PureConnect®

2019 R4

Generated:

11-November-2019

Content last updated:

28-February-2019

See Change Log for summary of changes.

GENESYS™

SOAP Notifier COM API

Developer's Guide

Abstract

This document contains Application Programming Interface (API) reference informaiton for the Simple Object Access Protocol (SOAP) with PureConnect Notifier Component Object Model (COM).

For the latest version of this document, see the PureConnect Documentation Library at: http://help.genesys.com/cic.

Table of Contents

Interfaces

Objects Exposed by this API

SOAP Notifier COM gives developers programmatic control over SOAP *request* and *response* objects. It also provides objects that manage Notifier connections and which process streams of Base64-encoded data. The Interfaces in the SOAP Notifier COM API are:

Interface	Description
ISOAPRequest	The ISOAPRequest interface issues raw Notifier SOAP requests. This component sends SOAP requests to a handler. It packages a request, sends it, waits for a response, and unpacks the response.
ISOAPResponse	The ISOAPResponseInterface component writes payload or transport data into ISequentialStream streams, and returns payload objects.
ISOAPNotifierTransport	This component provides a COM wrapper on the Notifier (client) library. The Notifier Transport component represents a client connection to a certain Notifier server. It is used by the SOAP Request component as transport for requests.
ISOAPBase64	The ISOAPBase64 interface provides stateless methods that convert binary data to and from Base64 encoded strings. Base64 is an encoding system (defined by RFC 2045) that formats binary information for transmission through network connections.

See Also

SOAP Transport Information and Control

Structure of IP Notification Messages

Using Microsoft SOAP Toolkit with ISoapConnector

ISOAPBase64 Interface

Overview

Derived From:	IDispatch
Interface ID:	{9DEB5514-1FCE-4708-94D9-F3918CB0F868}
Class ID:	{DF98E2DC-B3AD-47E2-AECB-FC52A5EEA826}
Prog ID:	ININ.SOAPBase64

The ISOAPBase64 interface provides stateless methods that convert binary data to and from Base64 encoded strings. Base64 is an encoding system (defined by RFC 2045) that formats binary information for transmission through network connections.

Methods

DecodeToBinary	Decodes the base64 encoded data and returns it as a SAFEARRAY of bytes.
DecodeToFile	Decodes base64 encoded data and writes it to a file. Existing output files can be appended or overwritten.
DecodeToStream	Decodes Base64 encoded data and writes it into the supplied IStream or ISequentialStream.
DecodeToString	Decodes Base64 encoded data into a string.
Encode	Encodes the supplied data into a string that is Base64 encoded.
EncodeFile	Encodes the binary content of a file to create a Base64 encoded string.

Properties

None.

ISOAPBase64::DecodeToBinary Method

Synopsis

Decodes the base64 encoded data and returns it as a SAFEARRAY of bytes.

Function Prototype

HRESULT DecodeToBinary(

[in] BSTR bstrEncodedData,

[out, retval] SAFEARRAY(BYTE)* paResult

C/C++ Syntax

HRESULT DecodeToBinary(BSTR bstrEncodedData, SAFEARRAY(BYTE)* paResult);

Parameters

bstrEncodedData

The input parameter is a string that contains Base64-encoded data.

paResult

The decoded data is returned as a SAFEARRAY of bytes.

ISOAPBase64::DecodeToFile Method

Synopsis

Decodes base64 encoded data and writes it to a file. Existing output files can be appended or overwritten.

Function Prototype

```
HRESULT DecodeToFile(
  [in] BSTR bstrEncodedData,
  [in] BSTR bstrFilename,
  [in,optional] VARIANT vtAppendToFile
);
```

C/C++ Syntax

```
HRESULT DecodeToFile(BSTR bstrEncodedData, BSTR bstrFilename, VARIANT vtAppendToFile);
```

Parameters

bstrEncodedData

A string containing the Base64-encoded data you wish to decode.

bstrFilename

The fully qualified path and filename of the file to be created.

vtAppendToFile

This optional parameter determines whether data will be appended to an existing file, if one exists. The default is False, meaning that existing files are replaced.

ISOAPBase64::DecodeToStream Method

Synopsis

Decodes Base64 encoded data and writes it into the supplied IStream or ISequentialStream.

Function Prototype

HRESULT DecodeToStream
  [in] BSTR bstrEncodedData,
  [in] IUnknown* pObject
);

C/C++ Syntax

HRESULT DecodeToStream(BSTR bstrEncodedData, IUnknown* pObject);

Parameters

bstrEncodedData

String containing the base54 encoded data.

pObject

An IUnknown pointer to an object supporting IStream or ISequentialStream.

ISOAPBase64::DecodeToString Method

Synopsis

Decodes Base64 encoded data into a string.

Function Prototype

```
HRESULT DecodeToString(
   [in] BSTR bstrEncodedData,
   [in,optional] VARIANT vtCharacterSet,
   [out, retval] BSTR* pbstrResult
);
```

C/C++ Syntax

```
HRESULT DecodeToString(BSTR bstrEncodedData, VARIANT vtCharacterSet, BSTR* pbstrResult);
```

Parameters

`bstrEncodedData`

String containing the base54 encoded data.

`vtCharacterSet`

This optional input parameter is a VARIANT that identifies a character set. Unicode (UTF-8) is used by default. After decoding, the data is converted to a UNICODE string using this character set. The character set may be specified as string or as a numeric code page.

`pbstrResult`

The return value is a string that contains the decoded data.

ISOAPBase64::Encode Method

Synopsis

Encodes the supplied data into a string that is Base64 encoded.

Function Prototype

```
HRESULT Encode(
   [in] VARIANT vtData,
   [in,optional] VARIANT vtCharacterSet,
   [in,optional] VARIANT vtMaxLineWidth,
   [in,optional] VARIANT vtLineSeparator,
   [out, retval] BSTR* pbstrResult
);
```

C/C++ Syntax

```
HRESULT Encode(VARIANT vtData, VARIANT vtCharacterSet, VARIANT vtMaxLineWidth, VARIANT
vtLineSeparator, BSTR* pbstrResult);
```

Parameters

`vtData`

The data to be encoded. This may be a string, a SAFEARRAY of bytes, or an object supporting IStream or ISequentialStream.

`vtCharacterSet`

This optional input parameter is a VARIANT that identifies a character set. Unicode (UTF-8) is used by default. If the data to be encoded is a string, the string is converted to the specified character set before it is encoded. The character set may be specified as string or as a numeric code page.

`vtMaxLineWidth`

This optional input parameter is a VARIANT that sets the maximum width of a line of encoded data in characters. The default is 0, which indicates no line breaks.

`vtLineSeparator`

This optional parameter specifies the characters used to separate lines (e.g. the characters that will be written to create a line break). The default is CRLF (\r\n).

`pbstrResult`

The return value is a Base64 encoded string.

ISOAPBase64::EncodeFile Method

Synopsis

Encodes the binary content of a file to create a Base64 encoded string.

Function Prototype

```
HRESULT EncodeFile(
   [in] BSTR bstrFilename,
   [in,optional] VARIANT vtMaxLineWidth,
   [in,optional] VARIANT vtLineSeparator,
   [out, retval] BSTR* pbstrResult
);
```

C/C++ Syntax

```
HRESULT EncodeFile(BSTR bstrFilename, VARIANT vtMaxLineWidth, VARIANT vtLineSeparator, BSTR*
pbstrResult);
```

Parameters

`bstrFilename`

The Name (and Path) of the file to encode.

`vtMaxLineWidth`

This optional input parameter is a VARIANT that sets the maximum width of a line of encoded data in characters. The default is 0, which indicates no line breaks.

`vtLineSeparator`

This optional parameter specifies the characters used to separate lines (e.g. the characters that will be written to create a line break). The default is CRLF (\r\n).

`pbstrResult`

The return value is a Base64 encoded string.

ISOAPNotifierTransport Interface

Overview

Derived From:	IDispatch
Interface ID:	{F37ABE5C-12BF-4757-95F8-EA6267528CA7}
Class ID:	{C26567CE-1B3F-475B-9650-38742A108581}
Prog ID:	ININ.SOAPNotifierTransport

This component provides a COM wrapper on the Notifier (client) library. The Notifier Transport component represents a client connection to a certain Notifier server. It is used by the SOAP Request component as transport for requests.

Frequent creation and destruction of Notifier transport objects is inefficient. We recommend that you reuse transport objects through caching. Since this object is multi-threaded safe, multiple requests may be sent over the same transport at the same time.

Methods

Connect	Connects the transport to the Notifier server.
GetProperty	The GetProperty method retrieves a supplemental transport property.
SetProperty	SetProperty assigns a value to a supplemental transport property.

Properties

Connected	This read-only property returns True if the transport is connected to a Notifier server, meaning that the ISOAPNotifierTransport::Connect method has been called.

ISOAPNotifierTransport::Connect Method

Synopsis

Connects the transport to the Notifier server.

Function Prototype

```
HRESULT Connect(
    [in,optional] VARIANT vtServer,
    [in,optional] VARIANT vtApplicationId,
    [in,optional] VARIANT vtUserId,
    [in,optional] VARIANT vtPassword,
    [in,optional] VARIANT vtClientName,
    [in,optional] VARIANT vtFlags
);
```

C/C++ Syntax

```
HRESULT Connect(VARIANT vtServer, VARIANT vtApplicationId, VARIANT vtUserId, VARIANT vtPassword,
VARIANT vtClientName, VARIANT vtFlags);
```

Parameters

`vtServer`

This optional input parameter is a VARIANT containing the Name of the Notifier server . If you do not specify a server name, the name stored in the registry of the localhost is used by default.

`vtApplicationId`

vtApplicationId is an optional input parameter of type VARIANT. It can contain the Name of the application establishing the connection.

`vtUserId`

vtUserId is an optional input parameter of type VARIANT. It can contain the Username for server login. The default value is an empty string.

`vtPassword`

vtPassword is an optional input parameter of type VARIANT. It can contain the Password for server login. The default value is an empty string.

`vtClientName`

vtClientName is an optional input parameter of type VARIANT. It can contain the Name of the SOAP client that will be sent with the SOAP request for debugging purposes. If you do not specify a value, it uses the value specified for vtApplicationId. If vtApplicationId is empty, the module name is used instead.

`vtFlags`

These optional connect flags are masked using any combination of the following:

Flag	Description
connectFlag_Default (0)	Default setting.
connectFlag_PromptOnFailure (1)	Pop dialog when login fails.
connectFlag_ForcePrompt (2)	Always pop a login dialog.
connectFlag_TryBackupFirst (4)	Attempt to connect to backup Notifier server first.
connectFlag_DontTryBackup (8)	Connect to server only and not to its backup.

ISOAPNotifierTransport::GetProperty Method

Synopsis

The GetProperty method retrieves a supplemental transport property.

Function Prototype

```
HRESULT GetProperty(
   [in] BSTR bstrName,
   [out, retval] VARIANT* pvtResult
);
```

C/C++ Syntax

```
HRESULT GetProperty(BSTR bstrName, VARIANT* pvtResult);
```

Parameters

`bstrName`

The name of a property to retrieve:

Property Name	Description
ClientName	SOAP client name.
ClientId	Read-only. SOAP Client identifier, a dynamically created object identifier that is unique to this instance.
ServerName	Read-only. Name of the server we are connected to.
UserId	Read-only. UserId of the connected user.
BackupConfigured	Read-only. True if the backup server is configured.
OnLocalHost	Read-only. True if this process and Notifier are on the same machine.

`pvtResult`

The return value is the property identified by bstrName.

ISOAPNotifierTransport::SetProperty Method

Synopsis

SetProperty assigns a value to a supplemental transport property.

Function Prototype

```
HRESULT SetProperty(
   [in] BSTR bstrName,
   [in] VARIANT vtValue
);
```

C/C++ Syntax

```
HRESULT SetProperty(BSTR bstrName, VARIANT vtValue);
```

Parameters

`bstrName`

The name of the property you wish to set:

Property Name	Description
ClientName	SOAP client name.
ClientId	Read-only. SOAP Client identifier, a dynamically created object identifier that is unique to this instance.
ServerName	Read-only. Name of the server we are connected to.
UserId	Read-only. UserId of the connected user.
BackupConfigured	Read-only. True if the backup server is configured.
OnLocalHost	Read-only. True if this process and Notifier are on the same machine.

`vtValue`

The value to assign to the property identified by bstrName.

ISOAPNotifierTransport::Connected Property

get_Connected

This read-only property returns True if the transport is connected to a Notifier server, meaning that the ISOAPNotifierTransport::Connect method has been called.

Function Prototype

```
HRESULT Connected(
   [out, retval] VARIANT_BOOL* pbResult
);
```

C/C++ Syntax

```
HRESULT get_Connected(VARIANT_BOOL* pbResult);
```

Parameters

`pbResult`

True if the ISOAPNotifierTransport::Connect method has been called; otherwise False.

ISOAPRequest Interface

Overview

Derived From:	IDispatch
Interface ID:	{5B18473E-8D58-4632-93CC-97ABE63E1F1A}
Class ID:	{4489E7D6-F97A-41D1-B64B-ACC63544441C}
Prog ID:	ININ.SOAPRequest

The ISOAPRequest interface issues raw Notifier SOAP requests. This component sends SOAP requests to a handler. It packages a request, sends it, waits for a response, and unpacks the response.

Methods

GetProperty	Retrieves a supplemental SOAP request property.
Initialize	Initializes the request object and sets the transport to be used. This method does not return a value.
Reset	Resets the request object by clearing SOAPAction, TransportInfo, and Payload data. This prepares the object for the next request.
Send	Sends the request and waits synchronously for the response object to be returned.
SetProperty	Sets a supplemental SOAP request property.
SetSOAPPayload	Sets the SOAP payload data to be sent with the request. Currently, this must pass an object that supports IStream. However, strings may be supported in a future release. This method does not return a value.
SetTransportInfo	Sets the transport information data to be sent with the request. Currently, this must pass an object that supports IStream. However, strings may be supported in a future release. This method does not return a value.

Properties

ExpectResponse	This property returns True if a response to this SOAP request is expected. False is returned if the request is a one-way request that does not generate a response.
InitiatorEvent	Returns the Initiator Event of this request. Uses SOAPAction if the event is not specified or empty. Changing the SOAPAction also resets this property.
SOAPAction	Gets the SOAPAction code of the request.
Transport	The Transport property is read-only. It returns an IUnknown pointer to the transport object used for requests.

ISOAPRequest::GetProperty Method

Synopsis

Retrieves a supplemental SOAP request property.

Each SOAP request is packaged into a eSOAP_REQUEST_OBJECT object. The component sends the notification and unpacks the Response message (eSOAP_RESPONSE_OBJECT).

Function Prototype

```
HRESULT GetProperty(
   [in] BSTR bstrName,
   [out, retval] VARIANT* pvtResult
);
```

C/C++ Syntax

```
HRESULT GetProperty(BSTR bstrName, VARIANT* pvtResult);
```

Parameters

`bstrName`

The input parameter is a string containing the property name to retrieve.

`pvtResult`

The return value is a VARIANT containing the value of the named property.

ISOAPRequest::Initialize Method

Synopsis

Initializes the request object and sets the transport to be used. This method does not return a value.

Function Prototype

```
HRESULT Initialize(
   [in] IUnknown* pTransport
);
```

C/C++ Syntax

```
HRESULT Initialize(IUnknown* pTransport);
```

Parameters

`pTransport`

The input parameter is an IUnknown pointer to an ISOAPNotifierTransport object. Currently this is the only transport object supported.

ISOAPRequest::Reset Method

Synopsis

Resets the request object by clearing SOAPAction, TransportInfo, and Payload data. This prepares the object for the next request.

Function Prototype

```
HRESULT Reset(void);
```

C/C++ Syntax

```
HRESULT Reset(void);
```

Parameters

None.

ISOAPRequest::Send Method

Synopsis

Sends the request and waits synchronously for the response object to be returned.

Function Prototype

```
HRESULT Send(
   [in] VARIANT vtTimeout,
   [out, retval] ISOAPResponse** ppResult
);
```

C/C++ Syntax

```
HRESULT Send(VARIANT vtTimeout, ISOAPResponse** ppResult);
```

Parameters

`vtTimeout`

This input parameter is an optional VARIANT that is the maximum time to wait for a response in milliseconds. The default is 60000 milliseconds, or one minute.

`ppResult`

The return value is an object that implements the ISOAPResponse interface, or NULL if no response was expected (ISOAPRequest::ExpectResponse = False). A COM exception is issued if the request is un-handled. (e.g. the handler goes out of scope without sending a response) or the request times out.

ISOAPRequest::SetProperty Method

Synopsis

Sets a supplemental SOAP request property.

Function Prototype

```
HRESULT SetProperty(
   [in] BSTR bstrName,
   [in] VARIANT vtValue
);
```

C/C++ Syntax

```
HRESULT SetProperty(BSTR bstrName, VARIANT vtValue);
```

Parameters

`bstrName`

The input value a string containing the name of the property you wish to set.

`vtValue`

The new value of the property identified by bstrName.

ISOAPRequest::SetSOAPPayload Method

Synopsis

Sets the SOAP payload data to be sent with the request. Currently, this must pass an object that supports IStream. However, strings may be supported in a future release. This method does not return a value.

The payload data must be well-formed, meaning that it conforms to the XML specification. This function does not check data to see if it is valid or well-formed.

Function Prototype

```
HRESULT SetSOAPPayload(
   [in] VARIANT vtSOAPPayload
);
```

C/C++ Syntax

```
HRESULT SetSOAPPayload(VARIANT vtSOAPPayload);
```

Parameters

`vtSOAPPayload`

The input parameter is a VARIANT that contains well-formed payload data.

ISOAPRequest::SetTransportInfo Method

Synopsis

Sets the transport information data to be sent with the request. Currently, this must pass an object that supports IStream. However, strings may be supported in a future release. This method does not return a value.

The transport information must be well-formed, meaning that it conforms to the XML specification. This function does not check data to see if it is valid or well-formed. For more information about TransportInfo, refer to *Appendix* A: SOAP Transport Information and Control in the *Installi*ng and Using IC's SOAP Functionality Technical Reference and Installation Guide.

Function Prototype

```
HRESULT SetTransportInfo(
   [in] VARIANT vtTransportInfo
);
```

C/C++ Syntax

```
HRESULT SetTransportInfo(VARIANT vtTransportInfo);
```

Parameters

`vtTransportInfo`

Well-formed XML data describing transport information that will be sent with the request.

ISOAPRequest::ExpectResponse Property

get_ExpectResponse

This property returns True if a response to this SOAP request is expected. False is returned if the request is a one-way request that does not generate a response.

Function Prototype

```
HRESULT ExpectResponse(
   [out, retval] VARIANT_BOOL* pbResult
);
```

C/C++ Syntax

```
HRESULT get_ExpectResponse(VARIANT_BOOL* pbResult);
```

Parameters

`pbResult`

True if a response is expected; otherwise False for one-way requests.

`put_ExpectResponse`

Set this property True if the request should generate a response. If the request is a one-way request that does not generate a response, set this property to False.

Function Prototype

```
HRESULT ExpectResponse(
   [in] VARIANT_BOOL bExpectResponse
);
```

C/C++ Syntax

```
HRESULT put_ExpectResponse(VARIANT_BOOL bExpectResponse);
```

Parameters

`bExpectResponse`

True if the request should generate a response; otherwise False.

ISOAPRequest::InitiatorEvent Property

get_InitiatorEvent

Returns the Initiator Event of this request. Uses SOAPAction if the event is not specified or empty. Changing the SOAPAction also resets this property.

Function Prototype

```
HRESULT InitiatorEvent(
   [out, retval] BSTR* pbstrResult
);
```

C/C++ Syntax

```
HRESULT get_InitiatorEvent(BSTR* pbstrResult);
```

Parameters

`pbstrResult`

The notification event of this request.

put_InitiatorEvent

Sets the Initiator Event of this request. Uses SOAPAction if the event is not specified or empty. Changing the SOAPAction also resets this property.

Function Prototype

```
HRESULT InitiatorEvent(
   [in] BSTR bstrInitiatorEvent
);
```

C/C++ Syntax

```
HRESULT put_InitiatorEvent(BSTR bstrInitiatorEvent);
```

Parameters

`bstrInitiatorEvent`

The notification event of this request.

ISOAPRequest::SOAPAction Property

get_SOAPAction

Gets the SOAPAction code of the request.

Function Prototype

```
HRESULT SOAPAction(
   [out, retval] BSTR* pbstrResult
);
```

C/C++ Syntax

```
HRESULT get_SOAPAction(BSTR* pbstrResult);
```

Parameters

`pbstrResult`

The return value is a string containing the SOAPAction for this request.

put_SOAPAction

Sets the SOAPAction code of this request.

Function Prototype

```
HRESULT SOAPAction(
   [in] BSTR bstrSOAPAction
);
```

C/C++ Syntax

```
HRESULT put_SOAPAction(BSTR bstrSOAPAction);
```

Parameters

`bstrSOAPAction`

The input parameter is a string containing the SOAPAction to assign to this request.

ISOAPRequest::Transport Property

get_Transport

The Transport property is read-only. It returns an IUnknown pointer to the transport object used for requests.

Function Prototype

```
HRESULT Transport(
   [out, retval] IUnknown** ppResult
);
```

C/C++ Syntax

```
HRESULT get_Transport(IUnknown** ppResult);
```

Parameters

ppResult

The return value is an IUnknown pointer to the transport object.

ISOAPResponse Interface

Overview

Derived From:	IDispatch
Interface ID:	{D3CD2DA4-E6E9-4845-B8A1-EF1098286ECC}

The ISOAPResponseInterface component writes payload or transport data into ISequentialStream streams, and returns payload objects.

Methods

WritePayload	This method writes payload data to an ISequentialStream passed as an argument.
WriteTransportCtrlData	This method writes transport control data to an ISequentialStream passed as an argument. If there is no transport control data, nothing is written, but the method succeeds.

Properties

Fault	Returns True if the server returned a SOAP Fault message; otherwise False.
Payload	Returns an IUnknown pointer to an object that implements IStream, so that you can access the object's SOAP response payload data.
RequestId	Returns the Identifier of this request.
Success	Returns True if the request was successfully executed on the server; otherwise False.
TransportCtrlData	This read-only property returns an object that implements IStream, so that you can access the object's transport control data.

ISOAPResponse::WritePayload Method

Synopsis

This method writes payload data to an ISequentialStream passed as an argument.

Function Prototype

```
HRESULT WritePayload(
   [in] IUnknown* pObject
);
```

C/C++ Syntax

```
HRESULT WritePayload(IUnknown* pObject);
```

Parameters

pObject

IUnknown pointer to an ISequentialStream object.

ISOAPResponse::WriteTransportCtrlData Method

Synopsis

This method writes transport control data to an ISequentialStream passed as an argument. If there is no transport control data, nothing is written, but the method succeeds.

Function Prototype

```
HRESULT WriteTransportCtrlData(
   [in] IUnknown* pObject
);
```

C/C++ Syntax

```
HRESULT WriteTransportCtrlData(IUnknown* pObject);
```

Parameters

pObject

IUnknown pointer to an ISequentialStream object.

ISOAPResponse::Fault Property

get_Fault

Returns True if the server returned a SOAP Fault message; otherwise False.

Function Prototype

```
HRESULT Fault(
   [out, retval] VARIANT_BOOL* pbResult
);
```

C/C++ Syntax

```
HRESULT get_Fault(VARIANT_BOOL* pbResult);
```

Parameters

pbResult

True if the server returned a SOAP Fault message; otherwise False.

ISOAPResponse::Payload Property

get_Payload

Returns an IUnknown pointer to an object that implements IStream, so that you can access the object's SOAP response payload data.

Function Prototype

```
HRESULT Payload(
   [out, retval] IUnknown** ppResult
);
```

C/C++ Syntax

```
HRESULT get_Payload(IUnknown** ppResult);
```

Parameters

ppResult

IUnknown pointer to an IStream object.

ISOAPResponse::RequestId Property

get_RequestId

Returns the Identifier of this request.

Function Prototype

```
HRESULT RequestId(
   [out, retval] long* plResult
);
```

C/C++ Syntax

```
HRESULT get_RequestId(long* plResult);
```

Parameters

plResult

The return value is a LONG that contains the request identifier number.

ISOAPResponse::Success Property

get_Success

Returns True if the request was successfully executed on the server; otherwise False.

Function Prototype

```
HRESULT Success(
   [out, retval] VARIANT_BOOL* pbResult
);
```

C/C++ Syntax

```
HRESULT get_Success(VARIANT_BOOL* pbResult);
```

Parameters

pbResult

True if the request was successfully executed on the server; otherwise False.

ISOAPResponse::TransportCtrlData Property

get_TransportCtrlData

This read-only property returns an object that implements IStream, so that you can access the object's transport control data.

Function Prototype

```
HRESULT TransportCtrlData(
   [out, retval] IUnknown** ppResult
);
```

C/C++ Syntax

```
HRESULT get_TransportCtrlData(IUnknown** ppResult);
```

Parameters

ppResult

An IUnknown pointer to the object.

SOAP Transport Information and Control

The transport info structure must have a **TransportInfo** root element that is in no namespace. It must have a **name** attribute that contains the name of the transport. The transport name is useful for debugging, tracing, or to perform transport specific operations. However, this Transport Information is not defined by the SOAP specification. The TransportInfo element may have any number of child elements. The following is the schema for the Transport Info structure. For efficiency, a client may chose not to include transport information, but still send the transport name. In this case, the **TransportInfo** element will be empty.

```
<?xml version="1.0"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
<xsd:element name="TransportInfo" type="TransportInfoType"/>
<xsd:complexType name="TransportInfoType">
<xsd:attribute name="name" type="xsd:string" use="required"/>
<any minOccurs="0" maxOccurs="unbounded" />
<anyAttribute />
</xsd:complexType>
</xsd:schema>
```

The Transport Control structure must have a TransportCtrl root element that is in no namespace. It may contain any number of attributes or child elements:

```
<?xml version="1.0"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
<xsd:element name="TransportCtrl" type="TransportCtrl"/>
<xsd:complexType name="TransportCtrl">
<any minOccurs="0" maxOccurs="unbounded" />
<anyAttribute />
</xsd:complexType>
</xsd:schema>
```

HTTP Transport

Request (Transport Info)

The following schema describes the transport information for the HTTP transport. The **HTTP** element is the child element of the **TransportInfo** element generated by the ISAPI Listener.

```
<?xml version="1.0"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
<xsd:element name="HTTP" type="HTTP"/>
<xsd:complexType name="HTTP">
<xsd:sequence>
<xsd:element name="Headers" type="Headers" minOccurs="0"/>
</xsd:sequence>
<xsd:attribute name="method" type="xsd:string" use="required"/>
<xsd:attribute name="url" type="xsd:string" use="required"/>
<xsd:attribute name="pathInfo" type="xsd:string" use="required"/>
<xsd:attribute name="queryString" type="xsd:string" use="required"/>
<xsd:attribute name="remoteAddr" type="xsd:string" use="required"/>
</xsd:complexType>
```

```
<xsd:complexType name="Headers">
<xsd:element name="Header" type="Header" minOccurs="0" maxOccurs="unbounded"/>
</xsd:complexType>
<xsd:complexType name="Header">
<xsd:simpleContent>
<xsd:extension base="xsd:string">
<xsd:attribute name="name" type="xsd:string" use="required"/>
</xsd:extension>
</xsd:simpleContent>
</xsd:complexType>
</xsd:schema>
```

HTTP Element Attributes

The attributes of the HTTP entry have the following meaning:

method

The HTTP method with which the request was made. In our case usually POST. This is equivalent to the value of the CGI variable REQUEST_METHOD.

url

Designates the base portion of the URL. Parameter values are not included (see pathInfo and queryString).

pathInfo

Contains the additional path information given by the client. This consists of the trailing part of the URL after the ISAPI DLL name, but before the query string, if any. Corresponds to the CGI variable PATH_INFO.

queryString

Contains the information that follows the first question mark in the URL Corresponds to the CGI variable QUERY_STRING.

remoteAddr

Contains the IP address of the client or agent of the client (for example gateway, proxy, or firewall) that sent the request. Corresponds to the CGI variable REMOTE_ADDR.

Request Transport Example

The following is a sample Transport Info structure adhering to the above schemas:

```
<TransportInfo name="HTTP">
<HTTP method="POST" url="/soapendpoint/I3SOAPISAPIAD.DLL" pathInfo=""
  queryString="" remoteAddr="127.0.0.1">
<Headers>
<Header name="Host">localhost</Header>
<Header name="Content-Type">text/xml</Header>
<Header name="Content-Length">1234</Header>
<Header name="SOAPAction">"uri:my-soap-request#MyMethod"</Header>
```

```
</Headers>
</HTTP>
</TransportInfo>
```

Response (Transport Control)

The following schema describes the transport control data for the HTTP transport. The **HTTP** element is the child element of the **TransportCtrl** element.

```
<?xml version="1.0"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
<xsd:element name="HTTP" type="HTTP"/>
<xsd:complexType name="HTTP">
<xsd:sequence>
<xsd:element name="Headers" type="Headers" minOccurs="0"/>
</xsd:sequence>
<xsd:attribute name="statusCode" type="xsd:positiveInteger" use="optional"/>
<xsd:attribute name="statusText" type="xsd:string" use="optional"/>
</xsd:complexType>
<xsd:complexType name="Headers">
<xsd:element name="Header" type="Header" minOccurs="0" maxOccurs="unbounded"/>
</xsd:complexType>
<xsd:complexType name="Header">
<xsd:simpleContent>
<xsd:extension base="xsd:string">
<xsd:attribute name="name" type="xsd:string" use="required"/>
</xsd:extension>
</xsd:simpleContent>
</xsd:complexType>
</xsd:schema>
```

Response Transport Example

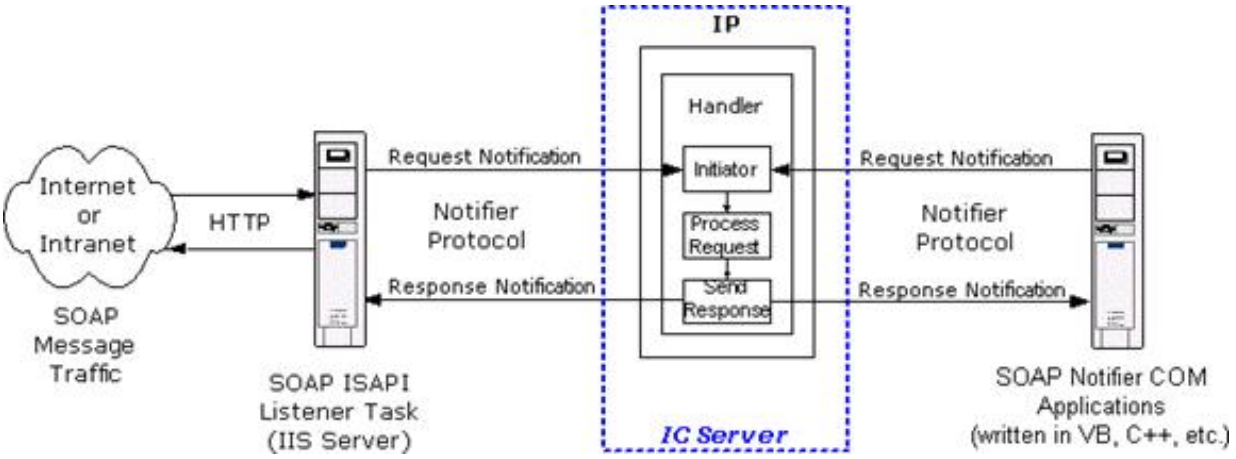
The following is an example of a transport control response structure that "asks" the ISAPI listener to send a 501 error (Not Implemented) back to the client. The default status codes are 200 (OK) for successfully processed requests, and 500 (Internal Server Error) for failed requests (body contains a <Fault> element).

```
<TransportCtrl>
   <HTTP statusCode="501" statusText="Not Implemented"/>
</TransportCtrl>
```

Header fields specified in the TransportControl structure will have *precedence* over the default headers generated by the ISAPI listener (such as "Content-Type:text/xml").

Structure of IP Notification Messages

For the purpose of the IC SOAP implementation, message transport is not limited to any kind of protocol. SOAP requests are sent as notifications containing payload data as well as transport-specific out-of-band information. As HTTP is most frequently used as transport for SOAP requests through the internet, an ISAPI listener is provided. However, any kind of client who "talks" Notifier could issue SOAP requests. For example, a SOAP Notifier COM object can directly send SOAP packets to IC.



HTTP and Notifier protocols transport SOAP messages between components in the IC environment.

Request Message Structure

Since Interaction Processor does not directly support Notifier Requests, notifications are used to emulate the request/response mechanism. The SOAP request notifications use the IC's eSOAP_REQUEST_OBJECT object type and an object ID that identifies the client.

Field Name	Type	Description
Version	int	2 (Version number of the message structure).
RequestId	DWORD	Request identifier specified by the client to identify the response. The server must send it back in the response.
ClientName	string	Name of the client
Respond	bool	*False* Server must not send a response back to the client. *True* Server must send a response to the client.
InitiatorEvent	string	String of the notification Event-ID. Often same as SOAPAction. The notification event ("Initiator Event") can either be explicitly specified or the SOAPAction is will be used as default. In early editions of this API, the Notification Event was always the SOAPAction, but since a handler may only trigger on one specific event (or a wildcard event), the framework was expanded so that the notification event ("Initiator Event") can now be specified explicitly.
SOAPAction	string	SOAP Action name
TransportInfoSize	DWORD	Size in bytes of the transport information data block
TransportInfoData	BYTE[]	Transport information data. This is an XML document that encodes transport specific information. For example, for HTTP it contains the verb as well as the HTTP header fields. The default character set is UTF-8, but the data block may contain an XML declaration with the appropriate encoding attribute. This field may be omitted (Size = 0).
PayloadSize	DWORD	Size in bytes of the SOAP payload data block
PayloadData	BYTE[]	This is the data of the SOAP envelope. The default character set is UTF-8, but the data block may contain XML declaration with the appropriate encoding attribute.

Response Message Structure

The response is sent back to the client with the object type eSOAP_RESPONSE_OBJECT. The object ID uniquely identifies the client and is used to send the response back to the right client. The clients use GetNotifierSequenceNumber to obtain a unique identifier to identify themselves. Clients that do not expect a response must set the 'Respond' flag in the request data block to 'false'. The Message data of the request and response have the following structure.

Field Name	Type	Description
Version	int	2 (Version number of the message structure).
RequestId	DWORD	Request identifier specified by the client to identify the response. The server fills this slot with the value in the request data.
ResultCode	enum	Enumeration indicating how the request was processed. Succeeded (0) The SOAP request was processed successfully and without fault. Failed (1) The SOAP request failed. This flag is set by the 'SOAP Send Response' tool when the body contains a <Fault> element. A client can thus check for a failed request without having to unpack the payload. Unhandled (2) The Initiator fired, but the handler did not invoke 'SOAP Send Response' to return a response (the 'SOAP Request' handle went out of scope). The payload and transport control data are empty.
TransportControlSize	DWORD	Size in bytes of the transport control data block
TransportControlData	BYTE[]	Transport control data. This is an XML document that contains transport specific out-of-band control data. For example, for HTTP it contains additional HTTP header fields or status codes to convey special failures. The default character set is UTF-8, but the data may contain an XML declaration with the appropriate encoding attribute. Data block may be empty.
PayloadSize	DWORD	Size in bytes of the SOAP response payload data block. The default character set is UTF-8, but the data may contain an XML declaration with the encoding attribute.
PayloadData	BYTE[]	This is the data of the SOAP response envelope. The data block is empty if the 'Unhandled' flag is set.

Using Microsoft SOAP Toolkit with ISoapConnector

Microsoft's SOAP Toolkit makes it possible for programmers to invoke a web service as easily as invoking a method on an object. The Microsoft SOAP Toolkit reads in a WSDL file, and dynamically generates COM interfaces for operations described in the file.

The SOAP Notifier COM API works cooperatively with Microsoft's SOAP Toolkit. The SOAP Notifier COM API provides a component named **ISoapConnector** that is used to initiate SOAP handlers using Microsoft SOAP Toolkit 2.0.

Using ISoapConnector (MSSOAP Notifier Connector)

```
ProgId:  ININ.MSSOAPNotifierConnector
```

The VB example below shows how to use the transport. It is assumed that a WSDL file with the service description exists, since this is required for MSSOAPLib.SoapClient.

Using the Transport with Visual Basic 6

```
Dim objTransport As New SOAPNotifierCOMLib.SOAPNotifierTransport
objTransport.Connect "<Notifier>", "<AppId>", "<user>", "<password>", "<ClientName>"
Dim objClient As New MSSOAPLib.SoapClient
objClient.ClientProperty("ConnectorProgID") = "ININ.MSSOAPNotifierConnector"
objClient.mssoapinit "<WSDL filename or URL>"
objClient.ConnectorProperty("Transport") = objTransport
Result = objClient.<method>(<arguments>...)
```

Instead of the SoapClient, you may use the MSSOAPLib.SoapSerializer and MSSOAPLib.SoapReader objects with any object that uses a ISoapConnector.

SOAP Notifier Connector Properties

Transport

Transport object to be used for server communication. Must be set before the first invocation.

SOAPAction

SOAP Action used in the request. If not defined (empty string), uses value from the WSDL file.

InitiatorEvent

Initator Event (notification event) of the request notification. If not specified or as default, the SOAPAction is used. Changing the SOAPAction also resets this property, unless the PreserveInitiatorEvent property is set.

If the SOAPAction has never been set or is an empty string and the value from the WSDL file is used, the InitiatorEvent is reset after each request (again, unless PreserveInitiatorEvent is True).

PreserveInitiatorEvent

If True, changing the SOAPAction does not change the InitiatorEvent property.

RequestTimeout

Maximum amount of time to wait for response in milliseconds. Value < 0 → infinite. Default = 60000 (1 minute).

TransportInfo

Write only. Transport info data. Must be object implementing IStream.

TransportCtrl

Read only. Transport control data, returns IUnknown of an object implementing IStream. Can only be retrieved after invocation until the object using the connector calls the 'BeginMessage' method of the connector (usually, as part of the next invocation).

ResponseObject

Read Only. Returns the ISOAPResponse object resulting from the request. Can only be retrieved after invocation until the object using the connector calls the 'BeginMessage' method of the connector (usually, as part of the next invocation).

Recommended Web Links

Information about XML and SOAP is available on the Internet, of course. The following sites are recommended:

XML

Extensible Markup Language (XML)

The Extensible Markup Language (XML) Home Page at the World Wide Web Consortium (W3C) offers numerous XML links:
http://www.w3.org/XML/

XML Tutorial

XML School is a free online tutorial offered by W3Schools:
http://www.w3schools.com/xml/default.asp

O'Reilly XML.COM

XML.com says that its mission is to help you discover XML and learn how this Internet technology can solve real-world problems in information management and electronic commerce:
http://www.xml.com/

DTD

DTD Table of Contents at XML101.com

Links on this page provide an introduction to DTD, building blocks, elements, attributes, entities, validation, examples, and more:
http://xml101.com/dtd/

Introduction to DTD

An introduction to DTD with sample code:
http://xml101.com/dtd/dtd_intro.asp

SOAP

Simple Object Access Protocol (SOAP) 1.1

W3C SOAP specification document:
http://www.w3.org/TR/SOAP/

SOAP Tutorial

SOAP School is a free online tutorial offered at W3Schools:
http://www.w3schools.com/soap/default.asp

WSDL

Web Services Description Language (WSDL) 1.1

http://www.w3.org/TR/wsdl

XML Namespaces

Namespaces in XML

http://www.w3.org/TR/REC-xml-names/

Glossary

COM

Microsoft's Component Object Model. The COM specification helps developers create component software that is compatible with a variety of languages, including C, ADA, Delphi, Java, and Visual Basic.

Denial of Service Attack

Denial of Service (DoS) attacks are attempts to overload a networked computer system so that it crashes, disconnects from the network, or becomes so overloaded that it cannot respond to legitimate requests.

DTD

Document Type Definition. A DTD defines the XML tags that can be used in an XML document, the order in which tags may appear, and limited information about data types. A DTD can be part of an XML document or can be referenced as an external file. The validating XML parser compares the DTD to the XML document and flags any errors. DTDs have been deprecated in favor of XML Schemas.

Handler

A program built in Interaction Designer that performs some action or actions in response to the occurrence of some event. A handler is a collection of steps organized and linked to form a logical flow of actions and decisions. Handlers are similar in structure to a detailed flowchart. Handlers can start other handlers called subroutines. A handler contains only one initiator step which identifies the type of event that will start the handler.

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. In Visual Basic, a zero result indicates success and a non-zero result indicates failure. Common HRESULT values are:

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.

HTML

Hypertext Markup Language (HTML) is the markup language used to create World Wide Web pages.

CIC Module

One of the many applications that make up the CIC server. These applications have names like manager, server, and services. For example, Queue Manager, Fax Server, and Directory Services are all IC modules.

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.

Initiator

The first step in a handler that waits for a specific type of event to occur. When that event occurs, the Interaction Processor starts an instance of any handler whose initiator is configured for that event. An initiator is a required step that starts a handler. There can be only one Initiator in a handler. Initiator names describe the kind of event used to start a handler. Initiators can pass information from the event into variables that can be used within a handler. Subroutine initiators are not configured to watch for an event. Rather, they start when called from another handler.

Customer Interaction Center (CIC)

The Customer Interaction Center Platform™ is a powerful platform for implementing comprehensive interaction management covering not only calls and faxes but also e-mail messages, Internet text chats, Web callback requests, and voice over Net calls. Using the Customer Interaction Center Platform, enterprises, contact centers, and service providers can centralize the processing of all customer interactions and provide a new level of service and consistency.

Interaction Designer

The CIC graphical application development tool for creating, debugging, editing, and managing handlers and subroutines.

Interaction Processor (IP)

Interaction Processor is the event processing subsystem of Customer Interaction Center that starts instances of handlers when an event occurs.

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.

Method

A method is a software subroutine that performs some type of data processing on an object in a computer system. Methods are sometimes called functions. Data can be passed when methods are called to perform some kind of work. For example, you might call a method named GetStockPrice and pass it a stock symbol to receive the current stock price as the return value.

Microsoft SOAP Toolkit

Microsoft's SOAP Toolkit makes it possible for programmers to invoke a web service as easily as invoking a method on an object. The Microsoft SOAP Toolkit reads in an WSDL file, and dynamically generates COM interfaces for operations described in the file. It packages method parameters in accordance with WSDL service descriptions.

Namespace

Since XML allows tags and attributes to be defined as needed, name collisions occur when the same name is assigned to a tag or an attribute, in different databases. For example, a teacher might define an element named "Grade" to represent a student's score. In the context of an agricultural operation, "Grade" could have a different meaning, as in "Grade A" eggs. *Namespaces* resolve collision issues by associating XML attribute and element names with a specific context, or "namespace". A namespace is an identifier that helps computer programs determine whether identically named elements refer to the same type of data. Using namespaces, a program can determine that a data element named "Grade" in the "Schoolwork" namespace is different from an element called "Grade" in the "EggQuality" namespace.

Notifier

The CIC module that acts as a communication center for all other modules. Notifier listens for events generated by other modules and notifies other interested modules that the event has occurred. Notifier uses a publish-and-subscribe paradigm.

Package

A SOAP package contains information needed to invoke a web service.

Payload

A payload contains data in XML format that is passed to or from a function. Request payloads contain everything needed to execute a function, including data and arguments passed as parameters. Response payloads contain the values that are returned from a function.

Processing Instruction

Processing instructions are read by application-level code (such as parsers) and are used to communicate information without changing the content of an XML document. For example, <?xml version="1.0"?> is a processing instruction that indicates that a document conforms to XML 1.0 specifications. Processing instructions use <?target declaration ?> notation; where target is the name of the application that should process the instruction, and declaration is an instruction or identifier that is meaningful to the application. In the above example, XML is a reserved target that identifies XML parsers.

Protocol

A protocol is a set of rules that one computer uses to communicate with another.

Schema

XML Schema are the successor to DTDs for XML. XML schemas describe method calls, and can recognize and enforce data-types, inheritance, and presentation rules. A schema can be part of an XML document or can be referenced as an external file.

SOAP

Simple Object Access Protocol. SOAP is an XML-based protocol that requests or receives information from peer computers in a decentralized, distributed network. SOAP defines the minimal set of conventions that are needed to invoke code using XML and HTTP. SOAP is used to invoke methods on servers, services, components and objects in another computer. SOAP specifies the XML vocabulary needed to specify method parameters, return values, and exceptions.

TCP/IP

Transmission Control Protocol/Internet Protocol.

Tool

The definition of a single action that can be performed within a handler. This definition includes name, label, runtime information (DLL and function), possible return codes, and parameters. Tools dragged into a handler become steps in that handler.

Valid

A valid XML document conforms to a document structure defined by a schema or DTD (Document Type Definition). Valid documents are well-formed documents that have a DTD or schema applied to them.

Vocabulary

A vocabulary is the set of tags and attributes that are used in an XML document.

Web Service

A web service is a method that can be invoked across the Internet. A web service can perform virtually any data processing activity, ranging from simple information lookups to complicated business transactions. SOAP is frequently employed to invoke web services.

Well-Formed

Well-formed documents follow the rules of XML.

WSDL

Web Services Description Language—an XML-based language that defines the functionality offered by a web service and how to access it. WSDL makes it possible to describe services on CIC so that a worldwide audience can find and use them. WSDL describes a service, the parameters required to invoke it, and the location of the endpoint where the service can be accessed.

XML

Extensible Markup Language. XML provides a structured way to define data in plain text format, so that data can be exchanged between computers.

XSL/XSLT

Extensible Style Language (XSL) is a specification used to transform XML documents into HTML. XSL Transformation (XSLT) provides similar functionality that transforms XML data into a different XML structure.

Change log

transfer content from revisions topic to this topic

Date	Changes
28-February-2019	Date should be in the dd-Month-yyy.
28-February-2019	Created this change log.