001
ID_INTEGER	0x00000002
ID_DOUBLE	0x00000004
ID_BOOLEAN	0x00000008
ID_DATETIME	0x00000010
ID_DATABASE	0x00000020
ID_DATABASE_CONNECTION	0x00000040
ID_STRING_LIST	0x00000080
ID_INTEGER_LIST	0x00000100
ID_DOUBLE_LIST	0x00000200
ID_BOOLEAN_LIST	0x00000400
ID_DATETIME_LIST	0x00000800
ID_DATABASE_LIST	0x00001000
ID_DATABASE_CONNECTION_LIST	0x00002000

I3IDMenuLocation TypeDef

This is used for menu operations in the COM interface to determine what menu items should be added under/retrieved for.

ID_UTILITIES_MENU	0x00000001
ID_PREFERENCES_MENU	0x00000002

I3IDDebugSessionState TypeDef

This returns the state of the debug session object.

ID_DEBUG_SESSION_WAITINGFORNOTIF	0
ID_DEBUG_SESSION_ATBREAKPOINT	1
ID_DEBUG_SESSION_STEPPING	2
ID_DEBUG_SESSION_RUNNING	3
ID_DEBUG_SESSION_SESSIONCOMPLETED	4

Glossary

IUnknown Interface

Every COM component implements an internal interface named IUnknown. Client applications can use the IUnknown interface to retrieve pointers to the other interfaces supported by the component.

IDispatch Interface

The IDispatch interface provides a late-bound mechanism that can be used to access information about the methods or properties of an object.

HRESULT Codes

All COM functions and interface methods return a value of the type HRESULT, which stands for 'result handle'. HRESULT returns success, warning, and error values. HRESULTs are 32-bit values with several fields encoded in the value. Common HRESULT values are:

VALUE	ERROR	MEANING
0x8000FFFF	E_UNEXPECTED	Unexpected failure.
0x80004001	E_NOTIMPL	Not implemented.
0x8007000E	E_OUTOFMEMORY	Ran out of memory.

0x80070057	E_INVALIDARG	One or more arguments are invalid.
0x80004002	E_NOINTERFACE	No such interface supported.
0x80004003	E_POINTER	Invalid pointer.
0x80070006	E_HANDLE	Invalid handle.
0x80004004	E_ABORT	Operation aborted.
0x80004005	E_FAIL	Unspecified error.
0x80070005	E_ACCESSDENIED	General access denied error.
0x80000001	E_NOTIMPL	Not implemented.
0x80020001	DISP_E_UNKNOWNINTERFACE	Unknown interface.
0x80020003	DISP_E_MEMBERNOTFOUND	Member not found.
0x80020004	DISP_E_PARAMNOTFOUND	Parameter not found.
0x80020005	DISP_E_TYPEMISMATCH	Type mismatch.
0x80020006	DISP_E_UNKNOWNNAME	Unknown name.
0x80020007	DISP_E_NONAMEDARGS	No named arguments.
0x80020008	DISP_E_BADVARTYPE	Bad variable type.
0x80020009	DISP_E_EXCEPTION	Exception occurred.
0x8002000A	DISP_E_OVERFLOW	Out of present range.
0x8002000B	DISP_E_BADINDEX	Invalid index.
0x8002000C	DISP_E_UNKNOWNLCID	Unknown LCID.
0x8002000D	DISP_E_ARRAYISLOCKED	Memory is locked.
0x8002000E	DISP_E_BADPARAMCOUNT	Invalid number of parameters.
0x8002000F	DISP_E_PARAMNOTOPTIONAL	Parameter not optional.

0x80020010	DISP_E_BADCALLEE	Invalid callee.
0x80020011	DISP_E_NOTACOLLECTION	Does not support a collection.

Copyright and Trademark Information

Interactive Intelligence, Inc.
7601 Interactive Way
Indianapolis, Indiana 46278
Telephone/Fax (317) 872-3000
www.ININ.com