SOAP API

Printed help

2018 R2

Last updated 2018-02-26

Table of Contents

SOAP API Developer's Guide

SOAP API Developer's Guide

CIC and SOAP API Developer's Guide

Audience

SOAP stands for *Simple Object Access Protocol*. SOAP is an XML-based protocol specification that defines how information can be exchanged between computers. SOAP supplies the conventions used to invoke methods on servers, services, components and objects. This document introduces XML/SOAP concepts and explains how SOAP facilitates robust data interactions between CIC and remote web services. SOAP supplies the conventions used to invoke methods on remote servers, services, components and objects.

This publication is for managers, technical implementers, and other decision-makers who need to understand the practical implications of SOAP technology in the CIC environment. The introduction is written for a general audience who may not be familiar with XML or SOAP technology. Subsequent sections of this document guide technical implementers through the process of preplanning, installing and configuring the SOAP ISAPI Listener Task and SOAP Notifier COM Components. Instructions for using the SOAP Tracer utility are also provided.

Organization of Material

This documentation is divided into logical, easy-to-digest sections that gradually introduce concepts and specific product features. To fully understand the material, we recommend that you read topics in order. However, most topics are hyperlinked for those who prefer to read in non-linear fashion.

- Introduction to SOAP in the CIC Environment provides short primers on XML and SOAP, and explains the relationship between XML, SOAP and the Interaction Center platform. It introduces CIC's SOAP Components.
- Install and Configure SOAP ISAPI Listener explains how to select a host server, apply prerequisite service packs and hotfixes, and then install SOAP Listener components. This section also explains how to configure the server to prevent denial of service attacks, and how to modify the configuration so that only supported SOAPActions are forwarded to CIC for processing.
- Install SOAP Notifier COM explains how to install and register components needed to run or develop third-party SOAPNotifierCOM applications on a desktop PC.
- Appendix A (SOAP Transport Information) describes HTTP schema used to transport SOAP packets in the CIC environment. This appendix is for advanced readers who are curious about SOAP transport mechanisms used in CIC.
- Appendix B (SOAP Tools) describes tools in Interaction Designer that process SOAP requests and responses.
- Appendix C (Structure of IP Notification Messages) explains the notification message format and protocols used to send requests to and from CIC's Notifier subsystem.
- Appendix D (SOAP ISAPI Listener Fault Messages) is a reference about fault messages returned by the SOAP ISAPI Listener.
- Special terms used with SOAP technology are defined in a Glossary.
- Revisions describes what's new by release.

Related Documentation

1. *CIC and SOAP API Developer's Guide* (this document). This paper provides primers on SOAP and XML, and discusses the components that must be installed to implement SOAP functionality in CIC.
2. *Interaction Center SOAP Listener* Setup installs SOAP ISAPI components on an IIS server. We highly recommend that you read Install and Configure SOAP ISAPI Listener before running the install.
3. The *SOAP Notifier COM Components Install* installs and registers component software used by developers to create high-performance SOAP applications.
4. *SOAP Notifier COM* setup optionally installs the *SOAP Notifier COM API Developer's Guide* (Soap_Notifier_COM_API_DG.chm). This windows help file cross-references the interfaces, methods, and properties exposed by SOAP Notifier COM objects.
5. *SOAP Tools* are documented in Interaction Designer help. These help topics appear when a SOAP tool or toolstep has focus and the F1 key is pressed in Interaction Designer.

Recommended Web Links

XML Home Page at the World Wide Web Consortium (W3C)

http://www.w3.org/XML/

XML Tutorial by W3Schools

http://www.w3schools.com/xml/default.asp

O'Reilly XML.COM

http://www.xml.com/

W3C SOAP specification document:

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

SOAP Tutorial by W3Schools

https://www.w3schools.com/xml/xml_soap.asp

Web Services Description Language (WSDL) 1.1

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

Namespaces in XML

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

Introduction to SOAP in the CIC Environment

Introduction to SOAP in the CIC Environment

This section is for managers and other decision makers who need to understand the practical implications of SOAP technology in a CIC environment. No prior knowledge of XML or SOAP is required to understand the concepts presented here. XML and SOAP are standards for information exchange that were developed for the Internet.

What is SOAP?

SOAP stands for *Simple Object Access Protocol*. SOAP is an XML-based wire protocol designed for decentralized, distributed networks such as the Internet. SOAP defines conventions that allow a computer to invoke a remote procedure in another. These remote procedure calls (SOAP requests) can be transported using a variety of network protocols.

For example, the SOAP Listener task on an IIS server uses HTTP protocol to transport SOAP messages to and from the Internet. Applications developed using SOAP Notifier COM components use Notifier protocol to transport SOAP messages to and from CIC server. SOAP itself is unconcerned with the protocol used for transport. For this reason, SOAP can be used on many types of computer networks.

SOAP makes it possible for programs running on different computers to request and receive data from one another in a structured way, even when different operating systems are used. SOAP provides the XML vocabulary needed to specify method parameters, return values, and exceptions.

SOAP empowers remote computers to start handlers on CIC and receive data from CIC in response. SOAP extends CIC interoperability to the entire Internet. Anything that "talks" SOAP through HTTP can communicate with CIC. Any computer platform (Windows, Unix, Linux, Mac, etc.) that can create and transport a SOAP message request can start a handler on CIC. Depending upon the type of request, the handler may or may not send back a response containing values looked up by CIC.

For example, a Unix Server might use *Enterprise JavaBeans* (EJB) to generate a SOAP Message requesting information about a user's status. When the request is received by CIC, it starts a handler that looks up the user's status, generates a SOAP response, and transports the response back to the requesting server. When the Unix system receives this *SOAP payload*, it uses another EJB to parse and process the information.

Conversely, handlers created using CIC's *SOAP tools* can request data from web services and remote procedures. For example, a handler might request the current price of a stock from a brokerage service, check inventory levels from an inventory management system, conduct a credit card transaction, or obtain a weather report. SOAP support in CIC is implemented by SOAP tools in Interaction Designer that define initiators, invoke remote procedures, process requests and payloads. SOAP messages are channeled through a SOAP ISAPI Listener task that runs on an IIS server. Developers can optionally use SOAP Notifier COM components to develop COM applications that directly invoke SOAP handlers. SOAP Notifier COM components are compatible with any language/application that supports Microsoft's Component Object Model. These options are discussed later in this document.

Who uses CIC's SOAP functionality?

SOAP tools support open standards (SOAP, XML, WSDL, etc.) These tools promote interoperability and are applicable to many types of application development. SOAP tools are primarily used by developers, advanced handler authors, and professional services personnel. However, the *services* created using SOAP tools are another matter. Anyone, anywhere on the Internet is potentially a consumer or provider of information processed by SOAP handlers. The possibilities are limitless.

For example, an caller might enter a PIN number into an auto-attendant menu created using Interaction Attendant. In turn, Attendant could start a SOAP handler that passes the PIN number to a remote web service to look up information that is spoken back to the caller using CIC's text-to-speech capability. A remote procedure invoked by SOAP can perform any kind of data processing tasks, ranging from a simple lookups to complex transactions that accept complex data types as input. SOAP does not impose any limits on the application functionality that can be invoked.

- OAP tools allow developers to create handlers that retrieve data from web services, or which function as web services. Handler-based services can be described using *Web Services Description Language (WSDL)*—an XML-based language that defines the functionality offered by a web service and how to access it. WSDL makes it possible to describe a service on CIC so that a worldwide audience can find and use it. WSDL describes the service, all parameters required to invoke it, and the location (endpoint) where the service can be accessed.
- WSDL's are not available for handler examples included with this release. However, you can easily create WSDL's to describe the example files.
- SOAP makes it possible for programs written in different languages and running on different platforms to communicate with each other.
- SOAP integrates CIC with business-to-business interactions and information services.
- Once a SOAP *endpoint* is exposed to the internet, a handler may call into the endpoint, which may be on the Internet or an Intranet.

SOAP is not appropriate for low-level, tightly-coupled transactions, due to network latency and the overhead imposed by the SOAP messaging encoding and decoding. SOAP is best suited for simple, high-level transactions, such as sending a name and PIN number to a service to obtain an account summary.

SOAP's Request/Response Model

CIC uses a *request/response* model to process SOAP requests. This mechanism should be familiar to anyone who has used a web browser.

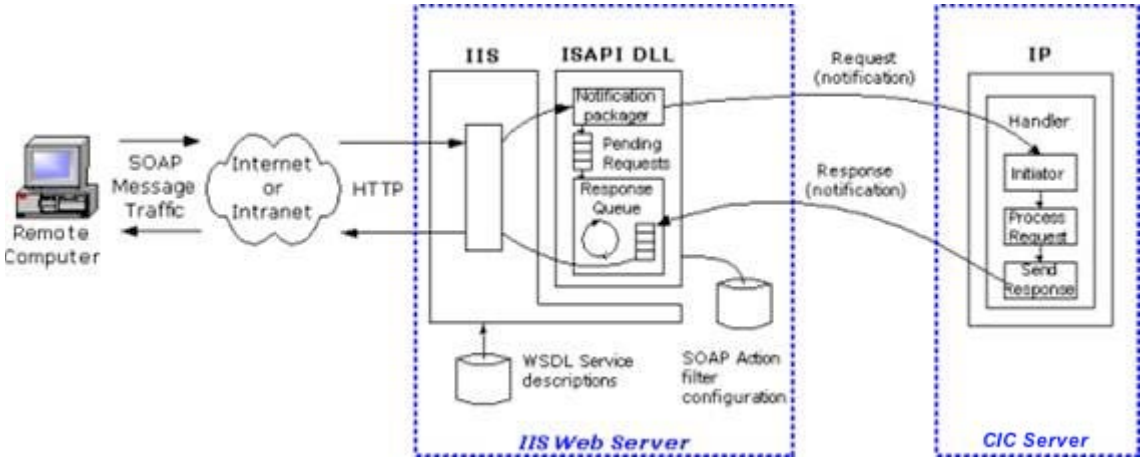
1. A *client* (e.g. web browser) connects to a *server* and passes a request (fetch a web page). The client then waits for the server to respond.
2. The server responds in one of two ways. It either returns the requested information, or it responds with an error message that tells the client why the request could not be completed.
3. Once the server has responded to the client, it closes the connection, discards all state information about the transaction, and listens for another request.

Web Services

In the world of SOAP, the client is a computer program that asks a server (another computer program) to execute a method (sometimes called a *web service*).

In CIC configuration, HTTP requests are received by *SOAP Listener*—an ISAPI DLL that runs on an IIS web server. SOAP Listener passes requests to the CIC Notifier subsystem for processing. Notifier alerts the Interaction Processor subsystem, which in turn starts the handler needed to process the request. Response data from the handler is passed back to the Listener task for transport to the remote computer.

In general, SOAP Listener translates HTTP requests into notifications and acts as a gatekeeper to prevent denial of service attacks.



On the receiving end, the response message is decoded and used by the requesting computer in some way. This low-overhead approach permits a single server to share information with many clients.

Requests and Responses are XML Documents

In order for the request/response model to work, messages must be formatted in a way that both computers understand. SOAP uses XML to accomplish this.

What is XML?

XML stands for *Extensible Markup Language*. XML provides a structured way to define data in plain text format, so that data can be exchanged between computers. SOAP messages are XML documents, which are just text files formatted according to some very specific guidelines. (SOAP is the specification that defines the guidelines used to describe remote procedure calls using XML.) XML provides the syntax needed to define a markup *vocabulary*—the tags and attributes needed to describe a particular type of data. XML files can be created using a simple text editor, such as Notepad. XML is more flexible than comma-delimited or fixed-length formats, since XML encloses information inside descriptive tags in a tree-based hierarchy. Before a SOAP request can be transported to another computer, the request is structured using XML so that the remote system can interpret the request in accordance with the SOAP specification. Responses from the remote procedure are returned as XML documents.

SOAP uses XML to package the data passed to a method, or received as a response. SOAP itself is nothing more than a set of rules that define how to describe method calls and return values using XML syntax. XML merely describes data, without consideration for the way that the data is processed or presented.

To summarize, SOAP defines conventions needed to invoke the methods of a web service. SOAP tools on CIC allow web services to be created using Interaction Designer. SOAP uses existing transport protocols (such as HTTP) to transmit an *XML payload* to another computer. The payload contains everything that the remote computer needs to execute a function (arguments and data). Services that understand SOAP requests can be expected to return XML responses in accordance with the rules of SOAP. The relationship between SOAP and XML can be expressed this way:

SOAP documents are XML documents that conform to a particular specification, allowing the exchange of messages. Therefore, to understand SOAP, you need a working knowledge of XML.

What is the relationship between XML and markup languages, such as HTML or SGML?

What is the relationship between XML and markup languages, such as HTML or SGML?

If you use the Internet, you probably know that HTML is the markup language used to create World Wide Web pages. (HTML stands for *Hypertext Markup Language*.) HTML and XML are both descendants of an earlier markup language called SGML (*Standard Generalized Markup Language*). SGML is a complicated set of rules that define document structures. XML is a subset of SGML that does the same thing, using fewer rules. Since XML is a less-complicated derivative of SGML, XML is more easily implemented on large networks such as the Internet. The primary role of XML is to *define data*.

XML delivers the power of SGML without the complexity. XML does not utilize features that make the authoring difficult or costly. Yet XML preserves most of the flexibility and richness associated with SGML.

Web browsers use a combined parsing and presentation engine that is tolerant of markup problems. Sloppy markup in HTML pages is ignored or interpreted in a proprietary way. For example, if a closing tag is omitted in an HTML document, the browser attempts to guess where the closing tag should have been. If the browser encounters a tag or attribute that it does not recognize (such as a tag supported by a different brand of browser), the tag or element is ignored.

The loose, uncontrolled nature of HTML makes it impossible to predict exactly how a web page will be displayed. Browsers attempt to render *something* on-screen, however odd, rather than display validation error messages. Since HTML is presentation-oriented, it uses markup tags for formatting as well as to define structure. The complexity of HTML formatting can make it difficult to locate data in HTML documents. HTML was not originally designed to provide *precise* control over the layout of page elements. To compensate, savvy page designers use tables, style sheets, and DHTML layers to control the placement of text and graphics. This creates visually-appealing web pages at the expense of clear-cut document structures. Complex web pages bury data in a mix of structures in the information stream. The lack of structural consistency in HTML documents makes it difficult for computer programs to locate, extract or update data. XML resolves this problem, by demanding that document authors get structure and syntax right.

XML Parsers

XML documents are often *parsed* to ensure that they are *valid* and *well-formed*.

- A *well-formed* document conforms to the XML specification.
- 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.

It is important to note the distinction between parsers and browsers. Parsers validate data. Browsers display information. SGML and XML are focused on *parsing* documents rather than presenting them. Parsing is the computer equivalent of reading a document. A parser is a program that reads in a text file, breaks it down into component parts, and validates the document using rules in a DTD file. Internet Explorer offers a built-in parser that you can use to validate XML files. For details, see Viewing XML in Internet Explorer.

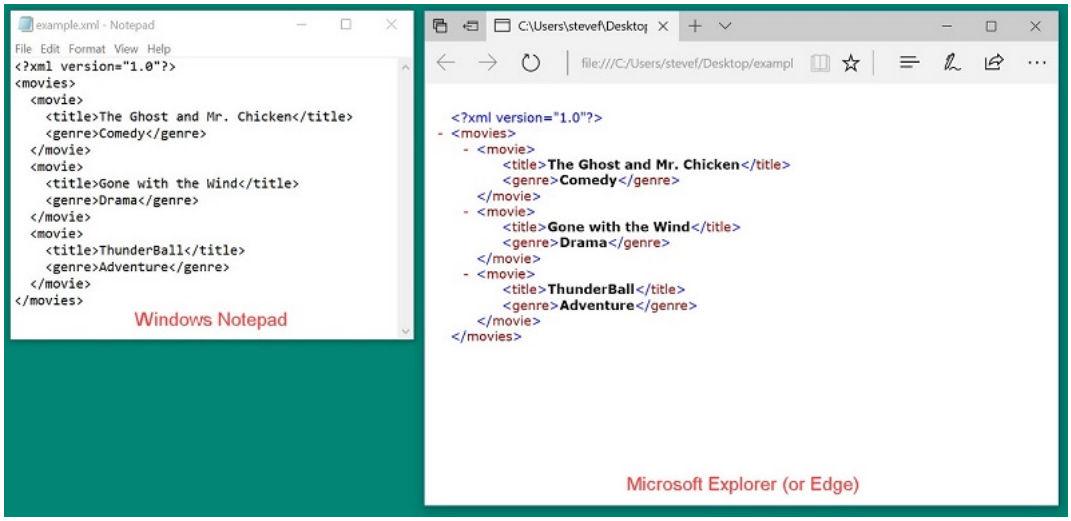
DTD stands for *Document Type Definition*. DTD's define hierarchy structure and elements that can be used in an XML document. For links to DTD tutorials, see Recommended Web Links.) The role of a parser is to identify portions of a document that are invalid in terms of structure or syntax. XML and SGML parsers ensure that documents are coded correctly.

Viewing XML in Internet Explorer or Edge

The tree structure of XML documents is easy to understand when seen visually. Microsoft's Edge and Internet Explorer 6 (or later) browsers provide a built-in parser that you can use parse, validate, and view XML files.

Tip: To open an XML file, drag and drop an XML file from Windows Explorer into your browser's document window. Or, double-click an .xml filename in Windows Explorer.

The figure below shows the sample movie database (sample1.xml) after it has been opened in Notepad and Internet Explorer. As you can see, Notepad displays the statements appear as they were entered. Edge and Internet Explorer display a tree of elements, which makes the content easier to view.



Edge and Internet Explorer automatically add DHTML code so that you can expand or collapse nodes in the tree. Internet Explorer doesn't allow you to do much besides view XML files. However, if you save your XML file with an extension of .htm or .html, IE will render the data contained in the XML file.

Advantages of XML over HTML

XML syntax closely resembles HTML; data is enclosed between opening and closing tags. However, XML is more flexible than HTML:

- XML encodes data in tightly-validated tree structures. Data is easy to locate since its context is well defined by tags and rules of structure.
- HTML attempts to control the appearance and presentation of data, while XML does not. XML defines data separately from its presentation. This makes XML data easier to locate and manipulate.
- XML is a standard data format that permits applications to exchange information across platforms and operating systems. HTML is markup used to display information in a web browser.
- XML is open and extensible. XML authors can create their own tags. HTML is limited by a fixed vocabulary that browser developers have agreed to support. In fact, XML has no predefined tags of its own. New XML tags are defined as needed —to define any type of data using syntactical rules that that permit browsers and XML Parsers to interpret proprietary tags on the fly. XML can describe any kind of data, such as a row in a table, a chemical formula, a financial transaction, a short story, or an object that exposes methods and properties—with equal finesse.
- Since XML is plain text, it is easily transmitted between computers and through firewalls. XML is more secure than binary files, since text files cannot be executed directly. Binary files, on the other hand, can contain malicious computer programs.
- XML is universally compatible. The XML file format is not tied to any particular program, operating system, database, or network. XML can be used by non-web applications to store data.
- XML files can be *transformed* into other types of documents. Transformation is controlled using XSL style sheets.

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. For these reasons, XML is becoming the preferred format for e-commerce and information exchange between computers of all types. XSL style sheets can reorder documents, display or hide information, or apply formatting, among many other things. XSL uses patterns and logical operations to determine which parts of a document tree it should transform. XSL works somewhat like a programming language—it can test for equality and perform processing based upon the results of a test.

Structure of an XML file

An XML file is just a structured text file. The best way to understand XML is to look at example files. Listing 1 below contains three records from a movie database. Each record contains two fields: the *title* of a movie, and its *genre*.

The example file is formatted using blank lines, tabs and white space that make the file easier to read. In practice, those items are ignored by XML parsers. Likewise, bold text and line numbers in the listing are for illustration purposes only. Actual XML files do not contain line numbers.

Listing 1: Sample XML File

```
1  <?xml version="1.0"?>
2  <movies>
3    <movie>
4       <title>The Ghost and Mr. Chicken</title>
5       <genre>Comedy</genre>
6    </movie>
7    <movie>
8       <title>Gone with the Wind</title>
9       <genre>Drama</genre>
10   </movie>
11   <movie>
12      <title>ThunderBall</title>
13      <genre>Adventure</genre>
14   </movie>
15 </movies>
```

XML Declaration

Line 1 contains a processing instruction known as the XML *declaration*. This statement tells parsers that the file contains XML. The remainder of the file is composed of XML *elements*. Each element consists of a *start tag* and an *end tag*. XML *data* is just information that appears between tags.

The terms *tag* and *element* are often used interchangeably. A tag is an identifier that defines something. An element is an instance of a set of tags. In our example, <title> is a tag, and <title>Gone with the Wind</title> is an element. Elements are the basic building blocks of HTML files. Elements can be nested inside of other elements.

Rules that govern tags

Tags are governed by a few basic rules:

- Tag names are case-sensitive. <movie>, <Movie>, and <MOVIE> are not equivalent. Attribute names are also case-sensitive.
- Tag names must begin with an alphabetic character, an underscore, or a colon.
- Tag and attribute names cannot begin with "xml", which is reserved.
- All tags must be closed. A start tag must be closed by a corresponding end tag. Empty elements with no attributes can use a backslash as a shortcut for the end tag (e.g. <movie/> is equivalent to <movie></movie>.

The Root Element

Line 2 defines the **root element**. Since an XML document is a tree of elements, each document has a single root element that denotes the beginning and end of the XML statements in the file. In the example, the root element begins with a start tag <movies> and is closed by an end tag </movies>. All other elements are nested inside the root element.

Child Elements

Line 3 identifies <movie> as a **child** of the <movies> root element. Parent-child relationships are common in XML files. Parent elements can have many children. All elements must be properly closed, meaning that each element has a start tag and an end tag. Likewise, tags must be balanced. The close tag of a child cannot appear after the close tag of its parent. For example:

```
<title>ThunderBall<genre>Adventure</title></genre> is incorrect.
<title>ThunderBall<genre>Adventure</genre></title> is correct.
```

Line 4 contains some data (the title of a movie) between tags that identify the data.

Line 5 contains a different data item. In this case, it is a movie category between genre tags.

Line 6 closes this movie element.

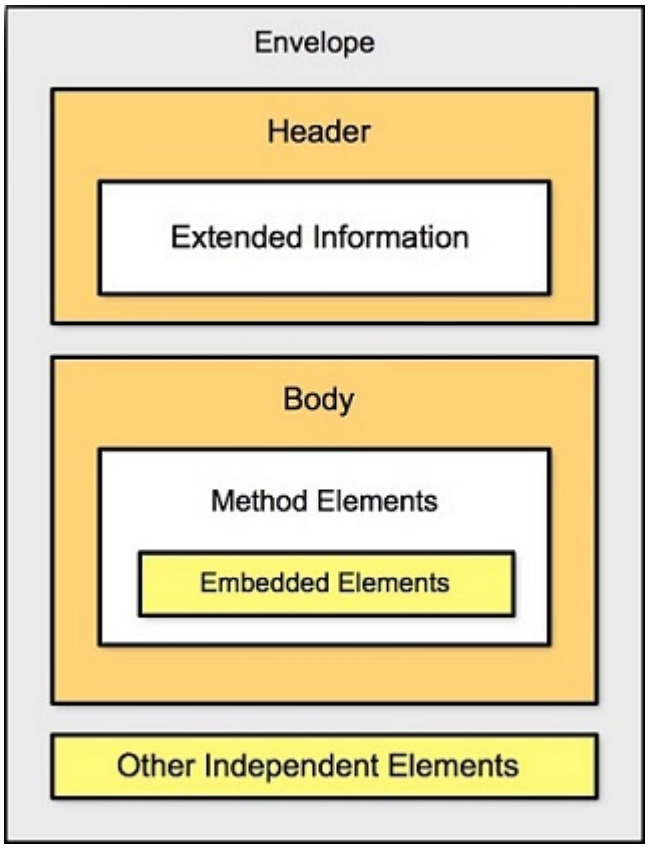
This basic structure is repeated in lines 7 through 14, which define two more records.

Line 15 contains the closing tag for the root element.

Structure of SOAP Messages

Structure of SOAP Messages

SOAP messages are constructed using a framework that describes what is in a SOAP message, and how it should be processed. This is known as the SOAP *envelope*.



SOAP messages may contain *encoding rules*, which express instances of application-defined data types. Remote procedure calls and responses are also described in a SOAP message. As mentioned earlier, there are two types of SOAP messages:

- ***Request*** messages ask a remote process to perform some sort of processing.
- ***Response*** messages are replies from a remote process that return data or an error message that indicates why the request could not be processed.

The ***payload*** contains data in XML format that is passed to or from a function. R*equest 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. SOAP uses XML to express payload information accurately and concisely. Every SOAP message has a main envelope section, which can contain header and body sub-sections.

Envelope Section

The envelope is always the outer most element. Everything else in a SOAP message appears inside **SOAP-ENV** tags. The envelope in Listing 2 is empty—it doesn't contain any header or body tags.

Listing 2: SOAP Envelope Elements

```
1 <SOAP-ENV:Envelope xmlns:SOAP-
ENV="http://schemas.xmlsoap.org/soap/envelope/"
2    SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"/>
3 </SOAP-ENV:Envelope>
```

---

1. Line 1 of the envelope refers an external XML *namespace* (**xmlns**) that defines elements and attributes that can appear in the envelope (such as header or body elements).
2. Namespacesresolve 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 an "Egg Quality" namespace.
3. Most SOAP envelopes refer to XML schema defined by the W3C. It is very common to see http://schemas.xmlsoap.org/soap/envelope/ as the namespace reference in a message envelope.

   XML Schema are the successor to DTDs for XML. XML schema 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.

4. Line 2 refers to **encodingStyle** schema that describes basic data types (Booleans, Integers, Strings, etc.) that can be passed to a remote procedure call. SOAP messages typically define encoding rules using the W3C schema at http://schemas.xmlsoap.org/soap/encoding/.
5. Line 3 closes the envelope.

Header Section

As mentioned earlier, the envelope can contain header and body sections. These are defined using **Header** and **Body** elements. Listing 3 shows a SOAP message with empty Header and Body sections.

Listing 3: Header and Body Sections of a SOAP Envelope

```
1  <SOAP-ENV:Envelope xmlns:SOAP-
ENV="http://schemas.xmlsoap.org/soap/envelope/"
2    SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"/>
3    <SOAP-ENV:Header>
4    </SOAP-ENV:Header>
5    <SOAP-ENV:Body>
6    </SOAP-ENV:Body>
```

```
7   </SOAP-ENV:Envelope>
```

As you can see, lines 3-4 define the Header section. Lines 5-6 define the Body. Other independent elements can optionally be defined inside the envelope, but for purposes of this discussion, we do not need to be concerned with independent elements. Refer to the W3C SOAP Specification at http://www.w3.org/TR/SOAP/ for more information about independent elements.

The Header section can contain *meta data* about the message. Meta data is "data that describes data". A SOAP message does not have to contain a Header. Header elements make it possible to extend the base SOAP protocol, to accommodate needs that the SOAP specification does not include.

For example, Header elements might maintain session information between a server and a client, or might contain authentication information about a transaction. A Header can contain any number of namespace-qualified child elements, each of which extends the default protocol in some way. Each header element provides extra content for processing the Body of the message.

Each Header element may be annotated with a "mustUnderstand" attribute, which indicates whether or not the element is mandatory. When "mustUnderstand" is True for an element, the server that processes the message must know how to interpret that element. If it doesn't, it must reject the message. Headers that do not have a "mustUnderstand" attribute, or which have this attribute set False, are considered to be optional, meaning that the recipient server is allowed to process the message as best it can.

Body Section

The most important part of a SOAP message is the Body section, since it contains the message's payload. In a *request* message, the Body defines the method to execute, and parameters that must be passed to it. The Body of a *response* message contains references to the method called, and return values from the method. If an error occurs, the response contains information about the fault. To better understand these concepts, let's look at some actual request/response messages. The request message in Listing 4 invokes a simple method that adds two numbers. Listing 5 contains the response from the web service.

Request Messages

Listing 4: Request to Invoke Add Method

```
1   <SOAP-ENV:Envelope
2           xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
3           SOAP-4
ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
5     <SOAP-ENV:Body>
6        <m:Add xmlns:m="uri:my-calculator">
7           <Parameter1>2</Parameter1>
8           <Parameter2>3</Parameter2>
9        </m:Add>
```

```
10    </SOAP-ENV:Body>
11 </SOAP-ENV:Envelope>
```

The **m:Add** method name element in line 6 contains the name of the method (Add) we wish to call, and the namespace it is found in (uri:my-calculator). The URI (Universal Resource Indicator) specifies which computer offers an Add method web service.

Lines 7-8 define two arguments (Parameter1 and Parameter2) that the Add method requires. In this example, the numbers to be added are 2 and 3.

Line 9 closes the method name element.

Response Messages

The response from the computer at **uri:my-calculator** is listed below. This response message contains return values from the Add method. By convention, "Response" is appended to the name of the method called. However, the format of the method name can also be defined using WSDL.

Listing 5: Response from the Add Method

```
1  <SOAP-ENV:Envelope
2          xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
3          SOAP-
ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"/>
4     <SOAP-ENV:Body>
5        <m:AddResponse xmlns:m="uri:my-calculator">
6           <Result>5</Result>
7        </m:AddResponse>
8     </SOAP-ENV:Body>
9  </SOAP-ENV:Envelope>
```

Line 5 identifies the remote procedure call. The Result tag in line 5 contains the sum of 2+3, which is 5. Note that the response message does not contain any of the data passed to call the function. Responses contain a return value from the function, or a fault message that indicates why the function call failed.

Fault Messages

When a message is rejected, the server generates a Fault, or error message. Faults are commonly caused by unrecognizable header fields, messages that cannot be authenticated, or problems that occurred when the server attempted to invoke a method or process the message.

Listing 6: A Typical Fault Response Message

```
<S:Envelope xmlns:S='http://schemas.xmlsoap.org/soap/envelope/'>
  <S:Body>
    <S:Fault>
       <faultcode>S:Server</faultcode>
       <faultstring>S:Server</faultstring>
       <detail>
         <e:mydetails xmlns:e="http://foo.com/detail">Some Error
Message</e:mydetails>
       </detail>
    </S:Fault>
  </S:Body>
</S:Envelope>
```

CIC's SOAP Components

CIC's SOAP Components

The SOAP components in a Customer Interaction Center environment are:

1. **Interaction Designer SOAP Components:** When Interaction Designer is installed, SOAP tools, SOAP Tool help, and a SOAP message trace utility are installed to the C:\I3\IC\Install\Admin\IC_Admin directory on the CIC server.

   SOAP tools are implemented in a dynamic link library named SOAPToolsIDA.DLL. When Interaction Designer starts up, it adds the tools defined in this DLL to Interaction Designer's tool palette. SOAP Tools are always installed with Interaction Designer. See SOAP Tools in Interaction Designer for more information. Context-sensitive online help for SOAP tools is available in Interaction Designer. Soap help topics are displayed when a SOAP tool has focus in Interaction Designer and the F1 key is pressed.

2. **SOAP Tracer Utility:** (SOAPTracerA.exe) is optionally installed with Interaction Designer when the "SOAP" option is selected. It permits users to "spy" on SOAP notification traffic. Soap Tracer displays request and response packets in a list and allows inspection of request and response payloads. For usage information, see SOAP Tracer Utility later in this section.
3. The **SOAP ISAPI Listener: Task** is responsible for parsing incoming SOAP requests, dispatching requests to the appropriate method, and packaging return values into outgoing SOAP responses. This process runs on an IIS Server. See SOAP ISAPI Listener Task for IIS later in this section.
4. **SOAP Notifier COM Objects** issue SOAP notifications from automation compatible applications. These components provide a high-performance method of initiating handlers without incurring the performance penalty of HTTP-based Listener operations. Third-party applications created using the SOAP Notifier COM components can directly create and forward packets to Interaction Processor, bypassing the need to create packets received using HTTP and the Soap Listener task. See SOAP Notifer COM Objects later in this section.

SOAP Tools in Interaction Designer

This topic summarizes SOAP-related tools in Interaction Designer that create handlers to process SOAP requests or responses. For additional information, see Appendix B: SOAP Tools.

Initiator Tools

SOAP Initiator

> This initiator triggers if the "Notification Event" of the request matches a specified string. The Notification Event on which the Initiator triggers is specified in the property dialog.

Request Tools

SOAP Get Request Info

> Queries some information from the request handle.

SOAP Abort Request

> Aborts the request. Aborting a request is useful if a SOAP request handler is registered as a Monitor handler.

SOAP Get Transport Info

> Returns an XML document containing transport specific (header) data. It allows the client to include any kind of out-of-band data in the request.

SOAP Expects Response

> Takes a different exit path depending on whether the SOAP request requires a response (YES) or not (NO).

SOAP Parse Request Payload

> Parses the payload of the request into an XML document.

SOAP Send Response

> Sends the specified payload as response to the sender of the request. To support transport specific features, the "Transport Control Data" argument takes an XML node whose content will be sent back to the client. It can be used to send transport specific out-of-band data to the client.

Payload Processing Tools

SOAP Create Envelope

Creates a new SOAP envelope.

SOAP Get Body

Retrieves the Body element from the SOAP envelope. A body must exist. If it can't be found, the tool exits through "Failure" and attaches error information to the envelope.

SOAP Get Body Element

Retrieves the first body element that matches the given base name and namespace.

SOAP Add Body Element

Adds an entry to the body of the SOAP envelope.

SOAP Query Encoding Style

Matches a space separated list of URIs against the "encodingStyle" attribute of the given element. If the element doesn't have an ‘encodingStyle' attribute, the parent of the element is checked and so on, until an element with an "encodingStyle" attribute is found. If that attribute contains any of the specified encoding style URIs, the tool returns through "Found" and returns the style that was found.

SOAP Get Header

Retrieves the header element from the SOAP envelope if it has one.

SOAP Get Header Element

Retrieves the first header element that matches the given base name and namespace. Returns the first element in the header if neither a name nor namespace is given. Takes "Not Found" exit if the envelope doesn't have a header or the element can't be found.

SOAP Get Header Elements

Returns iterator to a list of header elements filtered by the given arguments. Takes the "None" exit if envelope has no header or none of the header elements matched the filter criteria.

SOAP Add Header Element

Creates a header element and adds it to the given envelope. If the envelope does not have a header, one is inserted before the Body element.

SOAP Get Fault

Retrieves fault information from the SOAP envelope. If there is no style="color: #0e5470;"><Fault> element in the envelope, the "No Fault" exit is taken and NULL elements and empty strings are returned.

SOAP Set Fault

Adds a style="color: #0e5470;"><Fault> element to the envelope or replaces an existing one.

SOAP Create Fault Response

Copies the request envelope and replaces all children of the style="color: #0e5470;"><Body> element with a single style="color: #0e5470;"><Fault> element. It combines the functionality of the "SOAP Create Envelope" tool with the functionality of the "SOAP Set Fault" tool.

SOAP Get RPC Parameter

This is a convenience tool for cracking RPC requests. It retrieves a parameter element (child) from the first element in the <Body> element (method in an RPC request). It returns the first element that matches all of the specified arguments.

SOAP Add RPC Parameter

This is a convenience tool for composing RPC requests or responses. It adds a parameter element to the first element in the body of the envelope, which represents the method in RPC requests. Use the XML tools to add complex data (not just a string) to the parameter by manipulating the returned "Parameter Element" node.

SOAP Get RPC Method Info

This is a convenience tool for cracking RPC requests. It retrieves the first child element of the SOAP <Body> element (Method element in RPC requests). It returns a collection containing the child elements of the method, which constitute the method arguments.

SOAP Get Next RPC Parameter

This tool returns the element node at the current iterator position and returns an iterator to the next position.

SOAP Create RPC Response

This is a convenience tool for composing the response envelope for an RPC request. It copies the source envelope and replaces the method element in the body with an element that has the same name but "Response" added to its name. It also adds a <Result> element as child of the method element.

SOAP Set Element Type

In SOAP, the type of an argument or the return value is specified by the service description and doesn't need to be included in the payload. However, the service may define the type as xsd:anyType, for VARIANT types. This tool allows you to include the type in the argument.

SOAP Create Array

Turns an element, for example an RPC parameter, into a SOAP array. The array is created for values supplied as list of strings or just a number of empty elements that can be populated with complex data.

Invocation Tools

SOAP HTTP Request

This tool issues an HTTP request to the specified URL with the SOAP request envelope as payload. The response body is parsed and returned as response envelope.

Helper Tools

SOAP Base64 Encode

Converts a supplied UNICODE string to the specified character set (default = UTF-8) and encodes the resulting data into a Base64 string. Characters that cannot be translated to the destination character set will be represented as '?'. Wide character sets, such as UTF-16 are currently not supported.

SOAP Base64 Decode

Decodes the base64 encoded string into the binary representation and converts it to UNICODE based on the specified character set.

SOAP Base64 Encode File

Reads the specified file as binary data and encodes it into a base64 string. This tool can be used to send any kind of data through SOAP requests. For example, you could encode a wave file in a SOAP message.

SOAP Base64 Decode To File

Decodes the base64 encoded string into the binary representation and writes the data to the specified file as binary data.

The SOAP Tracer Utility

The SOAP Tracer Utility

SOAP Tracer is used to debug SOAP requests and responses. It displays notifications exchanged between the client (SOAP Notifier COM or ISAPI listener) and the CIC server. It spies on SOAP notification messages. It records and displays request and response packets in a list and allows inspection of request and response payloads. Filtering for particular SOAP actions or clients is not supported in the current release, but may be added in the future. SOAP Tracer is optionally installed on the CIC server when Interaction Designer is installed—if the "SOAP" option is selected. It can be used from any machine that has access to a CIC server through a Notifier connection. However, SOAP Tracer is unrelated to Interaction Designer. It is also unrelated to ISAPI Listener.

IMPORTANT—Do not run SOAP Tracer for extended periods of time. It can consume a lot of memory and may degrade performance of CIC.

Starting SOAP Tracer

The default shortcut created under *Program Files* > PureConnect > *SOAP Tracer* is:

```
C:\I3\IC\Install\Admin\IC_Admin\SOAPTracerAD.exe /notifier=localhost
```

To run this utility, press the *Start* button, then select *Programs* > PureConnect > *Soap Tracer*. SOAP Tracer optionally accepts the command line arguments listed below.

Command Line Arguments

/NOTIFIER=<hostname>

Hostname or IP address of the Notifier server. Default: default Notifier server of the user.

/USER=<username>

User name of the CIC user. Default: current user

/PASSWORD=<password>

Password of the CIC user.

/TEMPDIR=<directory>

Directory in which to store the temporary files generated by the utility. Default is the system's TEMP directory.

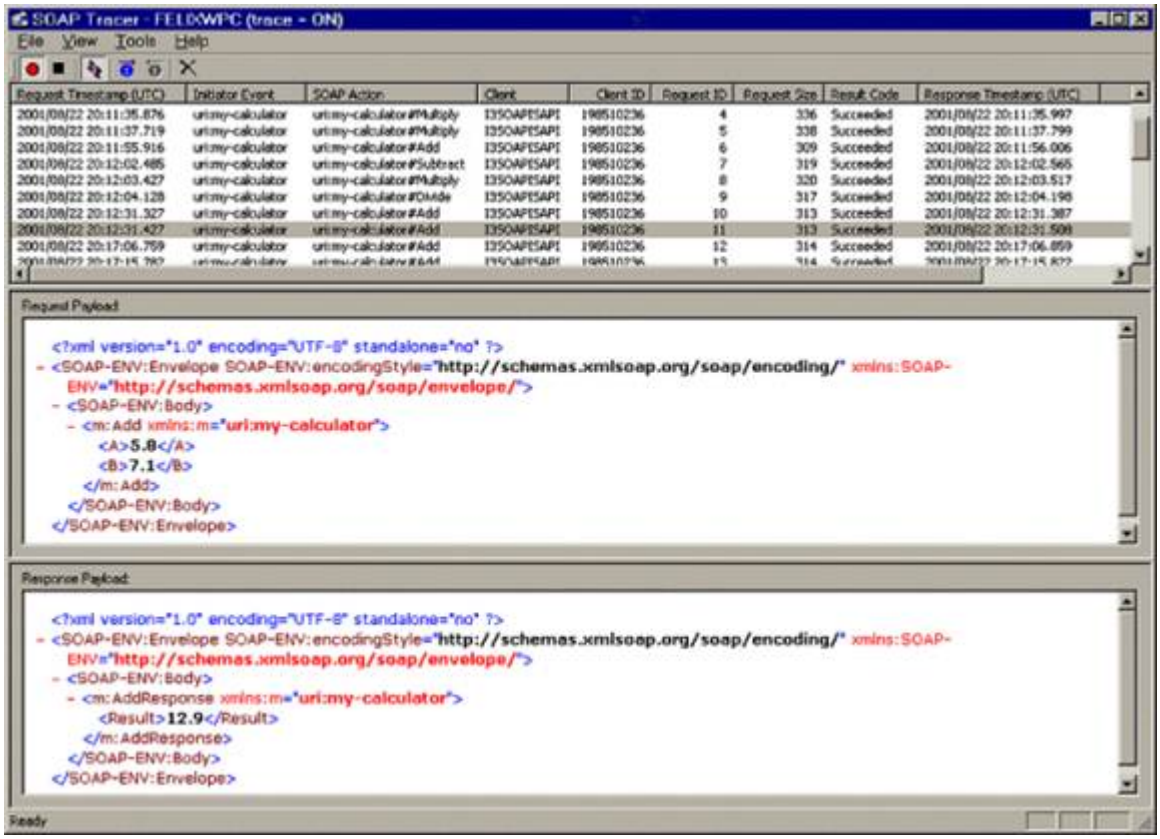
/STARTCAPTURE

If this argument is specified, the SOAP Tracer immediately starts capturing SOAP notifications. If not specified, the capture must be started by selecting Start Capture from the Tools menu, or by pressing the corresponding tool bar button for this command.

/KEEPTEMPFILES

By default, temporary files used to store the SOAP payload are deleted when the traces are cleared or the utility is exited. When this switch is specified, SOAP Tracer won't delete its temporary files automatically.

SOAP Tracer's User Interface



The SOAP Tracer window is divided into three panes. Users select a message in the top pane to display corresponding request and response messages in the other panes.

The Request List Pane

The top pane is the Request List. It displays information about SOAP requests. such as the name of the client who issued the request, the time, and whether or not the request succeeded. This pane contains the following columns:

Request Timestamp (UTC)

Date and time in UTC when request notification was recorded.

Initiator Event

Notification Event of the request (often same as SOAP Action).

SOAP Action

SOAP Action of the request.

Client

Name of the client issuing the request.

Client ID

Dynamic identifier of the client.

Request ID

Identifier of the request (generated by and scoped to client).

Request Size

Size of the request payload (SOAP envelope) in bytes.

Result Code

Result code returned by the server.

Succeeded

Request successfully processed

Failed

Request failed, server returned SOAP fault

Unhandled

Request was not handled by the server

Response Timestamp (UTC)

Date and time in UTC when response notification was recorded

Duration

Difference between Response Timestamp and Request Timestamp

Response Size

Size of the response payload data in bytes.

Key Term—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.

The Request Payload Pane

The Request Payload pane displays XML payload data that was sent to the handler for the selected request.

The Response Payload Pane

The Response Payload pane panes display payload data that was sent back to the client by the handler.

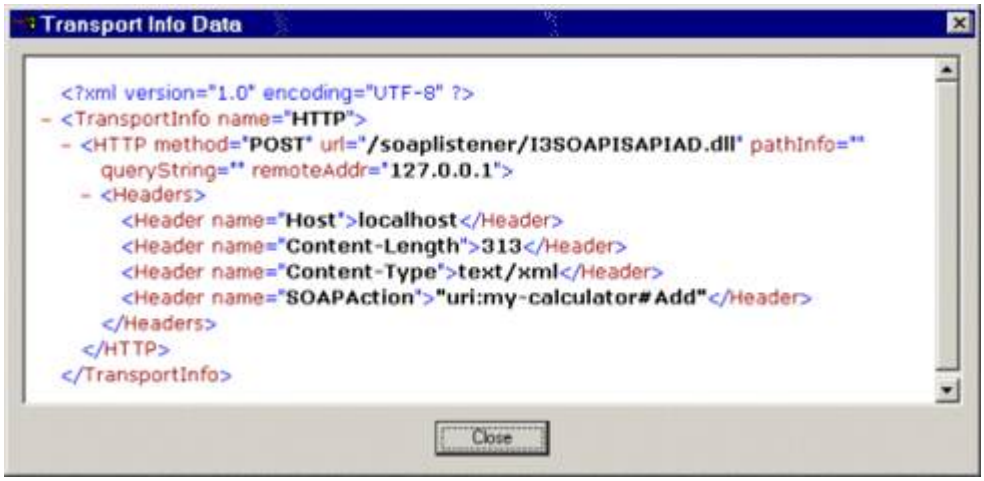
Menu Commands

File > Exit

Closes Soap Tracer.

View > Transport Info Data

Displays a dialog containing the transport info data of the request. This option is enabled if the SOAP request notification included transport information data.



View > Transport Control Data

Displays a dialog containing the transport control data of the response. This option is enabled if the SOAP request notification included transport control data.

View > Follow Requests

If checked, the selection in the request list will follow the recorded requests and always select the most recent one.

View > Toolbar

Hides or displays toolbar icons.

View > Status Bar

Hides or displays the status bar.

Tools > Start Capture

Starts recording the SOAP notification traffic.

Tools > Stop Capture

Stops recording the SOAP notification traffic.

Tools > Clear View

Clears the list of recorded SOAP notifications.

Tools > Settings

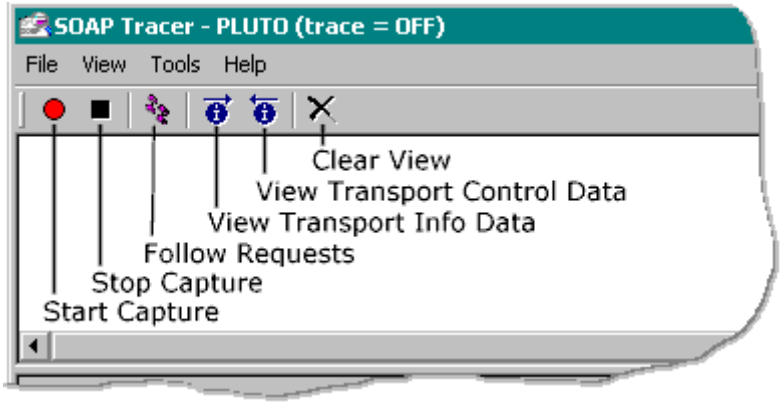
Displays dialog used to configure the application. This feature is disabled in the current release.

Help > About SOAPTracer…

Opens a dialog that displays copyright information.

Toolbar

Some SOAP Tracer commands have tool bar equivalents.



Tool bar icons in SOAP Tracer.

SOAP ISAPI Listener Task for IIS

SOAP ISAPI Listener translates HTTP SOAP packets into notifications and sends them to the CIC server. The SOAP ISAPI Listener must be installed on a machine that has IIS installed.

What is a Listener?

A *listener* receives incoming HTTP messages that contain SOAP requests for some type of service. It parses these messages, decides whether to process the request (based upon threshold values and filter configurations), and dispatches the request to the appropriate method for processing. If the service returns a response, the listener packages the response into an HTTP payload, and sends that back to the client. A listener also handles requests for WSDL information about web services.

The *SOAP ISAPI Listener* looks at incoming SOAP requests, decides whether requests should be forwarded to CIC to invoke a handler, and forwards appropriate requests to CIC's Notifier subsystem, which in turn calls Interaction Processor to invoke the handler associated with the initiator specified in the incoming message. SOAP ISAPI listener and packages return values from handlers into outgoing HTTP responses, and sends them to the client. If the listener decides not to forward a request to CIC for processing, it returns a fault message (SOAP and/or HTTP) to the requesting client application.

What is ISAPI?

The SOAP ISAPI Listener is sometimes called the *SOAP ISAPI DLL*, since it is a dynamic link library developed in conformance with Microsoft's Internet Server Application Programming Interface (ISAPI). ISAPI allows developers to extend the functionality of Microsoft's Internet Information Server (IIS). The component that implements the ISAPI Soap Listener task is I3SOAPISAPIU.DLL. This DLL is installed by the Interaction Center SOAP Listener Install to the IIS server of your choice. It translates HTTP requests into notifications and acts as a gatekeeper to prevent denial of service attacks. An ISAPI DLL is not a COM DLL. To invoke an ISAPI DLL, it must be explicitly referenced in a HTTP header. For example:

```
http://www.foo.com/virtual_directory_name/I3SOAPISAPIU.DLL
```

The virtual directory name is optional, so long as the server can resolve the location of the DLL.

What is an endpoint?

SOAP invokes methods at HTTP *endpoints*. An *endpoint* is a URL that uniquely identifies a namespace URI (Universal Resource Indicator), and the name of the method to execute (known as the NCName). Consider the following endpoint:

```
uri:my-calculator#Add"
```

The URI namespace (my-calculator) identifies the code module that contains the method to be called (Add), just as an interface name scopes a method in Java, CORBA, or COM. The namespace and the method name are separated by a pound sign.

When a SOAP request is transported to invoke the method, the endpoint name is passed in the SOAPMethodName header of the HTTP POST request. Consider the following sample HTTP header:

```
POST /objectURI HTTP/1.1
```

```
Host: www.foo.com

SOAPMethodName: urn:foo.com:my-calculator#Add

Content-Type: text/xml

Content-Length: nnnn
```

The HTTP header indicates that the **Add** method (from the **urn:foo.com:my-calculator** namespace) should be invoked against the endpoint identified by http://www.foo.com/objectURI.. The rest of the HTTP request is an XML document that contains additional information needed to invoke the request, such as parameters passed to the method. The server-side software that receives the request (e.g. the SOAP ISAPI Listener) is responsible for processing the request. Unlike other RPC protocols, SOAP doesn't define specific actions that must occur when a request is received. It leaves the implementation details to the process running at the endpoint. See http://www.w3.org/TR/REC-xml-names/

SOAP Notifier COM Objects

SOAP Notifier COM is a set of software components that allow custom applications to invoke handlers. SOAP Notifier COM objects issue SOAP notifications from automation compatible applications. Microsoft's .NET framework makes it possible for programmers to invoke a web service as if they were invoking a method of an object. SOAP Notifier COM components provide a high-performance method of calling handlers without incurring the performance penalty of HTTP-based Listener operations. Third-party applications created using the SOAP Notifier COM components can directly create and forward packets to Interaction Processor, bypassing the need to create packets received using HTTP and the Soap Listener task. These packets that are identical to those created by SOAP ISAPI Listener. However, the process is faster than HTTP-based Listener operations.

SOAP Notifier COM is appropriate for Windows client workstations that can run COM applications. It is not appropriate for operating systems (Linux, for example) that do not support COM. SOAP Notifier COM Components Setup registers SOAP Notifer COM API components on desktop PCs used to develop or run SOAP Notifier COM API applications. Soap_Notifier_COM_API_DG.chm is the *SOAP Notifier COM API Developer's Guide*. It describes interfaces, methods, and properties of the SOAP Notifier COM API. You will find this publication in the System APIs section of the PureConnect Documentation Library.

ISoapConnector: the MSSOAP Notifier Connector

ProgId: ININ.MSSOAPNotifierConnector

The SOAP Notifier COM API provides a component named **ISoapConnector** that is used to initiate SOAP handlers. Programmers can invoke a web service as easily as invoking a method on an object. 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. Instead of the SoapClient, you may use the MSSOAPLib.SoapSerializer and MSSOAPLib.SoapReader objects with any object that uses a ISoapConnector.

```
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>...)
```

Properties

SOAP Notifier Connector supports the following 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).

Related Topics

Appendix C: Structure of IP Notification Messages

Install and Configure SOAP ISAPI Listener

Install and Configure SOAP ISAPI Listener

The components of SOAP follow the client/server model. Some components are installed when Interaction Designer is installed on the CIC Server. Other components are installed on IIS web servers and client PCs. This section explains how to install and configure SOAP Tools, SOAP Tracer, SOAP Listener, and Soap Notifier COM components.

- **SOAP Tools Installation**: When Interaction Designer is installed (as part of the CIC Admin setup), new SOAP tools are added to Interaction Designer's tool palette. SOAP tools are implemented in a DLL (SOAPToolsIDA.DLL) that is installed with Interaction Designer. Appendix B in this document also contains a summary of each SOAP Tool.
- **SOAP Tracer Installation**: The Soap Tracer Utility (SOAPTracerA.exe) is optionally installed if the "SOAP" option is selected during installation of Interaction Designer.
- **SOAP ISAPI Listener Installation**: The *Interaction Center SOAP Listener Install* installs the SOAP Listener Task on an IIS server. The SOAP Listener task is an ISAPI DLL. Installation requires preplanning on your part to address security and configuration issues, and some post-installation work to customize the default SOAP filter configuration. The SOAP ISAPI DLL must be installed on a server running Microsoft Internet Information Server (IIS), version 5 or later.

Installation and configuration pre-planning

This section describes issues that SOAP implementers must resolve before installing ISAPI SOAP Listener on an IIS server. Security issues are particular important to consider if you plan to pass SOAP requests across the Internet.

1. **Select a server to host SOAP ISAPI Listener.**

   The SOAP Listener task is an ISAPI DLL that you must install on a computer running Microsoft's Internet Information Server (IIS) service. SOAP Listener uses IIS (version 5 or later) solely for HTTP operations. It does not consume other IIS services. You can install this task on a dedicated IIS server, or on a CIC server that is running IIS. Before choosing a platform, you should carefully consider security, performance, and capacity issues.

SOAP Listener will work if it is installed on a CIC server running IIS. Theoretically, this could improve performance by eliminating latency between CIC and a dedicated IIS server. In practice, performance could be degraded if the CIC server becomes too highly tasked, and this configuration could compromise network security. As a rule of thumb, *do not* install SOAP ISAPI Listener on a CIC server unless:

Port 80 HTTP traffic is tightly controlled (e.g. SOAP will be used exclusively for interactions between servers inside a firewall). This is appropriate for some corporate Intranets. Use a different port than 80 (e.g. 8080) that is blocked by the firewall. SOAP requests will not be received from the Internet, or use another port. The CIC server has the capacity to run IIS without degrading performance.

If SOAP requests will be received from the Internet, you should install SOAP ISAPI Listener on an IIS server in a DMZ (Demilitarized Zone) between two firewalls. This can be an existing CIC/web server or a dedicated web server.

2. Open port 2633 on the firewall between the DMZ and the Intranet on which the CIC server is located, so that Notifier traffic can pass between the CIC server and the SOAP listener. Do not open port 2633 to the Internet.

   What is a Demilitarized Zone?
   In an Internet-connected world, any public access server, such as a web server that connects outside of an internal network is unprotected against hacking. A public access server can expose the rest of a network to potential intrusion.
   Demilitarized zones (DMZ) reduce security risks by using multiple firewalls to delimit an internal network from publicly connected devices, such as web servers. A DMZ configuration protects both public servers and the internal network. The first firewall isolates essential Internet services (web, email, DNS, etc). The second firewall protects the internal network.
   A DMZ is not the only solution that you might employ to protect your network. It is completely acceptable to use different security measures. The exact method is up to you—be reminded that if you connect a server to the outside world, you must manage the risk that your internal network might be penetrated through a public server.

3. **Apply service packs and hotfixes to IIS.**

   Network security is a topic outside the scope of this paper. However, we strongly recommend that you keep server operating system and IIS software up-to-date. Apply Microsoft service

packs and hot fixes regularly. Hackers frequently exploit known security holes that you can close by applying free software updates. You can automate this process to a limited extent. For example, Microsoft's HFNETCHK is an executable that runs on your server. It retrieves an XML file that contains information about security hot fixes that your system might need. Browse Microsoft's web site (http://www.microsoft.com/security) for security bulletins, upgrades and other information. As a rule of thumb, you should not install services that you do not need. Subscribe to "NTBugtraq" or a similar discussion list. This mailing list discusses security exploits and security bugs in Windows NT, Windows 2000, and Windows XP plus related applications. To sign up, visit http://www.ntbugtraq.com/.

4. **Decide how to configure SOAP Listener to prevent DoS Attacks**

Denial of Service (DoS) Attacks are attempts to flood a server with false requests for information, with the goal of overwhelming the system and ultimately crashing it. Not much can be done to prevent a denial of service attack. However, you can minimize the impact of DoS attacks by supplying the a couple of threshold values at installation time, and by customizing an ISAPI filter after installation is complete.

Default Request Timeout

Since DoS attacks can degrade performance of the CIC Server, ISAPI Listener can be configured (at installation time) to return a fault message (Server.RequestTimeout) if the CIC Server fails to respond within a specific time interval.

Before installing SOAP Listener, decide what value to enter into the *Default Request Timeout* field. This value sets the maximum amount of time in milliseconds that ISAPI Listener will wait for the CIC Server to respond to a SOAP request. When this interval is exceeded, ISAPI Listener sends a fault message to the requester. The default is 60,000 milliseconds (1 minute). If your IIS server has a fast processor, and is dedicated to IIS, you may be able to reduce the default value.

This value sets the default timeout for all SOAPActions. Following installation, you can assign timeout values to specific SOAPActions, by editing a configuration file. For details, see Step 2: Set SOAPAction-Specific Timeout Values in the Post-Installation Procedures section of this document.

Maximum SOAP Payload Size

SOAP ISAPI Listener uses a threshold setting named *Maximum SOAP Payload Size* to limit the size of incoming SOAP messages. By default, the maximum SOAP payload Size is 128 KB. Larger messages are not forwarded by the Listener to the CIC Server for processing. Based upon the size of data passed to your handlers, you may be able to reduce this value significantly. This helps minimize the impact of denial of service (DoS) attacks.

Maximum Pending Requests

The *Maximum Pending Requests* threshold limits the maximum number of SOAP requests that the CIC server should process concurrently. It helps to think of this as the maximum number of

pending *responses* that SOAP Listener will wait for at any given time, since SOAP Listener waits for a response to each request that it sends to CIC.

If Listener finds itself waiting for more responses that are allowed, it stops sending additional inbound request messages to the CIC Server until the number of pending requests falls below the threshold. SOAP ISAPI Listener does not queue unprocessed requests. It fails unprocessed requests with a fault message (Server.TooBusy).

Process Isolation Level

There is one last setting that you must consider before installing SOAP ISAPI Listener, and that is the level of process isolation (Low or High) that you wish to assign to the ISAPI Listener DLL. Process isolation protects the main IIS process against application faults—in this case, against potential failure of the ISAPI Listener DLL .

Process Isolation provides an additional layer of durability for your Web server. *Low* process isolation provides the best performance. *High* process isolation offers more protection against possible faults in the Listener application (unlikely). *Low* is the default.

Install SOAP Listener

If at this point, if you have IIS running with the latest service packs and hot fixes, behind an acceptable firewall configuration, and have formulated threshold values, you are ready to install SOAP Listener. This procedure explains how to run the ***SOAP Listener Setup*** to install, register, and configure the SOAP Listener task on an IIS server. The Soap Listener task is an ISAPI DLL that translates HTTP requests into notifications. It acts as a gatekeeper to prevent denial of service attacks. Complete this procedure at your dedicated IIS Server or CIC Server running IIS. Installation requires pre-planning on your part to address security and configuration issues. If you have not read the Installation and configuration pre-planning section, we strongly recommend that you do so before performing this procedure.

1. Download the CIC 2018 R1 or later .iso file from the Genesys Product Information site at https://my.inin.com/products/Pages/Downloads.aspx.
2. Copy the .iso file to a file server (non-CIC server) with a high bandwidth connection to the server(s) on which you will be running the CIC 2018 R1 or later installs.
3. Mount the .iso file and share the contents to make them accessible to the server(s) on which you will be running the CIC 2018 R1 or later installs.
4. Navigate to the \Installs\Off-ServerComponents directory on the file server.
5. Copy the SOAP Listener .msi file, for example, SOAPListener_2018_R1.msi, to the server on which you plan to run this install and double-click to launch it.

   The welcome page appears.

6. Press Next to proceed past the welcome screen. Then press Next a second time to accept installation of default features.
7. Supply user name, domain password, and domain for a user account with administrative privileges on the CIC server. Then press **Next**.

8. Type the name of the CIC server. Then press **Next**.
9. Supply values as indicated below:

   Default Request Timeout (in seconds)

   Enter the number of seconds that the ISAPI Listener should wait for the CIC Server to respond (to a SOAP request) before timing out and returning a fault message. The default value is 0 seconds. Press **Next** to proceed.

   Maximum Pending Requests

   Specify the maximum number of SOAP requests that your CIC server should handle concurrently during peak periods. This helps protect your server from denial of service (DoS) attacks. When this value is exceeded, additional requests will be denied.

   Maximum SOAP Payload Size (in KB)

   Specify the maximum size (in kilobytes) of SOAP payloads sent by the SOAP Listener to the CIC Server. Larger XML payloads will not be forwarded, to minimize the risk of denial of service (DoS) attacks.

10. Press **Next** to proceed. The next screen prompts for a location where log files will be stored. Accept the default path, or navigate to a different path. When you are finished, click Next.
11. Click **Install** to begin installing files.
12. Press **Finish** to exit Setup.
13. Click **Yes** to restart.
14. For the SOAP Listener machine to receive updates from the Interactive Update Provider on the CIC Server, you must run the *Interactive Update Client install* following the *SOAP Listener* install. The install will prompt for the Interactive Update Provider Server (CIC Server) name or IP address.

Post-installation procedures

Following installation of ISAPI SOAP Listener, you should complete additional security steps to defend against DoS Attack. Specifically, you should limit requests to known SOAPActions, and to assign timeout values to individual SOAPActions. You will modify the default ISAPI filter configuration file. The relative path to this file is ..\soaplistener\filter\I3SOAPISAPIConfig.xml. The SOAP ISAPI endpoint listener uses I3SOAPISAPIConfig.xml to filter incoming message requests. This file acts as a gatekeeper. It affects whether or not incoming messages are forwarded to the CIC Server by ISAPI Listener. Implementers are strongly encouraged to edit I3SOAPISAPIConfig.xml immediately after SOAP ISAPI Listener is installed, and whenever new handlers implement an additional SOAPAction.

The default configuration indiscriminately forwards all SOAP requests to the Interaction Center server identified in the ISAPI Listener install. You should modify the filter file to make the following modifications:

1. Add <Rule> elements that identify the specific operations (SOAPActions) that your CIC server should process. Thereafter, SOAP ISAPI Listener will forward only those particular SOAPActions to the CIC server.
2. Set timeout thresholds for specific SOAPActions used in your environment.

These modifications are particularly important if your SOAP Listener is exposed to the Internet. If you leave the default filter unchanged, your CIC server is more venerable to DoS attacks. Before we discuss the modification procedure steps in detail, it is necessary to introduce the format of the configuration file.

I3SOAPISAPIConfig.xml Filter File Format

The ISAPI filter is just an XML file whose structure can be described as follows. Its root element, <FilterConfig> has three child elements, <ICServers>, <Defaults> and <Rules>.

<ICServers>

The <ICServers> element contains a list of Interaction Center Servers to which to route the messages. <ICServers> can have <ICServer> and <ICServer2> child elements.

<ICServer2> Uses a remote subsystem connection. GenSSLCerts must be run prior to attempting to connect to a notifier with this type of connection. In a switchover situation, use <ICServer>. <ICServer2> will not work correctly in switchover environments.

The attributes of the <ICServer> child element are:

name

The name of the CIC server, used to identify the server in filter rules.

host

Hostname or IP address of the Notifier (CIC) server.

username

Login name for the Notifier connection.

password

Password for the Notifier session.

The attributes of the <ICServer2> child element are:

name

Name of the server (used to identify it in rule action).

host

Hostname or IP address of the Notifier server.

<Defaults>

The <Defaults> element stipulates default rule actions. It has two child elements. <ForwardRequest> identifies requests that will be forwarded. <HTTPResponse> identifies requests to be rejected.

The attributes of the <ForwardRequest> child element are:

server

Name of the Interaction Center Server configured through the corresponding <ICServer> tag. This attribute is (case-sensitively) matched against the name attributes of the <ICServer> tags.

initiatorEvent

Name of the InitiatorEvent (notification event) as which the request should be forwarded to the Interaction Center server. If not explicitly specified or an empty string, the soapAction from the HTTP header will be used.

soapAction

SOAPAction string to be forwarded to IP. If not defined or "*", use same action that matched the rule.

clientName

Client name value specified in the request notification. Default = "I3SOAPISAPI". This is mainly informational for use as a trace message.

requestTimeout

Timeout value used for the request. Default as specified by ‘DefaultRequestTimeout' registry key. Time in milliseconds

includeTransportInfo

Specifies whether to include the TransportInfo data in the request sent to IP. Possible values: "1", "0", "true", "false". Default = "1".

The attributes of the <HTTPResponse> child element are:

statusCode

HTTP status code. Default = "500".

statusText

HTTP status text. Default = lookup based on statusCode (for "500": "Internal Server Error").

soapFaultcode

Value of the <faultcode> element in the <Fault> element of the response sent back to the client. Default = "Server.SOAPAction".

soapFaultstring

Value of the <faultstring> element in the <Fault> element of the response sent back to the client.. Default = "The SOAPAction is not recognized by the server!"

<Rules>

The <Rules> element contain <Rule> child elements which define the action to be performed when the rule fires. That happens when the request's SOAPAction matches the rule's soapAction attribute. The <Rule>child element has only one attribute:

soapAction

SOAPAction that triggers this rule. SOAPAction matching is case-sensitive.

Sample I3SOAPISAPIConfig File

This sample filter listed below shows how the elements fit together. The numbers are for illustration purposes and do not appear in an actual configuration file. See SOAP ISAPI Filter Schema for the schema used by I3SOAPISAPIConfig.xml.

```
1  <FilterConfig xmlns="urn:schemas-inin-com:soapisapi-filter-config">
2    <ICServers>
3       <ICServer name="localhost"
4          host="localhost"
5          userName=""
6          password=""/>
7       <ICServer name="mars"
8          host="mars"
9          userName="eic_admin"
10         password="i3"/>
11   </ICServers>
12   <Defaults>
```

```
13      <ForwardRequest server="localhost"
14         clientName="I3SOAPISAPI"
15         requestTimeout="20000"
16         includeTransportInfo="1"/>
17         <HTTPResponse statusCode="500"
18            statusText="Internal Server Error"
19            soapFaultcode="Client.SOAPAction"
20            soapFaultstring="The specified method is not supported!"/>
21    </Defaults>
22   <Rules>
23      <Rule soapAction="uri:my-calculator#Add">
24         <ForwardRequest initatorEvent="uri:my-calculator"/>
25      </Rule>
26      <Rule soapAction="uri:my-calculator#Subtract">
27         <ForwardRequest initatorEvent="uri:my-calculator"/>
28       </Rule>
29      <Rule soapAction="uri:my-calculator#Multiply">
30         <ForwardRequest initatorEvent="uri:my-calculator"/>
31      </Rule>
32      <Rule soapAction="uri:my-calculator#Divide">
33         <ForwardRequest initatorEvent="uri:my-calculator"/>
34      </Rule>
35      <Rule soapAction="uri:test#foo">
36         <ForwardRequest server="mars"
37            soapAction="uri:test#bar"
38            requestTimeout="120000"/>
39      </Rule>
40      <Rule>
41         <HTTPResponse/>
42      </Rule>
43   </Rules>
44 </FilterConfig>
```

This sample specifies several SOAPActions that refer to a calculator service. On line 23, the SOAPActions of the calculator are forwarded with the "uri:my-calculator" InitiatorEvent, so all requests trigger the same initiator. All other attributes of that rule are inherited from the default <ForwardRequest> element (line 13). Accordingly, requests for my-calculator are sent to the "localhost" server, even

though that was not explicitly defined in the rule. It is easy to specify attributes in a Rule element that override default elements. In line 35, the SOAPAction "uri:test#foo" is forwarded as "uri:test#bar" (both the SOAPAction and InitiatorEvent) to the server "mars". The request timeout for this particular request is set to 2 minutes (120,000 milliseconds).

The last rule simply rejects all other SOAPActions with the default <HTTPResponse> rule action. To forward all SOAPActions indiscriminately, the following rule could be used:

```
<Rule>
  <ForwardRequest/>
</Rule>
```

Wildcard Pattern Matching

Currently, we do not support regular expression patterns as the soapAction attribute of a rule, although that may be added in a future release. However, to simplify filters for objects with many methods, a simple wildcard pattern is supported: The soapAction value may end with an asterisk (*), which means that the SOAPAction many be followed by one or more characters, that are ignored in the match. The * wildcard is supported only if it is the last character in a soapAction attribute. For example, this technique could be used to replace all rules for the calculator with a single one, where soapAction attribute has a value of " uri:my-calculator#*". Implement wildcards with care, or not at all, since this opens the possibility for DoS attacks on the Notifier event-ID caches. We thus strongly suggest explicitly adding rules for each SOAPAction that is to be forwarded to the server.

Forward only supported SOAPActions to CIC

1. Customize the ISAPI filter file to prevent the SOAP Listener task from indiscriminately forwarding all SOAP requests to the Interaction Center Server. Filtering ensures that the CIC Server receives only those requests that match supported SOAPActions.
2. Set SOAPAction Timeout Values. You can optionally modify this file to assign SOAPAction-specific timeout values, by adding requestTimeout attributes to ForwardRequest elements. The example below shows how to set the timeout value for a SOAPAction named "bar" to 120 seconds.

   <ForwardRequest server="mars" …identifies the CIC server

   soapAction="uri:test#bar" …identifies which SOAPAction

   requestTimeout="120000"/> …action-specific timeout value in milliseconds

3. Unload the SOAP ISAPI DLL. To put a modified filter configuration into effect, you must unload the ISAPI DLL. The DLL will reload automatically the next time that a SOAP request is received.
   1. From the desktop of your IIS server, press the *Start* button. Select *Settings*, then *Control Panel*.
   2. Double-click the *Administrative Tools* folder to open it.
   3. Double-click the icon titled *Internet Services Manager*.

4. Right-click the name of your virtual directory. Then select *Properties*.
5. Select the *Virtual Directory* tab. Then press the *Unload* button.
6. Press OK to close the active dialog.
7. Close the *Internet Services Manager* window. Changes made to the SOAP filter configuration will take effect the next time that a request is received.

Configuring IC SOAP Listener to work with IC 4.0 and 2015 or later

The IC SOAP Listener install does not properly handle certificate mappings necessary for connecting to the IC 4.0 and 2015+ Notifier. This article describes the steps required to enable IC SOAP Listener to function in an CIC 4.0 or COC 2015+ environment when installed on a separate web server.

Perform these steps after the CIC SOAP Listener component is installed on the web server and patched to the latest SU level.

Update the IC User Configuration

1. Navigate to `C:\Program Files (x86)\Interactive Intelligence\SOAPListener\Filter` directory.
2. Edit the `I3SOAPISAPIConfig.xml` file.
3. Update tag values, where:
    - `ICSERVER` is the name of the CIC server to which a connection should be made
    - `ICUSERNAME` is the name of a valid user account on the CIC server
    - `ICPASSWORD` is the password for CC user
4. Save the file.

Update the Registry

1. Open the registry by navigating to **Start** | **Run** | **regedit**.
2. Select the `HKEY_LOCAL_MACHINE\SOFTWARE\WOW6432Node\Interactive Intelligence\Certificates` registry key. If this key does not exist create it.
3. Under this key there should be an entry named `Path` with type `REG_SZ` and a data value of `C:\Program Files (x86)\Interactive Intelligence\Certificates`. If this key does not exist create it.
4. Right-click the `HKEY_LOCAL_MACHINE\SOFTWARE\WOW6432Node\Interactive Intelligence\Certificates` key. Then select **Permissions…**.
5. Grant the `NETWORK` user `Full Control` access over this key.

Use the `NETWORK` user not the `NETWORK SERVICE` user.

Update Environment Variables

1. Right-click `My Computer*` and select `*Properties`.
2. Select `Advanced System Settings`.
    - Click the `Environment Variables` button.
3. Add a new System variable with a `Variable name` of `ININ_Certificates`. Set a `Variable value` of `C:\Program Files (x86)\Interactive`

Intelligence\Certificates\<WEB SERVER NAME>_ININ_Certificates.xml where <WEB SERVER NAME> is the name of the web server the IC SOAP Listener component was installed on.Updating the Certificates Directory:

4. Navigate to C:\Program Files (x86)\Interactive Intelligence.
5. Right-click the Certificates folder and select Properties. If this folder does not exist create it.
6. Select the Security tab
7. Grant the NETWORK user Full Control over this directory

   Use the NETWORK user not the NETWORK SERVICE user.

8. Restart the server.
9. After the restart, use the command gensslcertsu -c <notifier> -f to generate certificates against each IC server the SOAP Listener will connect to.

Update IIS Settings

1. Under ISAPI and CGI Restrictions, add a new entry, specifying the I3SOAPISAPIU.dll, and select "Allow extension path to execute".
2. Under SoapListener web site, select Handler Mappings.
3. Add a Module Mapping.
   - Request path: *.dll
   - Module: IsapiModule
   - Executable: browse to I3SOAPISAPIU.dll
   - Enable Execute under the Access tab in Request Restrictions
4. Select the Application Pools tab, then select the app pool that the default website is configured under, then "Advanced Settings".
5. For SoapListener, set 32-Bit Applications to True.
6. Set Managed Pipeline Mode to Classic.
7. Reboot the server.

Additional steps for switchover pairs

On each CIC server:

1. Open regedit and navigate to HKEY_LOCAL_MACHINE\SOFTWARE\Interactive Intelligence\Certificates. Right click on this registry key, and select Permissions. Click on the **Add** button, and enter NETWORK SERVICE for the user. Give this user Full Control to allow the user permissions to access the registry key.
2. Open Windows Explorer.
3. Navigate to the \I3\IC\Certificates directory.
4. Select the <Machine Name>_ININ_Certificates.xml, <Machine Name>_PrivateKey.bin and <Machine Name>_PublicKey.bin files. Right click and then select Properties.
5. Click on the Security tab. Then click **Add**. Enter NETWORK SERVICE for the user, and give Full Control to this user to allow proper file access to these files.

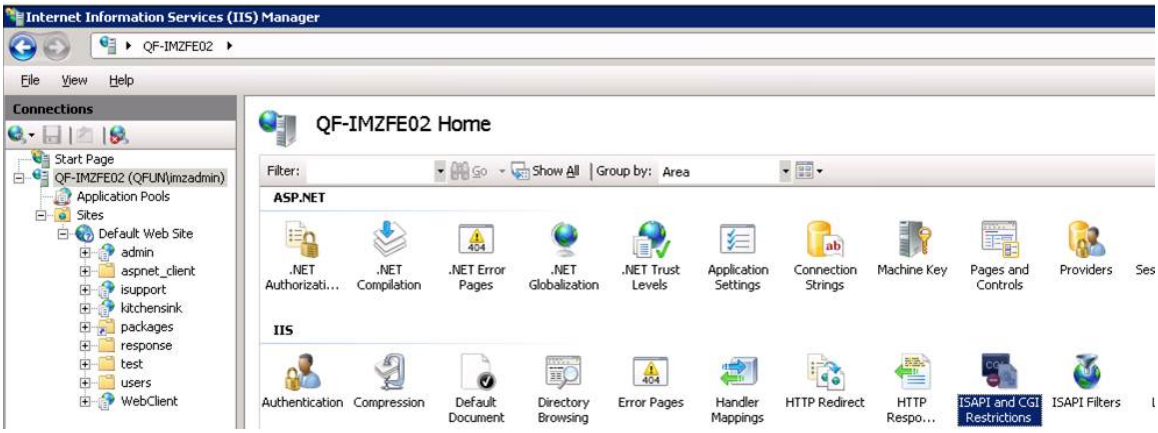
6. Do the same for the `\I3\IC\Certificates\Client\Remote_Client\<Notifier Name>\<Notifier Name>_TrustedCertificate.cer` file.

Additional configuration steps required for SOAP Listener when using IIS7

Several settings need to be configured in IIS7 for SOAP functionality in a CIC 4.0 (or later) environment. As a supplement to the installation document, the following article covers the extra steps that need to be made in IIS7 before SOAP Listener is fully operational.

When utilizing SOAP functionality in CIC 4.0 with IIS7, it is important to note that SOAP Listener is an ISAPI extension. To enable an ISAPI extension, make the following changes in IIS7:

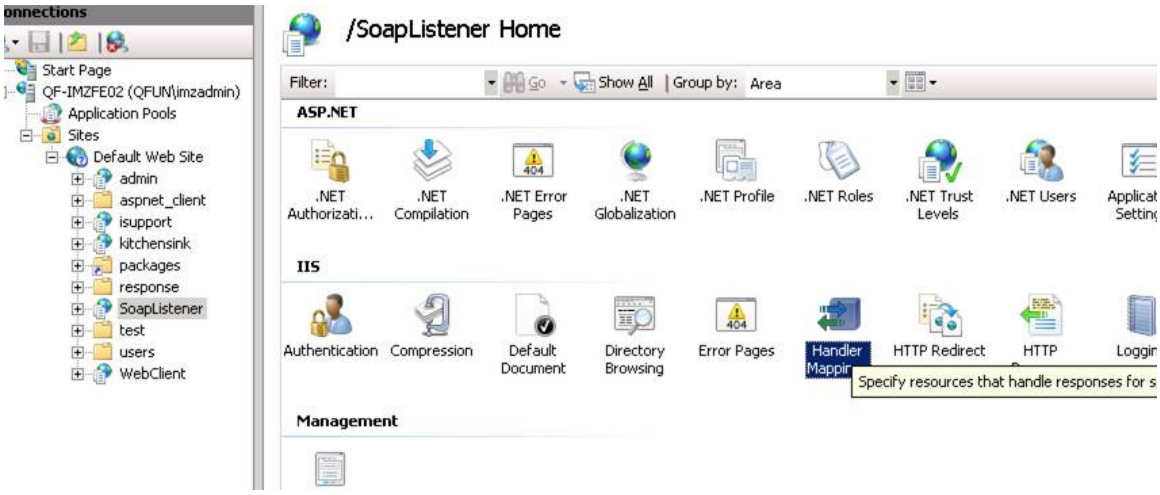
1. Add an exception to "ISAPI and CGI Restrictions" so that the "I3SOAPISAPIU.dll" is allowed to execute. This can be found at the server level in IIS Manager.



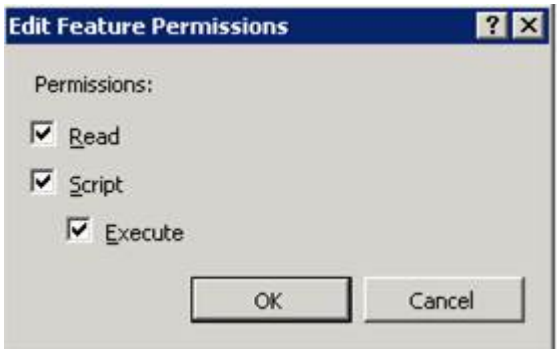
2. After selecting restrictions, choose "Add..." from the right-hand side of IIS Manager and input the filepath and description for SOAP Listener.

3. Next, enable "ISAPI-dll" handler mapping. Genesys recommends doing this at the "SoapListener" application level, but it can be accomplished at a site or server level as well depending on the desired inheritance model. After selecting SOAP Listener from the "Default Web Site" hierarchy,

highlight the "ISAPI-dll" handler mapping. Then select "Edit Feature Permissions..." from the right-hand side of the screen.



4. Check "Execute" in the popup window. Then click OK.



At this point, SOAP Listener is configured to work with IIS7 in a CIC 4.0 or later environment.

SOAP ISAPI Filter Schema

The ISAPI Filter Configuration file conforms to the following schema:

```
<?xml version="1.0"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
   xmlns:tns="urn:schemas-inin-com:soapisapi-filter-config"
   targetNamespace="urn:schemas-inin-com:soapisapi-filter-config"
   elementFormDefault="qualified"
   attributeFormDefault="unqualified">
   <xsd:element name="FilterConfig" type="tns:FilterConfig"/>
```

```
   <xsd:complexType name="FilterConfig">
     <xsd:sequence>
        <xsd:element name="ICServers" type="tns:ICServers" minOccurs="0"/>
        <xsd:element name="Defaults" type="tns:Defaults" minOccurs="0"/>
        <xsd:element name="Rules" type="tns:Rules" minOccurs="0"/>
     </xsd:sequence>
   </xsd:complexType>
   <xsd:complexType name="ICServers">
     <xsd:element name="ICServer" type="tns:ICServer" minOccurs="0"
        maxOccurs="unbounded"/>
   </xsd:complexType>
   <xsd:complexType name="ICServer">
     <xsd:attribute name="name" type="xsd:string" use="required"/>
     <xsd:attribute name="host" type="xsd:string" use="required"/>
     <xsd:attribute name="userName" type="xsd:int" use="optional"/>
     <xsd:attribute name="password" type="xsd:boolean" use="optional"/>
   </xsd:complexType>
   <xsd:complexType name="Defaults">
     <xsd:sequence>
        <xsd:element name="ForwardRequest" type="tns:ForwardRequest"
minOccurs="0"/>
        <xsd:element name="HTTPResponse" type="tns:HTTPResponse"
minOccurs="0"/>
     </xsd:sequence>
   </xsd:complexType>
   <xsd:complexType name="Rules">
     <xsd:element name="Rule" type="tns:Rule" minOccurs="0"
maxOccurs="unbounded"/>
   </xsd:complexType>
   <xsd:complexType name="Rule">
     <xsd:choice minOccurs="0">
        <xsd:element name="ForwardRequest" type="tns:ForwardRequest"/>
        <xsd:element name="HTTPResponse" type="tns:HTTPResponse"/>
     </xsd:choice>
     <xsd:attribute name="soapAction" type="xsd:string" use="optional"/>
   </xsd:complexType>
```

```
   <xsd:complexType name="ForwardRequest">
     <xsd:attribute name="server" type="xsd:string" use="optional"/>
     <xsd:attribute name="initatorEvent" type="xsd:string" use="optional"/>
     <xsd:attribute name="soapAction" type="xsd:string" use="optional"/>
     <xsd:attribute name="clientName" type="xsd:string" use="optional"/>
     <xsd:attribute name="requestTimeout" type="xsd:int" use="optional"/>
     <xsd:attribute name="includeTransportInfo" type="xsd:boolean"
use="optional"/>
   </xsd:complexType>
   <xsd:complexType name="HTTPResponse">
     <xsd:attribute name="statusCode" type="xsd:positiveInteger"
use="optional"/>
     <xsd:attribute name="statusText" type="xsd:string" use="optional"/>
     <xsd:attribute name="soapFaultcode" type="xsd:QName" use="optional"/>
     <xsd:attribute name="soapFaultstring" type="xsd:string" use="optional"/>
   </xsd:complexType></xsd:schema>
```

Reinstall/Uninstall SOAP Listener

If you run the *SOAP Listener Install* a second time, it provides the opportunity to change the way features are installed, repair installation errors, or remove SOAP Listener from your computer.

1. Insert your CIC installation DVD (or mount an ISO image). In many cases, the user interface application will start automatically. If it does not appear, run **autorun.exe** from the root directory.
2. Click the **Optional Installs (2)** button.
3. Click **CIC SOAP Listener**.
4. Click **Next** to dismiss the Welcome screen.
5. Click **Change**, **Repair**, or **Remove**.

Install SOAP Notifier COM

Install SOAP Notifier COM

SOAP Notifier COM objects issue SOAP notifications from automation compatible applications. SOAP Notifier COM components provide a high-performance method of initiating handlers without incurring the performance penalty of HTTP-based Listener operations.

Third-party applications created using the *SOAP Notifier COM API* directly create and forward packets to Interaction Processor, bypassing the need to create packets received using HTTP and the Soap Listener task.

What: Run **CC *SOAP Notifier COM Components Setup*** to install and register components needed to run or develop third-party SOAPNotifierCOM applications on a desktop PC. Components are installed to the destination folder specified by the user. The default folder is c:\Program Files\Interactive Intelligence. Setup registers two dynamic link libraries: SOAPNotifierCOMU.DLL and MSSOAPNotifierConnectorU.DLL. Setup optionally installs a help system that describes interfaces, methods, and properties in the Notifier COM API. When this option is selected, setup adds a shortcut named *SOAP Notifier COM Help* to the start menu, inside the *Interactive Intelligence* folder.

Where: Install these components on any PC used to develop or run SOAP Notifier applications.

Prerequisite: The desktop PC must be running a version of Windows that supports the Component Object Model. SOAP Notifier COM API is not compatible with operating systems that do not support COM (Linux, for example).

Steps to Complete

1. Download the CIC 2018 R1 or later .iso file from the Genesys Product Information site at https://my.inin.com/products/Pages/Downloads.aspx.
2. Copy the .iso file to a file server (non-CIC server) with a high bandwidth connection to the server(s) on which you will be running the CIC 2018 R1 or later installs.
3. Mount the .iso file and share the contents to make them accessible to the server(s) on which you will be running the CIC 2018 R1 or later installs.
4. Navigate to the \Installs\Off-ServerComponents directory on the file server.
5. Copy the SOAP Notifier COM .msi file, for example, SOAPCOM_2018_R1.msi, to the server on which you plan to run this install and double-click to launch it.
6. If prompted whether to run the install program, respond **Run**.
7. Press **Next** to dismiss the welcome screen.
8. Press **Next** to accept all default features.
9. Press *Install* to begin installation.
10. Wait while files are copied.
11. Press **Finish** to exit Setup.

Reinstall/Uninstall SOAP Notifier COM Components

If you run *SOAP Notifier COM Components Setup* a second time, it provides the opportunity to modify the way features in installed, to repair installation errors, or to remove SOAP Notifier COM components from your computer.

1. Click **Next** to proceed past the startup screen.
2. Then select **Change**, **Repair**, or **Remove**.

Appendix A: SOAP Transport Information and Control

Appendix A: 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>
```

HTTP Transport

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>
```

Request (Transport Info)

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

This sample Transport Info structure adheres to 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)

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>
```

Tip—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").

Appendix B: SOAP Tools

Appendix B: SOAP Tools

This appendix provides information about the tools in Interaction Designer that process SOAP requests and responses. SOAP tools are late-bound, meaning that the structure of data processed by a SOAP handler does not have to be specified at compile time when the handler is published. SOAP tool steps

can be added to any handler, to create and send SOAP requests to any server that understands SOAP. SOAP Tools do not support calls to an SSL server. In CIC 2.3 and later, the assumed namespace prefix is SOAP, rather than SOAP-ENV, for compatibility with Microsoft .NET. These tools are also documented in the Interaction Designer help.

There are 5 categories of SOAP tools:

- Initiator Tools
- Request Tools
- Payload Processing Tools
- Invocation Tools
- Helper Tools

Initiator Tools

Initiator Tools

SOAP Initiator

This initiator triggers if the 'Notification Event' of the request matches a specified string. The Notification Event on which the Initiator triggers is specified in the property dialog.

Parameter	Dir	Type	Remarks
SOAP Request	OUT	Handle	Handle representing the SOAP request. It can subsequently be used to query additional information from the (HTTP) header.
Initiator Event	OUT	String	String of the notification event that triggered the initiator.
SOAP Action	OUT	String	SOAP Action of the request that triggered the handler.

Request Tools

Request Tools

SOAP Get Request Info

Queries some information from the request handle. Exit Paths: Success, Failure.

Parameter	Dir	Type	Remarks
SOAP Request	IN	Handle	Handle of the SOAP request.
Initiator Event	OUT	String	Notification event that caused the initiator to trigger.
SOAP Action	OUT	String	SOAP action code of that request.
Client ID	OUT	Integer	Client ID (Notifier object id).
Client Name	OUT	String	Name of the client.
Request ID	OUT	Integer	Request ID (for debugging/tracing purposes).

Payload Size	OUT	Integer	Size of the request payload in bytes.
Transport Info Size	OUT	Integer	Size of the transport information in bytes.

SOAP Abort Request

Aborts the request. If 'Send Unhandled Response' is False, it does not send a response notification, not even an "Unhandled" response when the Request handle goes out of scope. Aborting a request is useful if a SOAP request handler is registered as Monitor handler, for example for wildcard SOAPAction. Multiple handlers may fire at the same time, but only one must send a response notification to the client.

Exit Paths: Success, Failure

Parameter	Dir	Type	Remarks
SOAP Request	IN	Handle	Handle of the SOAP request
Send Unhandled Response	IN	Boolean	Checkbox (default = False)

SOAP Get Transport Info

Returns an XML document containing transport specific (header) data. It allows the client to include any kind of out-of-band data in the request. For example, for HTTP requests, this document contains the HTTP method and a list of the header elements.

Tip—The data may be parsed every time the tool is invoked or cached. This may depend on the specified selection namespaces. The returned document is read-only.

See SOAP ISAPI Filter Schema for schema details. If there is no transport information data, an empty document is returned and the tool takes the 'No Info' exit. If there is an error (Failure), an empty document is returned which can be queried with 'XML Get Error Info'.

Exit Paths: Success, No Info, and Failure

Parameter	Dir	Type	Remarks
SOAP Request	IN	Handle	Handle of the SOAP request
Selection Namespaces	IN	String	Optional. Space delimited list of namespace declarations to be set as selection namespaces for the XPath queries.
Preserve Whitespace	IN	Boolean	Checkbox: False Default. Nonessential white space is ignored when parsing the payload. True Preserve nonessential white space.

Validate On Parse	IN	Boolean	Checkbox: False Default. Only verifies for well-formedness. True Validates against the schema during parse.
Resolve Externals	IN	Boolean	Checkbox: False Default. Do not resolve resolvable namespaces. True Resolve resolvable externals (namespaces, DTDs, entity references etc.) at parse time.
Transport Info	OUT	Node	Read-only. XML document containing transport-specific out-of-band information. Empty document if no transport information. See Appendix A: SOAP Transport Information and Control.

SOAP Expects Response

Takes a different exit path depending on whether the SOAP request requires a response (YES) or not (NO). If the request expects a response and the handler exits (the SOAP Request handle goes out of scope) without having invoked 'SOAP Send Response', a Response Notification is sent back with the 'Unhandled' flag set to true.

Exit Paths: YES, NO

Parameter	Dir	Type	Remarks
SOAP Request	IN	Handle	Handle of the SOAP request

SOAP Parse Request Payload

Parses the payload of the request into an XML document. If the 'Validate SOAPAction' parameter is True, the tool checks the SOAPAction field of the request against the payload. The payload data is parsed every time this tool is invoked (i.e. it is not cached). The document is furthermore *not* read-only and thus may be modified as needed, for example to create the response. The payload envelope node will still be returned, even if the SOAP Action does not match.

Heuristic

This tool uses a heuristic to match the action code (legend: *<NS>* = namespace of the first body element; *<MethodName>* = Name of the element [method name]):

<NS>

<NS> [**<AnyCharacter>**] **<MethodName>**

<MethodName>

[**<AnyCharacter>**] **<MethodName>**

This will catch actions such as "uri:my-uri#MyMethod", "http://soap.inin.com/e-faq", "MyMethod" etc. An empty SOAPAction matches all methods.

Parameter	Dir	Type	Remarks
SOAP Request	IN	Handle	Handle of the SOAP request
Validate SOAPAction	IN	Boolean	Checkbox: False Don't verify SOAPAction header field against payload. True Default. Check SOAPAction header field against method namespace and name.
Action Validation Mask	IN	String	Optional. Mask for validation of SOAP Action. Note: this is a future extension that has not been defined.
Selection Namespaces	IN	String	Optional. Space delimited list of namespace declarations to be set as selection namespaces the XPath queries. If this argument not specified, just "SOAP-ENV" is mapped to the envelope namespace. NOTE: The "SOAP-ENV" prefix will be used irrespective of the actual prefix in the payload. A declaration mapping "SOAP-ENV" to the envelope namespace will always be added to the declarations, unless SOAP-ENV is already declared in the argument.
Preserve Whitespace	IN	Boolean	Checkbox: False Default. Nonessential whitespace is ignored when parsing the payload. True Preserve nonessential white space
Validate On Parse	IN	Boolean	Checkbox: False Default. Only verifies for well-formedness. True Validates against the schema during parse.
Resolve Externals	IN	Boolean	Checkbox:

			False Default. Do not resolve resolvable namespaces. True Resolve resolvable externals (namespaces, DTDs, entity references etc.) at parse time.
Payload	OUT	Node	XML document with Envelope as document element. If there is an error, the document may be empty (but not NULL), and the 'XML Get Error Info' tool can be used to retrieve information about what failed).

Exit Paths

Success

Payload successfully parsed. SOAP Action matches.

Empty Payload

SOAP Payload is empty (XML document has no document element).

Wrong Action

SOAP Action validation enabled and action doesn't match.

Parse Error

A parse error occurred parsing the payload. Use 'XML Get Error Info'.

Failure

Some other failure. Use 'XML Get Error Info'.

SOAP Send Response

Sends the specified payload as response to the sender of the request. To support transport specific features, the 'Transport Control Data' argument takes an XML node whose content will be sent back to the client. It can be used to send transport specific out-of-band data to the client. For example, for the HTTP transport it allows to set additional header fields or specify a special status code. See SOAP ISAPI Filter Schema for schema details. The schema itself is not part of SOAP specification.

Parameter	Dir	Type	Remarks
SOAP Request	IN	Handle	Handle of the SOAP request
Payload	IN	Node	Node of the payload envelope to send back to the client. Must be document node or <Envelope> document element.
Transport Control Data	IN	Node	Optional. Node of an XML structure with additional transport specific control data. See Appendix A: SOAP Transport Information and Control.

Exit Paths

Success

Response was sent successfully.

No Response

This request does not expect a response.

Duplicate

Response for this request has already been sent.

Failure

Some other error. Check Payload node with XML Get Error Info.

Payload Processing Tools

Payload Processing Tools

SOAP Create Envelope

Creates a new SOAP envelope. To simplify composing RPC requests, where the first child element of the <Body> element is the method to invoke, the 'RPC Method Name' and 'RPC Method Namespace' argument can be used as shortcut. The same can be achieved by invoking 'SOAP Add Body Element' after creating the envelope. Therefore, this tool creates the following XML document:

```
<?xml version="1.0" encoding="{XML Encoding}" ?></{RPC Method Name}>]
   </{Envelope Prefix}:Body>
</{Envelope Prefix}:Envelope>
```

The 'Declare Namespaces' argument is used to declare namespaces in the envelope that will be used in other elements, such as the **xsd** or **xsi** prefixes for typed arguments. It keeps the size of the envelope low, as otherwise each element that uses a prefix will contain **xmlns** attributes. If the 'RPC Method Name' argument has no namespace prefix and an 'RPC Method Namespace' different than **""** (default namespace) is specified, a prefix will be synthesized, unless the local name starts with a '**:**' (which is illegal in XML, but signals to this tool *not* to add a synthesized namespace prefix). Adding a prefix can greatly reduce the size of the message if child elements are in no namespace (usually parameters are in the default namespace), as otherwise each child element would get a **xmlns=""** attribute.

Exit Paths: Success, Failure

Parameter	Dir.	Type	Remarks
XML Encoding	IN	String	Optional. Character encoding to be used for the XML document. If omitted, "UTF-8" is used. See remarks.
Envelope	IN	String	Optional. Namespace prefix for the envelope namespace. If not

Prefix			specified the default "SOAP-ENV" is used.
Encoding Style	IN	String	Optional. Space separated list of namespaces specifying the encoding style (value of the 'encodingStyle' attribute). If not specified or "STANDARD" is passed as string, "http://schemas.xmlsoap.org/soap/encoding/" is used. The encodingStyle attribute is omitted if "NONE" is specified.
RPC Method Name	IN	String	Optional. Fully qualified name of the method element (first child element of the body element). If not specified, no method element will be added. Please consult Remarks for additional details!
RPC Method Namespace	IN	String	Optional. Namespace of the method element.
Declare Namespaces	IN	String	Space delimited list of namespace declarations of the form xmlns:{prefix}='{URI}' to be declared in the envelope. See remarks.
Selection Namespaces	IN	String	Optional. Space delimited list of namespace declarations to be set as selection namespaces for the XPath queries. If argument not specified, the envelope prefix and the 'Declare Namespace' namespaces will be set as selection namespaces. NOTE: mapping for envelope prefix will always be added.
Envelope	OUT	Node	XML document with Envelope as document element.

SOAP Get Body

Retrieves the Body element from the SOAP envelope. A body must exist and if it can't be found, the tool exits through 'Failure' and attaches error information to the envelope.

Exit Paths: Success, Failure

Parameter	Dir	Type	Remarks
Envelope	IN	Node	Envelope node of the SOAP payload. Can be a document node whose document element is <SOAP-ENV:Envelope> or the node is the element itself.
Body	OUT	Node	Node of the <SOAP-ENV:Body> element.

SOAP Get Body Element

Retrieves the first body element that matches the given base name and namespace. If no namespace is specified, the first element matching ‘Base Name' is returned. Returns the first element in the body if neither a name nor namespace is given.

Exit Paths: Success, Not Found, Failure

Parameter	Dir	Type	Remarks
Envelope	IN	Node	Envelope node of the SOAP payload. Can be a document node whose document element is <SOAP-ENV:Envelope> or the node is the element itself.
Base Name	IN	String	Optional. Base name of the element to return. If no name given, the first entry in the body in ‘Namespace' is returned. This corresponds to the element of the method for RPC requests.
Namespace	IN	String	Optional. Namespace of the element to return
Retrieve Value	IN	Boolean	Checkbox: False Default. Do not retrieve value True Return node value
Body Element	OUT	Node	Child element of the <SOAP-ENV:Body> element that has the given base name and namespace. NULL node if the element is not in the body.
Element Base Name	OUT	String	Base name of the returned element
Element Namespace	OUT	String	Namespace URI of the returned element
Value	OUT	String	Value of the body element (if ‘Retrieve Value' = True)

SOAP Add Body Element

Adds an entry to the body of the SOAP envelope. Use the XML tools on the returned ‘Element' node to add rich contents to the element (not just a string).

Tip—If the ‘Name' argument has no namespace prefix and a ‘Namespace' different than **""** (default namespace) is specified, a prefix will be synthesized, unless the local name starts with a ‘**:**' (which is illegal in XML, and thus signals to this tool *not* to add a synthesized namespace prefix). Adding a prefix can greatly reduce the size of the message if child elements are in no namespace, as otherwise each child element would get an **xmlns=""** attribute.

Exit Paths: Success, Failure

Parameter	Dir	Type	Remarks
Envelope	IN	Node	Envelope node of the SOAP payload. Can be a document node whose document element is <SOAP-ENV:Envelope> or the node is the element itself.
Name	IN	String	Fully qualified name of the element to create and add to the body.
Namespace	IN	String	Optional. Namespace URI of the element. If the parameter is omitted and the name has a namespace prefix, the tool will search in the parent elements for the namespace with the same prefix and make the element a member of this namespace.
Encoding Style	IN	String	Optional. Value of the 'encodingStyle' attribute. Attribute is omitted if not specified or "NONE". Specify "STANDARD" for standard namespace ("http://schemas.xmlsoap.org/soap/encoding/").
Value	IN	String	Optional. String value to set as content of the element.
Replace Existing Body Element	IN	Boolean	Checkbox: False Default. Add the element as last child of the body. True Replace first element in the body that has the same (local) name and namespace. If body contains multiple elements with the same name and namespace, the remaining ones are not modified.
Delete All Existing Body Elements	IN	Boolean	Checkbox: False Default. Append to the child list of the body. True Remove all existing elements from the body prior to adding the new element.
Body Element	OUT	Node	Node of the element that has just been added.

SOAP Query Encoding Style

Matches a space separated list of URIs against the 'encodingStyle' attribute of the element. If the element doesn't have an 'encodingStyle' attribute, the parent of the element is checked until an element with an 'encodingStyle' attribute is found. If that attribute contains any of the specified encoding style URIs, the tool returns through 'Found' and returns the style that was found.

Tip: If the first 'encodingStyle' attribute found along the parent chain does not contain any of the specified styles, the search does not continue and the tool exits 'Not Found'.

Exit Paths: Found, Not Found, Failure

Parameter	Dir	Type	Remarks
Element	IN	Node	(child) Element of the SOAP envelope to query. If document node, the document element is queried.
Encoding Styles	IN	String	Space separated list of URIs to match against the ‘encodingStyle' attributes.
First Style Found	OUT	String	Encoding style namespace that was found
Element Of Style	OUT	Node	XML node of the element in which the encoding style attribute was found.

SOAP Get Header

Retrieves the header element from the SOAP envelope if it has one.

Exit Paths: Success, No Header, Failure

Parameter	Dir	Type	Remarks
Envelope	IN	Node	Envelope node of the SOAP payload. Can be a document node whose document element is <SOAP-ENV:Envelope> or the node is the element itself.
Header	OUT	Node	Node of the <SOAP-ENV:Header> element. NULL node if the envelope contains no header.

SOAP Get Header Element

Retrieves the first header element that matches the given base name and namespace. Returns the first element in the header if neither a name nor namespace is given. Takes ‘Not Found' exit if the envelope doesn't have a header or the element can't be found.

Exit Paths: Success, Not Found, No Header, Failure

Parameter	Dir	Type	Remarks
Envelope	IN	Node	Envelope node of the SOAP payload. Can be a document node whose document element is <SOAP-ENV:Envelope> or the node is the element itself.
Base Name	IN	String	Optional. Base Name of the element to return
Namespace	IN	String	Optional. Namespace of the entry to return
Retrieve Value	IN	Boolean	Checkbox: False Do not retrieve value True Default. Return node value

Header Element	OUT	Node	Child element of the <SOAP-ENV:Header> element that has the given base name and namespace. NULL node if the envelope contains no header or the element is not in the header.
Element Base Name	OUT	String	Base name of the returned element
Element Namespace	OUT	String	Namespace URI of the returned element
Value	OUT	String	Value of the element (if 'Retrieve Value' = True)

SOAP Get Header Elements

Returns iterator to a list of header elements filtered by the given arguments. Takes the 'None' exit if envelope has no header or none of the header elements matched the filter criteria.

Exit Paths: Success, None, No Header, Failure

Parameter	Dir	Type	Remarks
Envelope	IN	Node	Envelope node of the SOAP payload. Can be a document node whose document element is <SOAP-ENV:Envelope> or the node is the element itself.
Base Name	IN	String	Optional. Only include elements with this base name.
Namespace	IN	String	Optional. Only include elements in this namespace.
Must Understand	IN	Boolean	Optional: False Return header entries whose 'mustUnderstand' attribute is "0" (or no attribute is specified) True Return header entries whose 'mustUnderstand' attribute is "1". Default: Don't filter on 'mustUnderstand'
Actor URIs	IN	String	Optional. Space separated list of actor URIs. Only elements whose actor attribute has one of these namespaces is returned. If not specified, don't filter on actor namespace.
Header Elements	OUT	NodeIter	Iterator to collection of header entries. Use the 'XML Get Next Node' tool to iterate over collection.
Count	OUT	Integer	Number of items in the Header Entries collection

SOAP Add Header Element

Creates a header element and adds it to the given envelope. If the envelope doesn't yet have a header, one will be inserted before the Body element.

If the ‘Name' argument has no namespace prefix and a ‘Namespace' different than **""** (default namespace) is specified, a prefix will be synthesized, unless the local name starts with a ‘**:**' (which is illegal in XML, and thus signals to this tool *not* to add a synthesized namespace prefix). Adding a prefix can greatly reduce the size of the message if child elements are in no namespace, as otherwise each child element would get an **xmlns=""** attribute.

Exit Paths: Success, Failure

Parameter	Dir	Type	Remarks
Envelope	IN	Node	Envelope node of the SOAP payload. Can be a document node whose document element is <SOAP-ENV:Envelope> or the node is the element itself.
Name	IN	String	Fully qualified name of the header element to create and add to the header.
Namespace	IN	String	Optional. Namespace URI of the element. If the parameter is omitted and the name has a namespace prefix, the tool will search in the parent elements for the namespace with the same prefix and make the element a member of this namespace.
Must Understand	IN	Boolean	Optional. Specifies the value of the ‘mustUnderstand' attribute: False mustUnderstand="0" True mustUnderstand="1" Not specified: No ‘mustUnderstand' attribute is added.
Actor URI	IN	String	Optional. Value of the ‘actor' attribute.
Encoding Style	IN	String	Optional. Value of the ‘encodingStyle' attribute. Attribute is omitted if not specified or "NONE". Specify "STANDARD" for standard namespace ("http://schemas.xmlsoap.org/soap/encoding/").
Value	IN	String	Optional. String value to set as content of the element.
Replace Existing Header Element	IN	Boolean	Checkbox: False Default. Add the element as last child of the body. True Replace first element in the body that has the same (local) name and namespace. If body contains multiple elements with the same name and namespace, the remaining ones are not modified.
Delete All Existing Header Elements	IN	Boolean	Checkbox: False Default. Append to the child list of the body. True Remove all existing elements from the body prior to

			adding the new element.
Header Element	OUT	Node	Node of the element that just has been inserted.

SOAP Get Fault

Retrieves fault information from the SOAP envelope. If there is no <Fault> element in the envelope, the 'No Fault' exit is taken and NULL elements and empty strings are returned. If the envelope is read-only, the returned elements will be read-only too.

Exit Paths: Success, No Fault, Failure

Parameter	Dir	Type	Remarks
Envelope	IN	Node	Envelope node of the SOAP payload. Can be a document node whose document element is <SOAP-ENV:Envelope> or the node is the element itself.
Fault Element	OUT	Node	Node of the <Fault> element.
Fault Code	OUT	String	Value of the <faultcode> element. It provides programmatic information about the fault.
Fault String	OUT	String	Value of the <fautstring> element. It provides human readable information about the fault.
Fault Actor	OUT	String	Value of the <faultactor> element. It provides the URI of the source of the fault.
Detail Element	OUT	Node	Node of the <detail> element. It is used to transfer application specific fault information. NULL Node if there is no <detail> element.

SOAP Set Fault

Adds a <Fault> element to the envelope or replaces an existing one. If one of the mandatory fields (Fault Code, Fault Actor) is empty, the Failure path is taken and XML Get Error Info may be used on the Envelope node to query for error reasons. If the envelope already has a <Fault> element, the tool will remove the existing <Fault> element and replace it with the new element.

Parameter	Dir	Type	Remarks
Envelope	IN	Node	Envelope node of the SOAP payload. Can be a document node whose document element is <SOAP-ENV:Envelope> or the node is the element itself.
Fault Code	IN	String	String to set as value of the <faultcode> element. String must

			not be empty.
Fault String	IN	String	String to set as value of the <fautstring> element. Should be set to provide human readable information.
Fault Actor	IN	String	Optional. String to set as value of the <faultactor> element. If argument is not specified, no <faultactor> element is added.
Create Detail Element	IN	Boolean	Checkbox: *False* Don't create a <detail> element *True* Default. Create an empty <detail> element NOTE: According to the SOAP spec, a <detail> element must be present if the fault is because the <Body> could not be processed successfully.
Preserve Body Elements	IN	Boolean	Checkbox: *False* Default. Remove all existing body elements and replace with <Fault> element *True* Leave existing body elements and append <Fault> element as last child of <Body> NOTE: When sending a fault response to the client, only the <Fault> element is allowed in the body!
Detail Element	OUT	Node	Returns the node of the newly created <detail> element. If 'Create Detail Element' is False, a NULL node is returned.

Exit Paths: Success, Failure

SOAP Create Fault Response

Copies the request envelope and replaces all children of the <Body> element with a single <Fault> element. It thus combines the 'SOAP Create Envelope' and 'SOAP Set Fault' tools. The selection namespaces from the source envelope document are copied to the response envelope document as well.

Exit Paths: Success, Failure

Parameter	Dir	Type	Remarks
Envelope	IN	Node	Envelope node of the request SOAP payload. Can be a document node whose document element is <SOAP-ENV:Envelope> or the node is the element itself.
Fault Code	IN	String	String to set as value of the <faultcode> element. String must not be empty.
Fault String	IN	String	String to set as value of the <fautstring> element. Should be

			set to provide human readable information.
Fault Actor	IN	String	Optional. String to set as value of the <faultactor> element. If argument is not specified, no <faultactor> element is added.
Create Detail Element	IN	Boolean	Checkbox: *False* Don't create a <detail> element *True* Default. Create an empty <detail> element NOTE: According to the SOAP spec, a <detail> element must be present if the fault is because the <Body> could not be processed successfully.
Copy Header	IN	Boolean	Checkbox: *False* Does not copy the <Header> element from the source envelope. *True* Copies the <Header> element and its content from the source envelope.
Response Envelope	OUT	Node	Document node of the response envelope
Detail Element	OUT	Node	Node of the <detail> element of the <Fault> element. If 'Create Detail Element' is False, a NULL node is returned.

SOAP Get RPC Parameter

This is a convenience tool for examining RPC requests. It retrieves a parameter element (child) from the first element in the <Body> element (method in an RPC request). It returns the first element that matches all of the specified arguments. If 'Base Name', 'Namespace', and 'Index' are undefined, the first element will be returned.

For example, to retrieve the 2nd parameter from the 'Add' method in the calculator example presented in Listing 4, you would specify "Parameter2" as name and "" as namespace, or '1' as index.

Exit Paths: Success, Not Found, Failure

Parameter	**Dir**	**Type**	**Remarks**
Envelope	IN	Node	Envelope node of the SOAP payload. Can be a document node whose document element is <SOAP-ENV:Envelope> or the node is the element itself.
Base Name	IN	String	Optional. Base name of the parameter
Namespace	IN	String	Optional. Namespace of the parameter
Index	IN	Integer	Optional. Zero based index into parameters of the method. If

			this parameter is specified, ‘Name' and ‘Namespace' may be omitted, but if present must match the name and namespace of the parameter.
Retrieve Value	IN	Boolean	Checkbox: *False* Do not retrieve value *True* Default. Return node value Disable retrieval of value if parameter contains a large XML document and the value is not used (performance option).
Parameter Element	OUT	Node	Parameter element
Parameter Base Name	OUT	String	Base name of the parameter element
Parameter Namespace	OUT	String	Namespace URI of the parameter element
Parameter Index	OUT	Integer	Zero based index of the parameter element in the child list of the method element.
Value	OUT	String	Value of the parameter

SOAP Add RPC Parameter

This is a convenience tool for composing RPC requests or responses. It adds a parameter element to the first element in the body of the envelope, which represents the method in RPC requests. Use the XML tools to add complex data (not just a string) to the parameter by manipulating the returned ‘Parameter Element' node.

The <Body> element must have a child element (method element). Otherwise this tool fails. When using ‘SOAP Create Envelope', you must add a method element using ‘SOAP Add Body Element'. The ‘SOAP Create RPC Response' tool already adds a method element.

Exit Paths: Success, Failure

Parameter	**Dir**	**Type**	**Remarks**
Envelope	IN	Node	Envelope node of the SOAP payload. Can be a document node whose document element is <SOAP-ENV:Envelope> or the node is the element itself.
Name	IN	String	Qualified name of the parameter
Namespace	IN	String	Optional. Namespace URI of the element. If the parameter is omitted and the name has a namespace prefix, the tool will search in the parent elements for the namespace with the same prefix and make the element a member of that namespace.
Value	IN	String	Optional. Value of the parameter
Parameter	OUT	Node	Node of the element that just has been added to the method

Element			element.

SOAP Get RPC Method Info

This is a convenience tool for examining RPC requests. It retrieves the first child element of the SOAP <Body> element (Method element in RPC requests). It also returns a collection containing the child elements of the method, which constitute the method arguments. The tool exits through 'No Method' if the body does not contain an element. It returns through <Fault> if the body contains a <Fault> element.

Exit Paths: Success, Fault, No Method, Failure

Parameter	Dir	Type	Remarks
Envelope	IN	Node	Envelope node of the SOAP payload. Can be a document node whose document element is <SOAP-ENV:Envelope> or the node is the element itself.
Method Element	OUT	Node	Node of the method element (first child of the Body)
Method Base Name	OUT	String	Base name of the method element
Method Namespace	OUT	String	Namespace URI of the method element
Parameters	OUT	NodeIter	Iterator to collection of RPC parameter elements. Use the 'SOAP Get Next RPC Parameter' or 'XML Get Next Node' tool to iterate over collection.
Parameter Count	OUT	Integer	Number of items in the Parameters collection

SOAP Get Next RPC Parameter

This tool returns the element node at the current iterator position and returns an iterator to the next position. As the iterator is just a variable, you can make copies at any time to remember a certain position, for example the start position. By using the same variable as input and output iterator, you can easily iterate over the list by connecting the Success path back to this tool (after processing the node, of course). The tool takes the 'End' exit when the iterator points to an empty list or the iteration is complete (list traversed to end).

The tool will fail (take the Failure exit) if the node to which 'Parameter Iterator' points is not an element! This cannot happen if the iterator was obtained through 'SOAP Get RPC Method Info'.

Exit Paths: Success, End, Failure

Parameter	Dir	Type	Remarks
Parameter Iterator	IN	NodeIter	Iterator to collection of parameter of a method.
Retrieve Value	IN	Boolean	Checkbox: *False* Do not retrieve value *True* Default. Return node value Disable retrieval of value if value is not used and parameter may contain a large XML document.
Next Parameter	OUT	NodeIter	Iterator pointing to next parameter in the list
Parameter Element	OUT	Node	Node of the parameter element
Parameter Base Name	OUT	String	Base name of the parameter element
Parameter Namespace	OUT	String	Namespace URI of the parameter element
Value	OUT	String	Value of the parameter

SOAP Create RPC Response

This is a convenience tool for composing the response envelope for an RPC request. It copies the source envelope and replaces the method element in the body with an element that has the same name but "Response" added to its name. It also adds a <Result> element as child of the method element. Usually, the type of the return value is given by the service description and doesn't need to be included in the <Result> element. However, the service may define the type as **xsd:anyType**, for example for VARIANT types. In this case, the type must be included in the argument. The 'Return& Value Type' argument permits specifying the type of the result value. For example, if a type of "double" is specified, the <Result> element will look as follows:

```
<Result xmlns:xsd="http://www.w3.org/2001/XMLSchema"
     xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
     xsi:type="xsd:double">1234.567</Result>
```

The selection namespaces from the source envelope document are copied to the response envelope document as well. The tool fails if the request body does not contain a method element.

Exit Paths: Success, Failure

Parameter	Dir	Type	Remarks

Envelope	IN	Node	Envelope node of the request SOAP payload. Can be a document node whose document element is <SOAP-ENV:Envelope> or the node is the element itself.
Method Name Mask	IN	String	Optional. Mask to create the name of the response method. The string passed here may contain the following substitution tags: %1 Namespace prefix of the first child element of the <Body> element (RPC method). %2 Base name of the first child element of the <Body> element (RPC method). %{ Treat everything up to closing '}' as XPath query to be run against the 'Envelope' node and substitute the value of the first node found into element name string. %% '%' character Default: "%1:%2Response".
Method Namespace	IN	String	Optional. Namespace of the method element. If not specified, namespace of request method is used.
Result Element Name	IN	String	Optional. Name of the return value element (first child of the method element). Default: "result"
Result Element Namespace	IN	String	Optional. Namespace URI of the result element. If the parameter is omitted and the name has a namespace prefix, the tool will search in the parent elements for the namespace with the same prefix and make the element a member of that namespace.
Return Value	IN	String	Optional. Return value of the method. It will be set as content of the <Result> child element.
No Return Value (void response)	IN	Boolean	Checkbox: *False* Default. Add a <Result> element. *True* No <Result> element is added (void method).
Copy Header	IN	Boolean	Checkbox: *False* Default. Does not copy the <Header> element from the source envelope. *True* Copies the <Header> element and its content from the

			source envelope.
Copy Method Element Attributes	IN	Boolean	Checkbox: *False* Default. Don't copy attributes from request method element. *True* Copy all attributes of the request method element into response method element.
Response Envelope	OUT	Node	Document node of the response envelope
Method Element	OUT	Node	Node of the response method element.
Result Element	OUT	Node	Node of the <Result> element in the method element.

SOAP Set Element Type

In SOAP, the type of an argument or the return value is specified by the service description and doesn't need to be included in the payload. However, the service may define the type as **xsd:anyType**, for example for VARIANT types. In this case, the type must be included in the argument. For example, if a type of "double" is specified, an element will look as follows:

```
<Element xmlns:xsd="http://www.w3.org/2001/XMLSchema"
     xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
     xsi:type="xsd:double">1234.567</Element>.
```

The type may be a user defined (complex) type. For example:

```
<ns1:Order xmlns:ns1="uri:my-order-type"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:type="ns1:Order">
<ns1:Product>Watchmacallit</ns1:Product>
<ns1:Quantity>7</ns1:Quantity>
<ns1:Price>19.99</ns1:Price>
</ns1:Order>.
```

Please refer to http://www.w3.org/TR/xmlschema-0 or http://www.w3.org/TR/xmlschema-2 for details on the XML Schema Data types.

Parameter	Dir	Type	Remarks
Element	IN	Node	Node of an element whose Schema instance type to set
Type	IN	String	XSD type to declare for this element. The argument may either be just the type name or have schema namespace prefix, such as xsd:string. If the type argument does not contain a prefix, xsd will be used.
Type Namespace	IN	String	Optional. Namespace of the type. Default: http://www.w3.org/2001/XMLSchema
XSI Namespace	IN	String	Optional. XML Schema Instance namespace. Default: http://www.w3.org/2001/XMLSchema-instance
XSI Namespace Prefix	IN	String	Optional. Prefix of the schema instance namespace. Default: xsi
Declare Namespaces in Envelope	IN	Boolean	Checkbox: *False* Declares the XSD and XSI namespaces in the element itself. *True* Default. Declare the XSD and XSI namespaces in the Envelope element (actually, the document element is used, as this tool may be for other purposes than SOAP). If any of the parent elements already has a NS declaration for a prefix and the namespace URI is different, the declaration will be added to the element, and not the Envelope.

Exit Paths: Success, Failure

SOAP Create Array

Turns an element, for example an RPC parameter, into a SOAP array. The array is created for values supplied as list of strings or just a number of empty elements that can be populated with complex data. The following is a sample array as produced by this tool (default argument):

```
<Element xmlns:xsd="http://www.w3.org/2001/XMLSchema"
     xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
     xmlns:SOAP-ENC="http://schemas.xmlsoap.org/soap/encoding/"
     SOAP-ENC:arrayType="xsd:string[5]"
     xsi:type="SOAP-ENC:Array">
   <xsd:string>first</xsd:string>
```

```
   <xsd:string>second</xsd:string>
   <xsd:string>third</xsd:string>
   <xsd:string>fourth</xsd:string>
   <xsd:string>fifth</xsd:string>
</Element>
```

If the element already has child elements, they are all removed before the array elements are added. The array items may be user defined (complex) types. Use the 'XML Get Next Item' tool to iterate through the 'Item Elements' collection and populate the items. For example:

```
<Element xmlns:ns1="uri:my-order-type"
     xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
     xmlns:SOAP-ENC="http://schemas.xmlsoap.org/soap/encoding/"
     SOAP-ENC:arrayType="ns1:Order[3]"
     xsi:type="SOAP-ENC:Array">
   <ns1:Order>
     <ns1:Product>Watchmacallit</ns1:Product>
     <ns1:Quantity>3</ns1:Quantity>
     <ns1:Price>19.99</ns1:Price>
   </ns1:Order>
   <ns1:Order>
     <ns1:Product>Doodleany</ns1:Product>
     <ns1:Quantity>9</ns1:Quantity>
     <ns1:Price>12.49</ns1:Price>
   </ns1:Order>
   <ns1:Order>
     <ns1:Product>Ozadingdong</ns1:Product>
     <ns1:Quantity>1</ns1:Quantity>
     <ns1:Price>43.15</ns1:Price>
   </ns1:Order>
</Element>
```

For details on the XML Schema Data types, refer to http://www.w3.org/TR/xmlschema-0 or http://www.w3.org/TR/xmlschema-2.

Exit Paths: Success, Empty, Failure

Parameter	Dir	Type	Remarks
Element	IN	Node	Node of the parameter to turn into an array.
Values	IN	StringList	Optional. List of strings to set as the array items. If not specified, empty elements will be created.
Size	IN	Integer	Optional. Size of the array. If not specified, the length of the ‘Values' list specifies the size. If both a ‘Values' and ‘Size' argument are given, the ‘Size' has precedent and either not all items of the ‘Values' list are included or the array is padded with elements containing the ‘Default Value'.
Default Value	IN	String	Optional. Default array item value for padding items (if ‘Size' is larger than size of ‘Values' or no ‘Values' defined). Default: No value (padding elements will be empty)
Array Type	IN	String	Optional. Type of the array. The argument may either be just the type name or have schema namespace prefix, such as xsd:string. If the type argument does not have a prefix, xsd will be used. Default: xsd:string
Type Namespace	IN	String	Optional. Namespace of the array type. Default: http://www.w3.org/2001/XMLSchema
Encoding Prefix	IN	String	Optional. Prefix of the encoding namespace (http://schemas.xmlsoap.org/soap/encoding/). Default: SOAP-ENC
Item Element Name	IN	String	Optional. Qualified name of the array items. Default: Qualified array type (thus, the default item element name is xsd:string).
Item Element Namespace	IN	String	Optional. Namespace of the array items. Default: Namespace of the prefix of ‘Item Element Name'. If no prefix, empty namespace.
XSI Namespace	IN	String	Optional. XML Schema Instance namespace. Default: http://www.w3.org/2001/XMLSchema-instance
XSI Namespace Prefix	IN	String	Optional. Prefix of the schema instance namespace. Default: xsi
Include XSI Type Declaration	IN	String	Checkbox:

			False Do not add a type declaration for the array. *True* Default. Add XSI type declaration for the SOAP Array. If all parameters are default the declaration is: xsi:type="SOAP-ENC:Array".
Declare Namespaces in Envelope	IN	Boolean	Checkbox: *False* Declares the namespaces in the element itself. *True* Default. Declare the namespaces in the Envelope element (if they aren't already). If any of the parent elements already has a NS declaration for a prefix and the namespace URI is different, the declaration will be added to the element, and not the Envelope.
Return Item Element Collection	IN	Boolean	Checkbox: *False* Does not return collection of array items (‘Array Items' is returned as NULL). *True* Default. Return collection ‘Array Items' containing all items of the array.
Item Elements	OUT	NodeIter	Iterator pointing to first element of a collection containing the nodes of the array items.
Count	OUT	Integer	Number of items in the array. NOTE: this value is returned, even if ‘Return Item Collection' is False.

Invocation Tools

Invocation Tools

SOAP HTTP Request

This tool issues an HTTP request to the specified URL with the SOAP request envelope as payload. The response body is parsed and returned as response envelope. The URL may have the following format (also see RFC2396 at http://www.rfc.net/rfc2396.html):

```
['http://'] <host> [':' <port>] ['/' <path> ['?' <query>]]
```

The (UNICODE) string passed as URL is converted to UTF-8 and invalid characters in the resulting string are escaped according to RFC2396 (%<hexvalue>). The structure of the request sent to the host will be as follows:

```
'POST ' <path> ' HTTP/1.1' CRLF
'Host: ' <host> [':' <port>] CRLF
'Content-Type: text/xml; charset="' <charset> '"' CRLF
'Content-Length: ' <bodysize> CRLF
'SOAPAction: "' <SOAPAction> '"' CRLF
[<additional headers>]
CRLF
<SOAP envelope body>
```

The ‘Additional HTTP Headers' parameter can be used to supply additional HTTP header elements. The headers must have the form *{<name>* ' :' *<value> [CR] LF }** The header elements in this argument have precedence over the default headers generated by the tool. Thus, if the ‘Additional HTTP Headers' parameter contains a ‘Content-Length header, it will be used (with potentially unexpected results, of course).

The response body will be parsed and returned as ‘Response Envelope' if the content type is text/xml. Otherwise, the body is returned in ‘Raw Response Body' and an empty document node is returned as ‘Response Envelope'. This document node can be queried for information about what went wrong.

This tool maintains a global cache of the most recently resolved and successfully connected host addresses to improve performance. Each address resolution is kept for at most 5 minutes.

Exit Paths

Success

Request was processed successfully (2xx code) and body is valid XML.

SOAP Fault

Response body contains a <Fault> element.

EmptyResponse

Response body was empty and the HTTP status code was 2xx. Some servers use this to signal success for methods with no result (void).

Unknown Host

Invalid or unknown hostname (DNS lookup failed)

Connection Error

Unable to establish connection to server: connection failed or existing connection was lost

prematurely.

HTTP Error

HTTP error (3xx, 4xx, 5xx) and it was not a SOAP Fault.

Parse Error

Error parsing the returned XML payload (status was 200 or 500).

Timeout

The request timed out.

Size Limit

The response data exceeded the size limit.

Failure

Some other failure. Use ‘XML Get Error Info' on the ‘Response Envelope' to obtain more information.

Parameter	Dir	Type	Remarks
Request Envelope	IN	Node	XML Node of the SOAP envelope to send. Can be document node or <Envelope> element node.
URL	IN	String	URL of the request. See remarks for details.
SOAP Action	IN	String	Optional. String to be passed as SOAPAction header. The string passed here may contain the following substitution tags: %1 Namespace of the first child element of the <Body> element (RPC method). %2 Base name of the first child element of the <Body> element (RPC method). %{ Treat everything up to closing ‘}' as XPath query to be run against the ‘Request Envelope' node and substitute the value of the first node found into the SOAPAction string. %% ‘%' character If this argument is not specified, the following mask will be used as default: "%1#%2".

			The value "NONE" may be specified to suppress addition of the SOAPAction header.
Additional HTTP Headers	IN	String	Optional. Additional HTTP Headers, separated by LF characters (\n). See remarks for details.
Selection Namespaces	IN	String	Optional. Selection namespaces to set in response envelope document. Default: Copy selection namespaces from request envelope document.
Timeout	IN	Integer	Optional. Maximum time the request may take before timing out (in milliseconds). -1 à Never timeout. Default: 60000 (60 seconds)
Max Response Size	IN	Integer	Optional. Size limit of the response data. If the data returned by the server exceeds this limit, the data is not processed and the tool fails. This prevents denial of service attacks. Default: 1MB.
Escape URL	IN	Boolean	Checkbox: *False* URL is already escaped. *True* Default. Escape invalid characters in the URL with %<hexvalue> according to RFC2396.
Always Return Raw Response Body	IN	Boolean	Checkbox: *False* Default. Do not return raw response body. *True* Returns the raw data of the response body as string (‘Raw Response Body).
Response Envelope	OUT	Node	Document node of the response envelope. If an error occurred, an empty document is returned which can be queried using ‘XML Get Error Info'.
Status Code	OUT	Integer	HTTP status code of the response (e.g. 200, 500, etc).
Status Text	OUT	String	HTTP status text of the response (e.g. "OK", "Internal Server Error", etc.)
Response Headers	OUT	String	HTTP Headers returned by the server, separated by a LF (\n).
Raw Response Body	OUT	String	Raw data of the response body (data that is parsed as response envelope). This string is only returned if the ‘Always Return Raw Response Body' parameter is True, an error occurs, or the response content type is not XML.

Helper Tools

Helper Tools

SOAP Base64 Encode

Converts the string (which is UNICODE) into the specified character set (default = UTF-8) and encodes the resulting data into a Base64 string. Characters that cannot be translated to the destination character set will be represented as '?'. Wide character sets, such as UTF-16 are currently not supported. SOAP does not mandate a maximum line width for base64 encoded data. Some other protocols, such as MIME, do.

Exit Paths: Success, Failure

Parameter	Dir	Type	Remarks
Data	IN	String	String to encode Base64
Character Set	IN	String	Optional. Character set to convert data into before encoding. Default: 'UTF-8'
Max Line Width	IN	Integer	Optional. Maximum width of a line in characters. –1 = unlimited (default).
Line Separator	IN	String	Optional. String inserted as line separator. Default: "\r\n" (CR/LF)
Encoded Data	OUT	String	String after encoding data Base64

SOAP Base64 Decode

Decodes the base64 encoded string into the binary representation and converts it to UNICODE based on the specified character set. Thus, the character set argument specifies the character set of the base-64 encoded data.

Exit Paths: Success, Failure

Parameter	Dir	Type	Remarks
Encoded Data	IN	String	Base64 encoded data
Character	IN	String	Optional. Character set of the base64 encoded data. Default:

Set			'UTF-8'
Decoded Data	OUT	String	Data after decoding from Base64 and transforming from 'Character Set' to UNICODE.

SOAP Base64 Encode File

Reads the specified file as binary data and encodes it into a base64 string. Encoding a file prepares it for transport inside a SOAP payload. For example, a SOAP request might encode a wave file, and send it to CIC server. SOAP does not mandate a maximum line width for base64 encoded data. Some other protocols, such as MIME, do. This tool can be used to send any kind of data through SOAP requests. For example, you could encode a wave file.

Exit Paths: Success, File Not Found, Access Denied, Failure

Parameter	Dir	Type	Remarks
Filename	IN	String	Filename and path of the file to encode
Max Line Width	IN	Integer	Optional. Maximum width of a line in characters. –1 = unlimited (default).
Line Separator	IN	String	Optional. String inserted as line separator. Default: "\r\n" (CR/LF)
Encoded Data	OUT	String	Base64 encoded content of the file

SOAP Base64 Decode To File

Decodes the base64 encoded string into the binary representation and writes the data to the specified file as binary data.

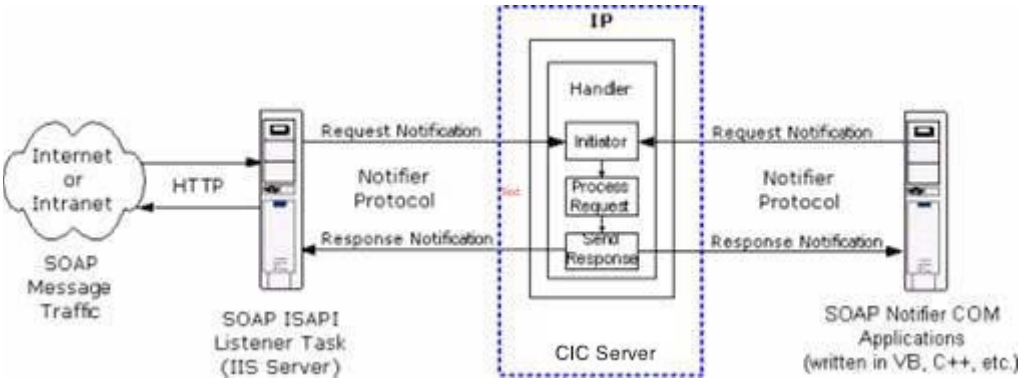
Exit Paths: Success, Access Denied, Failure

Parameter	Dir	Type	Remarks
Encoded Data	IN	String	Base64 encoded data
Filename	IN	String	Filename and path of the file to which to write the decoded data.
Append To Existing File	IN	Boolean	Checkbox: *False* Default. Create new file or truncate existing file. *True* Create new file or append to existing file.

Appendix C: Structure of IP Notification Messages

Appendix C: Structure of IP Notification Messages

For the purpose of the 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 (see SOAP ISAPI Listener Task for IIS). However, any kind of client who "talks" Notifier could issue SOAP requests. For example, a COM object that allows to directly send SOAP packets to CIC.



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

Since Interaction Processor does not directly support Notifier requests, notifications are used to emulate the request/response mechanism. The SOAP request notifications use CIC's eSOAP_REQUEST_OBJECT object type and an object ID that identifies the client. The notification event ("Initiator Event") can either be explicitly specified or the SOAPAction is will be used as default. 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.

Request Message 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 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.

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). See SOAP ISAPI Filter Schema for schema details.
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

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.

Appendix D: SOAP ISAPI Listener Fault Messages

This appendix lists fault messages returned by the SOAP ISAPI Listener. For general information about SOAP Faults, refer to section 4.4 of the SOAP Specification at W3C. The URL is http://www.w3.org/TR/SOAP/. SOAP ISAPI Listener may return the following codes:

Client.ContentType

Unsupported Content-Type specified. Expecting "text/xml" or "application/xml".

Client.ContentLength

The 'Content-Length' field of the HTTP header does not match the length of the data sent by client.

Client.SOAPAction

The HTTP header does not contain a 'SOAPAction' header field.

Client.PayloadSize

The SOAP payload exceeds the maximum size limit configured for the server.

Server.TooBusy

Server is too busy—too many requests are currently pending.

Server.SOAPAction

The SOAPAction is not recognized by the server (e.g. it doesn't match any filter rules).

Server.NotifierConnection

SOAP ISAPI Listener was unable to establish a Notifier connection with the CIC server to forward the request.

Server.RequestTimeout

The request was not processed by the CIC server in the allotted time.

Server.NotifierConnectionLost

The SOAP ISAPI Listener lost the Notifier connection while waiting for the request to be processed by the CIC server.

Server.Switchover

A Switchover was initiated while waiting for the request to be processed. The response was lost.

Server.Error

A general error occurred while server was waiting for request to be processed.

Server.Unhandled

The request was not processed by the CIC server (i.e. a handler was initiated but did not send a response with the SOAP Send Response tool).

Server.Shutdown

The web server was shut down (ISAPI unloaded) while the request was being processed by the CIC server.

Glossary

This section explains special terms used in this documentation.

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 CIC modules.

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.

Customer Interaction Center (CIC)

Customer Interaction Center offers comprehensive interaction management covering not only telephone calls, faxes, and e-mail messages, but also Internet text chats, Web callback requests, and voice over IP calls. Using CIC and the PureConnect platform,, enterprises, contact centers, and service providers can centralize the processing of all customer interactions and provide a new level of service and consistency.

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:

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.

HTML

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

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.

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.

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.

Revisions

CIC 2018 R2

1. Added procedure, Configuring IC SOAP Listener to work with IC 4.0 and 2015 or later.
2. Added procedure, Additional configuration steps required for SOAP Listener when using IIS7.

CIC 2018 R1

1. Rebranded this document to apply Genesys terminology. Colorized source code. Updated formatting, copyright and trademarks.
2. Deprecated the procedure titled "Install Microsoft SOAP Install Toolkit". Installing the toolkit is no longer necessary. All SOAP Toolkits were replaced by the Microsoft .NET Framework. SOAP Toolkits are no longer supported.

CIC 2015 R4

Added information about new <ICServer2> element in configuration file.

CIC 2015 R1

Updated documentation to reflect changes required in the transition from version 4.0 SU# to CIC 2015 R1, such as updates to product version numbers, system requirements, installation procedures, references to Interactive Intelligence Product Information site URLs, and copyright and trademark information.

CIC 4.0 SU1 and SU2

No revisions were made to this document.

CIC 4.0 GA

1. Installation should be performed using the CIC 4.0 GA DVD. Do not use an CIC 3.x DVD.
2. Updated copyrights and trademarks in this document.
3. The Installing and Using SOAP Functionality Technical Reference Guide was renamed to CIC and SOAP API Developer's Guide. The filename was changed from soap.chm to Soap_API_DG.chm.
4. The *SOAP Notifier COM API Developer Guide* was renamed to SOAP Notifier COM API Developer's Guide. The file name was changed from soapnotifiercom.chm to Soap_Notifier_COM_API_DG.chm.
5. Updated setup instructions for minor changes made to installs.

Copyright and Trademark Information

Genesys Telecommunications Laboratories, Inc.
2001 Junipero Serra Boulevard
Daly City, CA 94014
Telephone/Fax (844) 274-5992
www.genesys.com

CIC and SOAP API Developer's Guide

Audience

SOAP stands for *Simple Object Access Protocol*. SOAP is an XML-based protocol specification that defines how information can be exchanged between computers. SOAP supplies the conventions used to invoke methods on servers, services, components and objects. This document introduces XML/SOAP concepts and explains how SOAP facilitates robust data interactions between CIC and remote web services. SOAP supplies the conventions used to invoke methods on remote servers, services, components and objects.

This publication is for managers, technical implementers, and other decision-makers who need to understand the practical implications of SOAP technology in the CIC environment. The introduction is written for a general audience who may not be familiar with XML or SOAP technology. Subsequent sections of this document guide technical implementers through the process of preplanning, installing and configuring the SOAP ISAPI Listener Task and SOAP Notifier COM Components. Instructions for using the SOAP Tracer utility are also provided.

Organization of Material

This documentation is divided into logical, easy-to-digest sections that gradually introduce concepts and specific product features. To fully understand the material, we recommend that you read topics in order. However, most topics are hyperlinked for those who prefer to read in non-linear fashion.

- Introduction to SOAP in the CIC Environment provides short primers on XML and SOAP, and explains the relationship between XML, SOAP and the Interaction Center platform. It introduces CIC's SOAP Components.
- Install and Configure SOAP ISAPI Listener explains how to select a host server, apply prerequisite service packs and hotfixes, and then install SOAP Listener components. This section also explains how to configure the server to prevent denial of service attacks, and how to modify the configuration so that only supported SOAPActions are forwarded to CIC for processing.

- Install SOAP Notifier COM explains how to install and register components needed to run or develop third-party SOAPNotifierCOM applications on a desktop PC.
- Appendix A (SOAP Transport Information) describes HTTP schema used to transport SOAP packets in the CIC environment. This appendix is for advanced readers who are curious about SOAP transport mechanisms used in CIC.
- Appendix B (SOAP Tools) describes tools in Interaction Designer that process SOAP requests and responses.
- Appendix C (Structure of IP Notification Messages) explains the notification message format and protocols used to send requests to and from CIC's Notifier subsystem.
- Appendix D (SOAP ISAPI Listener Fault Messages) is a reference about fault messages returned by the SOAP ISAPI Listener.
- Special terms used with SOAP technology are defined in a Glossary.
- Revisions describes what's new by release.

Related Documentation

1. *CIC and SOAP API Developer's Guide* (this document). This paper provides primers on SOAP and XML, and discusses the components that must be installed to implement SOAP functionality in CIC.
2. *Interaction Center SOAP Listener* Setup installs SOAP ISAPI components on an IIS server. We highly recommend that you read Install and Configure SOAP ISAPI Listener before running the install.
3. The *SOAP Notifier COM Components Install* installs and registers component software used by developers to create high-performance SOAP applications.
4. *SOAP Notifier COM* setup optionally installs the *SOAP Notifier COM API Developer's Guide* (Soap_Notifier_COM_API_DG.chm). This windows help file cross-references the interfaces, methods, and properties exposed by SOAP Notifier COM objects.
5. *SOAP Tools* are documented in Interaction Designer help. These help topics appear when a SOAP tool or toolstep has focus and the F1 key is pressed in Interaction Designer.

Recommended Web Links

XML Home Page at the World Wide Web Consortium (W3C)

http://www.w3.org/XML/

XML Tutorial by W3Schools

http://www.w3schools.com/xml/default.asp

O'Reilly XML.COM

http://www.xml.com/

W3C SOAP specification document:

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

SOAP Tutorial by W3Schools

https://www.w3schools.com/xml/xml_soap.asp

Web Services Description Language (WSDL) 1.1

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

Namespaces in XML

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

Introduction to SOAP in the CIC Environment

Introduction to SOAP in the CIC Environment

This section is for managers and other decision makers who need to understand the practical implications of SOAP technology in a CIC environment. No prior knowledge of XML or SOAP is required to understand the concepts presented here. XML and SOAP are standards for information exchange that were developed for the Internet.

What is SOAP?

SOAP stands for *Simple Object Access Protocol*. SOAP is an XML-based wire protocol designed for decentralized, distributed networks such as the Internet. SOAP defines conventions that allow a computer to invoke a remote procedure in another. These remote procedure calls (SOAP requests) can be transported using a variety of network protocols.

For example, the SOAP Listener task on an IIS server uses HTTP protocol to transport SOAP messages to and from the Internet. Applications developed using SOAP Notifier COM components use Notifier protocol to transport SOAP messages to and from CIC server. SOAP itself is unconcerned with the protocol used for transport. For this reason, SOAP can be used on many types of computer networks.

SOAP makes it possible for programs running on different computers to request and receive data from one another in a structured way, even when different operating systems are used. SOAP provides the XML vocabulary needed to specify method parameters, return values, and exceptions.

SOAP empowers remote computers to start handlers on CIC and receive data from CIC in response. SOAP extends CIC interoperability to the entire Internet. Anything that "talks" SOAP through HTTP can communicate with CIC. Any computer platform (Windows, Unix, Linux, Mac, etc.) that can create and transport a SOAP message request can start a handler on CIC. Depending upon the type of request, the handler may or may not send back a response containing values looked up by CIC.

For example, a Unix Server might use *Enterprise JavaBeans* (EJB) to generate a SOAP Message requesting information about a user's status. When the request is received by CIC, it starts a handler that looks up the user's status, generates a SOAP response, and transports the response back to the

requesting server. When the Unix system receives this *SOAP payload*, it uses another EJB to parse and process the information.

Conversely, handlers created using CIC's *SOAP tools* can request data from web services and remote procedures. For example, a handler might request the current price of a stock from a brokerage service, check inventory levels from an inventory management system, conduct a credit card transaction, or obtain a weather report. SOAP support in CIC is implemented by SOAP tools in Interaction Designer that define initiators, invoke remote procedures, process requests and payloads. SOAP messages are channeled through a SOAP ISAPI Listener task that runs on an IIS server. Developers can optionally use SOAP Notifier COM components to develop COM applications that directly invoke SOAP handlers. SOAP Notifier COM components are compatible with any language/application that supports Microsoft's Component Object Model. These options are discussed later in this document.

Who uses CIC's SOAP functionality?

SOAP tools support open standards (SOAP, XML, WSDL, etc.) These tools promote interoperability and are applicable to many types of application development. SOAP tools are primarily used by developers, advanced handler authors, and professional services personnel. However, the *services* created using SOAP tools are another matter. Anyone, anywhere on the Internet is potentially a consumer or provider of information processed by SOAP handlers. The possibilities are limitless.

For example, an caller might enter a PIN number into an auto-attendant menu created using Interaction Attendant. In turn, Attendant could start a SOAP handler that passes the PIN number to a remote web service to look up information that is spoken back to the caller using CIC's text-to-speech capability. A remote procedure invoked by SOAP can perform any kind of data processing tasks, ranging from a simple lookups to complex transactions that accept complex data types as input. SOAP does not impose any limits on the application functionality that can be invoked.

- OAP tools allow developers to create handlers that retrieve data from web services, or which function as web services. Handler-based services can be described using *Web Services Description Language (WSDL)*—an XML-based language that defines the functionality offered by a web service and how to access it. WSDL makes it possible to describe a service on CIC so that a worldwide audience can find and use it. WSDL describes the service, all parameters required to invoke it, and the location (endpoint) where the service can be accessed.
- WSDL's are not available for handler examples included with this release. However, you can easily create WSDL's to describe the example files.
- SOAP makes it possible for programs written in different languages and running on different platforms to communicate with each other.
- SOAP integrates CIC with business-to-business interactions and information services.
- Once a SOAP *endpoint* is exposed to the internet, a handler may call into the endpoint, which may be on the Internet or an Intranet.

SOAP is not appropriate for low-level, tightly-coupled transactions, due to network latency and the overhead imposed by the SOAP messaging encoding and decoding. SOAP is best suited for simple, high-level transactions, such as sending a name and PIN number to a service to obtain an account summary.

SOAP's Request/Response Model

CIC uses a *request/response* model to process SOAP requests. This mechanism should be familiar to anyone who has used a web browser.

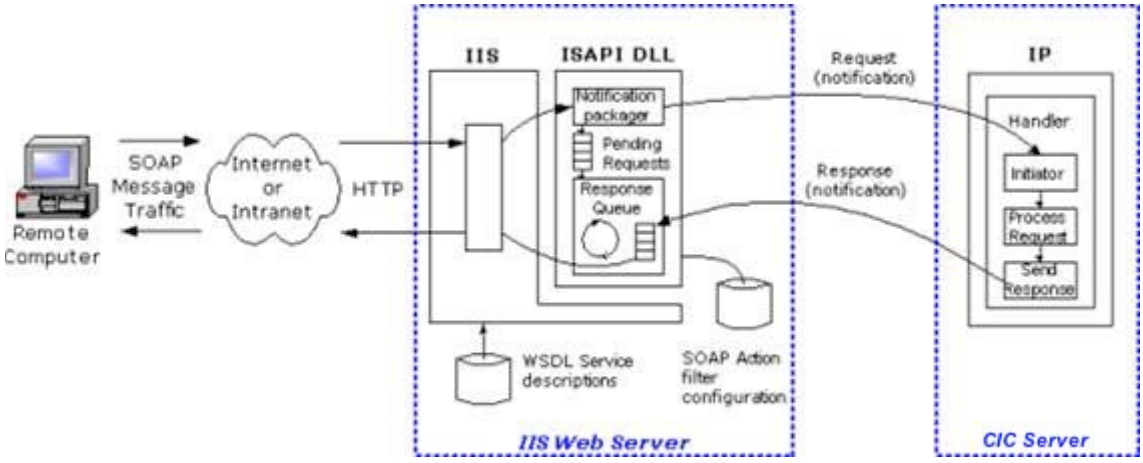
1. A *client* (e.g. web browser) connects to a *server* and passes a request (fetch a web page). The client then waits for the server to respond.
2. The server responds in one of two ways. It either returns the requested information, or it responds with an error message that tells the client why the request could not be completed.
3. Once the server has responded to the client, it closes the connection, discards all state information about the transaction, and listens for another request.

Web Services

In the world of SOAP, the client is a computer program that asks a server (another computer program) to execute a method (sometimes called a *web service*).

In CIC configuration, HTTP requests are received by *SOAP Listener*—an ISAPI DLL that runs on an IIS web server. SOAP Listener passes requests to the CIC Notifier subsystem for processing. Notifier alerts the Interaction Processor subsystem, which in turn starts the handler needed to process the request. Response data from the handler is passed back to the Listener task for transport to the remote computer.

In general, SOAP Listener translates HTTP requests into notifications and acts as a gatekeeper to prevent denial of service attacks.



On the receiving end, the response message is decoded and used by the requesting computer in some way. This low-overhead approach permits a single server to share information with many clients.

Requests and Responses are XML Documents

In order for the request/response model to work, messages must be formatted in a way that both computers understand. SOAP uses XML to accomplish this.

What is XML?

XML stands for *Extensible Markup Language*. XML provides a structured way to define data in plain text format, so that data can be exchanged between computers. SOAP messages are XML documents, which are just text files formatted according to some very specific guidelines. (SOAP is the specification that defines the guidelines used to describe remote procedure calls using XML.) XML provides the syntax needed to define a markup *vocabulary*—the tags and attributes needed to describe a particular type of data. XML files can be created using a simple text editor, such as Notepad. XML is more flexible than comma-delimited or fixed-length formats, since XML encloses information inside descriptive tags in a tree-based hierarchy. Before a SOAP request can be transported to another computer, the request is structured using XML so that the remote system can interpret the request in accordance with the SOAP specification. Responses from the remote procedure are returned as XML documents.

SOAP uses XML to package the data passed to a method, or received as a response. SOAP itself is nothing more than a set of rules that define how to describe method calls and return values using XML syntax. XML merely describes data, without consideration for the way that the data is processed or presented.

To summarize, SOAP defines conventions needed to invoke the methods of a web service. SOAP tools on CIC allow web services to be created using Interaction Designer. SOAP uses existing transport protocols (such as HTTP) to transmit an *XML payload* to another computer. The payload contains everything that the remote computer needs to execute a function (arguments and data). Services that understand SOAP requests can be expected to return XML responses in accordance with the rules of SOAP. The relationship between SOAP and XML can be expressed this way:

SOAP documents are XML documents that conform to a particular specification, allowing the exchange of messages. Therefore, to understand SOAP, you need a working knowledge of XML.

What is the relationship between XML and markup languages, such as HTML or SGML?

What is the relationship between XML and markup languages, such as HTML or SGML?

If you use the Internet, you probably know that HTML is the markup language used to create World Wide Web pages. (HTML stands for *Hypertext Markup Language*.) HTML and XML are both descendants of an earlier markup language called SGML (*Standard Generalized Markup Language*). SGML is a complicated set of rules that define document structures. XML is a subset of SGML that does the same thing, using fewer rules. Since XML is a less-complicated derivative of SGML, XML is more easily implemented on large networks such as the Internet. The primary role of XML is to *define data*.

XML delivers the power of SGML without the complexity. XML does not utilize features that make the authoring difficult or costly. Yet XML preserves most of the flexibility and richness associated with SGML.

Web browsers use a combined parsing and presentation engine that is tolerant of markup problems. Sloppy markup in HTML pages is ignored or interpreted in a proprietary way. For example, if a closing tag is omitted in an HTML document, the browser attempts to guess where the closing tag should have been. If the browser encounters a tag or attribute that it does not recognize (such as a tag supported by a different brand of browser), the tag or element is ignored.

The loose, uncontrolled nature of HTML makes it impossible to predict exactly how a web page will be displayed. Browsers attempt to render *something* on-screen, however odd, rather than display validation error messages. Since HTML is presentation-oriented, it uses markup tags for formatting as well as to define structure. The complexity of HTML formatting can make it difficult to locate data in HTML documents. HTML was not originally designed to provide *precise* control over the layout of page elements. To compensate, savvy page designers use tables, style sheets, and DHTML layers to control the placement of text and graphics. This creates visually-appealing web pages at the expense of clear-cut document structures. Complex web pages bury data in a mix of structures in the information stream. The lack of structural consistency in HTML documents makes it difficult for computer programs to locate, extract or update data. XML resolves this problem, by demanding that document authors get structure and syntax right.

XML Parsers

XML documents are often *parsed* to ensure that they are *valid* and *well-formed*.

- A *well-formed* document conforms to the XML specification.
- 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.

It is important to note the distinction between parsers and browsers. Parsers validate data. Browsers display information. SGML and XML are focused on *parsing* documents rather than presenting them. Parsing is the computer equivalent of reading a document. A parser is a program that reads in a text file, breaks it down into component parts, and validates the document using rules in a DTD file. Internet Explorer offers a built-in parser that you can use to validate XML files. For details, see Viewing XML in Internet Explorer.

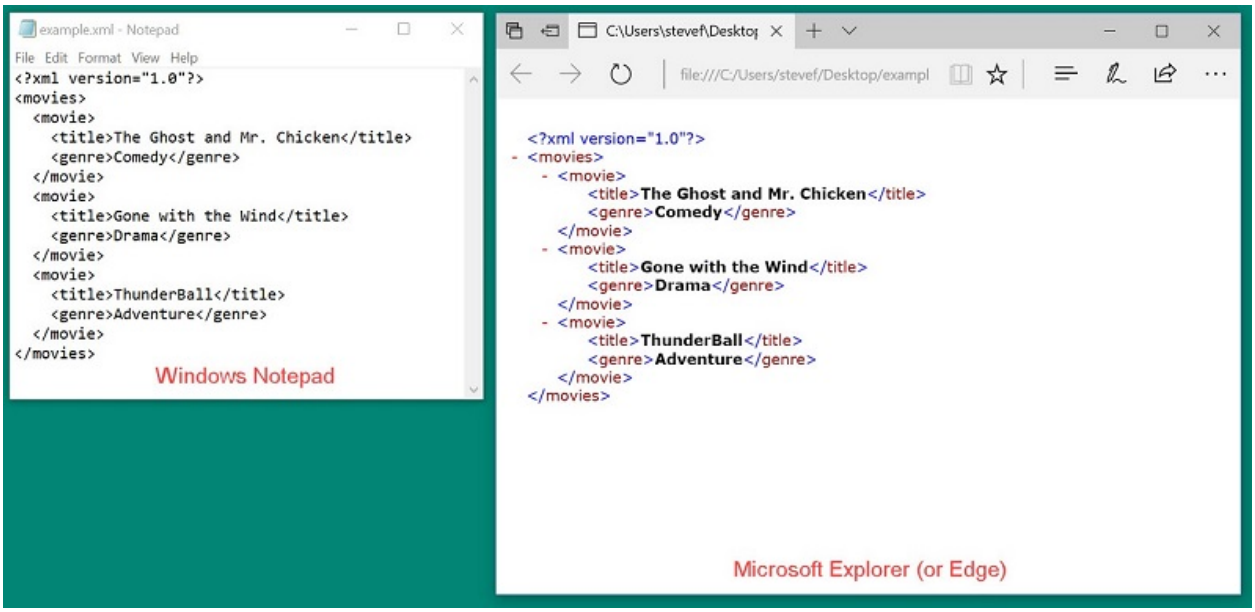
DTD stands for *Document Type Definition*. DTD's define hierarchy structure and elements that can be used in an XML document. For links to DTD tutorials, see Recommended Web Links.) The role of a parser is to identify portions of a document that are invalid in terms of structure or syntax. XML and SGML parsers ensure that documents are coded correctly.

Viewing XML in Internet Explorer or Edge

The tree structure of XML documents is easy to understand when seen visually. Microsoft's Edge and Internet Explorer 6 (or later) browsers provide a built-in parser that you can use parse, validate, and view XML files.

Tip: To open an XML file, drag and drop an XML file from Windows Explorer into your browser's document window. Or, double-click an .xml filename in Windows Explorer.

The figure below shows the sample movie database (sample1.xml) after it has been opened in Notepad and Internet Explorer. As you can see, Notepad displays the statements appear as they were entered. Edge and Internet Explorer display a tree of elements, which makes the content easier to view.



Edge and Internet Explorer automatically add DHTML code so that you can expand or collapse nodes in the tree. Internet Explorer doesn't allow you to do much besides view XML files. However, if you save your XML file with an extension of .htm or .html, IE will render the data contained in the XML file.

Advantages of XML over HTML

XML syntax closely resembles HTML; data is enclosed between opening and closing tags. However, XML is more flexible than HTML:

- XML encodes data in tightly-validated tree structures. Data is easy to locate since its context is well defined by tags and rules of structure.
- HTML attempts to control the appearance and presentation of data, while XML does not. XML defines data separately from its presentation. This makes XML data easier to locate and manipulate.
- XML is a standard data format that permits applications to exchange information across platforms and operating systems. HTML is markup used to display information in a web browser.
- XML is open and extensible. XML authors can create their own tags. HTML is limited by a fixed vocabulary that browser developers have agreed to support. In fact, XML has no predefined tags of its own. New XML tags are defined as needed —to define any type of data using syntactical rules that that permit browsers and XML Parsers to interpret proprietary tags on the fly. XML can describe any kind of data, such as a row in a table, a chemical formula, a financial transaction, a short story, or an object that exposes methods and properties—with equal finesse.
- Since XML is plain text, it is easily transmitted between computers and through firewalls. XML is more secure than binary files, since text files cannot be executed directly. Binary files, on the other hand, can contain malicious computer programs.
- XML is universally compatible. The XML file format is not tied to any particular program, operating system, database, or network. XML can be used by non-web applications to store data.

- XML files can be *transformed* into other types of documents. Transformation is controlled using XSL style sheets.

  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. For these reasons, XML is becoming the preferred format for e-commerce and information exchange between computers of all types. XSL style sheets can reorder documents, display or hide information, or apply formatting, among many other things. XSL uses patterns and logical operations to determine which parts of a document tree it should transform. XSL works somewhat like a programming language—it can test for equality and perform processing based upon the results of a test.

Structure of an XML file

An XML file is just a structured text file. The best way to understand XML is to look at example files. Listing 1 below contains three records from a movie database. Each record contains two fields: the *title* of a movie, and its *genre*.

The example file is formatted using blank lines, tabs and white space that make the file easier to read. In practice, those items are ignored by XML parsers. Likewise, bold text and line numbers in the listing are for illustration purposes only. Actual XML files do not contain line numbers.

Listing 1: Sample XML File

```
1  <?xml version="1.0"?>
2  <movies>
3    <movie>
4       <title>The Ghost and Mr. Chicken</title>
5       <genre>Comedy</genre>
6    </movie>
7    <movie>
8       <title>Gone with the Wind</title>
9       <genre>Drama</genre>
10   </movie>
11   <movie>
12      <title>ThunderBall</title>
13      <genre>Adventure</genre>
14   </movie>
15 </movies>
```

XML Declaration

Line 1 contains a processing instruction known as the XML *declaration*. This statement tells parsers that the file contains XML. The remainder of the file is composed of XML *elements*. Each element consists of a *start tag* and an *end tag*. XML *data* is just information that appears between tags.

The terms *tag* and *element* are often used interchangeably. A tag is an identifier that defines something. An element is an instance of a set of tags. In our example, <title> is a tag, and <title>Gone with the Wind</title> is an element. Elements are the basic building blocks of HTML files. Elements can be nested inside of other elements.

Rules that govern tags

Tags are governed by a few basic rules:

- Tag names are case-sensitive. <movie>, <Movie>, and <MOVIE> are not equivalent. Attribute names are also case-sensitive.
- Tag names must begin with an alphabetic character, an underscore, or a colon.
- Tag and attribute names cannot begin with "xml", which is reserved.
- All tags must be closed. A start tag must be closed by a corresponding end tag. Empty elements with no attributes can use a backslash as a shortcut for the end tag (e.g. <movie/> is equivalent to <movie></movie>.

The Root Element

Line 2 defines the **root element**. Since an XML document is a tree of elements, each document has a single root element that denotes the beginning and end of the XML statements in the file. In the example, the root element begins with a start tag <movies> and is closed by an end tag </movies>. All other elements are nested inside the root element.

Child Elements

Line 3 identifies <movie> as a **child** of the <movies> root element. Parent-child relationships are common in XML files. Parent elements can have many children. All elements must be properly closed, meaning that each element has a start tag and an end tag. Likewise, tags must be balanced. The close tag of a child cannot appear after the close tag of its parent. For example:

```
<title>ThunderBall<genre>Adventure</title></genre> is incorrect.
<title>ThunderBall<genre>Adventure</genre></title> is correct.
```

Line 4 contains some data (the title of a movie) between tags that identify the data.

Line 5 contains a different data item. In this case, it is a movie category between genre tags.

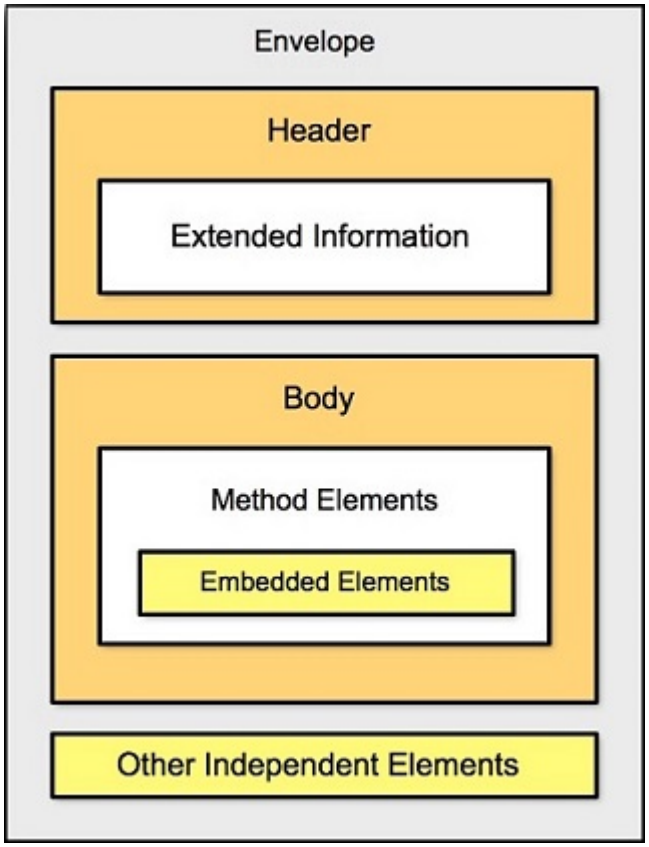
Line 6 closes this movie element.

This basic structure is repeated in lines 7 through 14, which define two more records.

Line 15 contains the closing tag for the root element.

Structure of SOAP Messages

Structure of SOAP Messages



SOAP messages are constructed using a framework that describes what is in a SOAP message, and how it should be processed. This is known as the SOAP *envelope*.

SOAP messages may contain *encoding rules*, which express instances of application-defined data types. Remote procedure calls and responses are also described in a SOAP message. As mentioned earlier, there are two types of SOAP messages:

- ***Request*** messages ask a remote process to perform some sort of processing.
- ***Response*** messages are replies from a remote process that return data or an error message that indicates why the request could not be processed.

The ***payload*** contains data in XML format that is passed to or from a function. R*equest 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. SOAP uses XML to express payload information accurately and concisely. Every SOAP message has a main envelope section, which can contain header and body sub-sections.

Envelope Section

The envelope is always the outer most element. Everything else in a SOAP message appears inside **SOAP-ENV** tags. The envelope in Listing 2 is empty—it doesn't contain any header or body tags.

Listing 2: SOAP Envelope Elements

```
1 <SOAP-ENV:Envelope xmlns:SOAP-
ENV="http://schemas.xmlsoap.org/soap/envelope/"
2    SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"/>
3 </SOAP-ENV:Envelope>
```

---

1. Line 1 of the envelope refers an external XML *namespace* (**xmlns**) that defines elements and attributes that can appear in the envelope (such as header or body elements).
2. Namespacesresolve 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 an "Egg Quality" namespace.
3. Most SOAP envelopes refer to XML schema defined by the W3C. It is very common to see http://schemas.xmlsoap.org/soap/envelope/ as the namespace reference in a message envelope.

   XML Schema are the successor to DTDs for XML. XML schema 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.

4. Line 2 refers to **encodingStyle** schema that describes basic data types (Booleans, Integers, Strings, etc.) that can be passed to a remote procedure call. SOAP messages typically define encoding rules using the W3C schema at http://schemas.xmlsoap.org/soap/encoding/.
5. Line 3 closes the envelope.

Header Section

As mentioned earlier, the envelope can contain header and body sections. These are defined using **Header** and **Body** elements. Listing 3 shows a SOAP message with empty Header and Body sections.

Listing 3: Header and Body Sections of a SOAP Envelope

```
1  <SOAP-ENV:Envelope xmlns:SOAP-
ENV="http://schemas.xmlsoap.org/soap/envelope/"
2    SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"/>
```

```
3     <SOAP-ENV:Header>
4     </SOAP-ENV:Header>
5     <SOAP-ENV:Body>
6     </SOAP-ENV:Body>
7   </SOAP-ENV:Envelope>
```

As you can see, lines 3-4 define the Header section. Lines 5-6 define the Body. Other independent elements can optionally be defined inside the envelope, but for purposes of this discussion, we do not need to be concerned with independent elements. Refer to the W3C SOAP Specification at http://www.w3.org/TR/SOAP/ for more information about independent elements.

The Header section can contain *meta data* about the message. Meta data is "data that describes data". A SOAP message does not have to contain a Header. Header elements make it possible to extend the base SOAP protocol, to accommodate needs that the SOAP specification does not include.

For example, Header elements might maintain session information between a server and a client, or might contain authentication information about a transaction. A Header can contain any number of namespace-qualified child elements, each of which extends the default protocol in some way. Each header element provides extra content for processing the Body of the message.

Each Header element may be annotated with a "mustUnderstand" attribute, which indicates whether or not the element is mandatory. When "mustUnderstand" is True for an element, the server that processes the message must know how to interpret that element. If it doesn't, it must reject the message. Headers that do not have a "mustUnderstand" attribute, or which have this attribute set False, are considered to be optional, meaning that the recipient server is allowed to process the message as best it can.

Body Section

The most important part of a SOAP message is the Body section, since it contains the message's payload. In a *request* message, the Body defines the method to execute, and parameters that must be passed to it. The Body of a *response* message contains references to the method called, and return values from the method. If an error occurs, the response contains information about the fault. To better understand these concepts, let's look at some actual request/response messages. The request message in Listing 4 invokes a simple method that adds two numbers. Listing 5 contains the response from the web service.

Request Messages

Listing 4: Request to Invoke Add Method

```
1   <SOAP-ENV:Envelope
2           xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
3           SOAP-4
ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
5     <SOAP-ENV:Body>
```

```
6       <m:Add xmlns:m="uri:my-calculator">
7          <Parameter1>2</Parameter1>
8          <Parameter2>3</Parameter2>
9       </m:Add>
10   </SOAP-ENV:Body>
11 </SOAP-ENV:Envelope>
```

The **m:Add** method name element in line 6 contains the name of the method (Add) we wish to call, and the namespace it is found in (uri:my-calculator). The URI (Universal Resource Indicator) specifies which computer offers an Add method web service.

Lines 7-8 define two arguments (Parameter1 and Parameter2) that the Add method requires. In this example, the numbers to be added are 2 and 3.

Line 9 closes the method name element.

Response Messages

The response from the computer at **uri:my-calculator** is listed below. This response message contains return values from the Add method. By convention, "Response" is appended to the name of the method called. However, the format of the method name can also be defined using WSDL.

Listing 5: Response from the Add Method

```
1  <SOAP-ENV:Envelope
2          xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
3          SOAP-
ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"/>
4     <SOAP-ENV:Body>
5        <m:AddResponse xmlns:m="uri:my-calculator">
6           <Result>5</Result>
7        </m:AddResponse>
8     </SOAP-ENV:Body>
9  </SOAP-ENV:Envelope>
```

Line 5 identifies the remote procedure call. The Result tag in line 5 contains the sum of 2+3, which is 5. Note that the response message does not contain any of the data passed to call the function. Responses contain a return value from the function, or a fault message that indicates why the function call failed.

Fault Messages

When a message is rejected, the server generates a Fault, or error message. Faults are commonly caused by unrecognizable header fields, messages that cannot be authenticated, or problems that occurred when the server attempted to invoke a method or process the message.

Listing 6: A Typical Fault Response Message

```
<S:Envelope xmlns:S='http://schemas.xmlsoap.org/soap/envelope/'>
  <S:Body>
    <S:Fault>
       <faultcode>S:Server</faultcode>
       <faultstring>S:Server</faultstring>
       <detail>
         <e:mydetails xmlns:e="http://foo.com/detail">Some Error
Message</e:mydetails>
       </detail>
    </S:Fault>
  </S:Body>
</S:Envelope>
```

CIC's SOAP Components

CIC's SOAP Components

The SOAP components in a Customer Interaction Center environment are:

1. **Interaction Designer SOAP Components:** When Interaction Designer is installed, SOAP tools, SOAP Tool help, and a SOAP message trace utility are installed to the C:\I3\IC\Install\Admin\IC_Admin directory on the CIC server.

   SOAP tools are implemented in a dynamic link library named SOAPToolsIDA.DLL. When Interaction Designer starts up, it adds the tools defined in this DLL to Interaction Designer's tool palette. SOAP Tools are always installed with Interaction Designer. See SOAP Tools in Interaction Designer for more information. Context-sensitive online help for SOAP tools is available in Interaction Designer. Soap help topics are displayed when a SOAP tool has focus in Interaction Designer and the F1 key is pressed.

2. **SOAP Tracer Utility:** (SOAPTracerA.exe) is optionally installed with Interaction Designer when the "SOAP" option is selected. It permits users to "spy" on SOAP notification traffic. Soap Tracer displays request and response packets in a list and allows inspection of request and response payloads. For usage information, see SOAP Tracer Utility later in this section.

3. The **SOAP ISAPI Listener: Task** is responsible for parsing incoming SOAP requests, dispatching requests to the appropriate method, and packaging return values into outgoing SOAP responses. This process runs on an IIS Server. See SOAP ISAPI Listener Task for IIS later in this section.
4. **SOAP Notifier COM Objects** issue SOAP notifications from automation compatible applications. These components provide a high-performance method of initiating handlers without incurring the performance penalty of HTTP-based Listener operations. Third-party applications created using the SOAP Notifier COM components can directly create and forward packets to Interaction Processor, bypassing the need to create packets received using HTTP and the Soap Listener task. See SOAP Notifer COM Objects later in this section.

SOAP Tools in Interaction Designer

This topic summarizes SOAP-related tools in Interaction Designer that create handlers to process SOAP requests or responses. For additional information, see Appendix B: SOAP Tools.

Initiator Tools

SOAP Initiator

This initiator triggers if the "Notification Event" of the request matches a specified string. The Notification Event on which the Initiator triggers is specified in the property dialog.

Request Tools

SOAP Get Request Info

Queries some information from the request handle.

SOAP Abort Request

Aborts the request. Aborting a request is useful if a SOAP request handler is registered as a Monitor handler.

SOAP Get Transport Info

Returns an XML document containing transport specific (header) data. It allows the client to include any kind of out-of-band data in the request.

SOAP Expects Response

Takes a different exit path depending on whether the SOAP request requires a response (YES) or not (NO).

SOAP Parse Request Payload

Parses the payload of the request into an XML document.

SOAP Send Response

Sends the specified payload as response to the sender of the request. To support transport specific features, the "Transport Control Data" argument takes an XML node whose content will be sent back to the client. It can be used to send transport specific out-of-band data to the client.

Payload Processing Tools

SOAP Create Envelope

Creates a new SOAP envelope.

SOAP Get Body

Retrieves the Body element from the SOAP envelope. A body must exist. If it can't be found, the tool exits through "Failure" and attaches error information to the envelope.

SOAP Get Body Element

Retrieves the first body element that matches the given base name and namespace.

SOAP Add Body Element

Adds an entry to the body of the SOAP envelope.

SOAP Query Encoding Style

Matches a space separated list of URIs against the "encodingStyle" attribute of the given element. If the element doesn't have an ‘encodingStyle' attribute, the parent of the element is checked and so on, until an element with an "encodingStyle" attribute is found. If that attribute contains any of the specified encoding style URIs, the tool returns through "Found" and returns the style that was found.

SOAP Get Header

Retrieves the header element from the SOAP envelope if it has one.

SOAP Get Header Element

Retrieves the first header element that matches the given base name and namespace. Returns the first element in the header if neither a name nor namespace is given. Takes "Not Found" exit if the envelope doesn't have a header or the element can't be found.

SOAP Get Header Elements

Returns iterator to a list of header elements filtered by the given arguments. Takes the "None" exit if envelope has no header or none of the header elements matched the filter criteria.

SOAP Add Header Element

Creates a header element and adds it to the given envelope. If the envelope does not have a header, one is inserted before the Body element.

SOAP Get Fault

Retrieves fault information from the SOAP envelope. If there is no style="color: #0e5470;"><Fault> element in the envelope, the "No Fault" exit is taken and NULL elements and empty strings are returned.

SOAP Set Fault

Adds a style="color: #0e5470;"><Fault> element to the envelope or replaces an existing one.

SOAP Create Fault Response

Copies the request envelope and replaces all children of the style="color: #0e5470;"><Body> element with a single style="color: #0e5470;"><Fault> element. It combines the functionality of the "SOAP Create Envelope" tool with the functionality of the "SOAP Set Fault" tool.

SOAP Get RPC Parameter

This is a convenience tool for cracking RPC requests. It retrieves a parameter element (child) from the first element in the <Body> element (method in an RPC request). It returns the first element that matches all of the specified arguments.

SOAP Add RPC Parameter

This is a convenience tool for composing RPC requests or responses. It adds a parameter element to the first element in the body of the envelope, which represents the method in RPC requests. Use the XML tools to add complex data (not just a string) to the parameter by manipulating the returned "Parameter Element" node.

SOAP Get RPC Method Info

This is a convenience tool for cracking RPC requests. It retrieves the first child element of the SOAP <Body> element (Method element in RPC requests). It returns a collection containing the child elements of the method, which constitute the method arguments.

SOAP Get Next RPC Parameter

This tool returns the element node at the current iterator position and returns an iterator to the next position.

SOAP Create RPC Response

This is a convenience tool for composing the response envelope for an RPC request. It copies the source envelope and replaces the method element in the body with an element that has the same name but "Response" added to its name. It also adds a <Result> element as child of the method element.

SOAP Set Element Type

In SOAP, the type of an argument or the return value is specified by the service description and doesn't need to be included in the payload. However, the service may define the type as xsd:anyType, for VARIANT types. This tool allows you to include the type in the argument.

SOAP Create Array

Turns an element, for example an RPC parameter, into a SOAP array. The array is created for values supplied as list of strings or just a number of empty elements that can be populated with complex data.

Invocation Tools

SOAP HTTP Request

This tool issues an HTTP request to the specified URL with the SOAP request envelope as payload. The response body is parsed and returned as response envelope.

Helper Tools

SOAP Base64 Encode

Converts a supplied UNICODE string to the specified character set (default = UTF-8) and encodes the resulting data into a Base64 string. Characters that cannot be translated to the destination character set will be represented as ‘?'. Wide character sets, such as UTF-16 are currently not supported.

SOAP Base64 Decode

Decodes the base64 encoded string into the binary representation and converts it to UNICODE based on the specified character set.

SOAP Base64 Encode File

Reads the specified file as binary data and encodes it into a base64 string. This tool can be used to send any kind of data through SOAP requests. For example, you could encode a wave file in a SOAP message.

SOAP Base64 Decode To File

Decodes the base64 encoded string into the binary representation and writes the data to the specified file as binary data.

The SOAP Tracer Utility

The SOAP Tracer Utility

SOAP Tracer is used to debug SOAP requests and responses. It displays notifications exchanged between the client (SOAP Notifier COM or ISAPI listener) and the CIC server. It spies on SOAP notification messages. It records and displays request and response packets in a list and allows inspection of request and response payloads. Filtering for particular SOAP actions or clients is not supported in the current release, but may be added in the future. SOAP Tracer is optionally installed on the CIC server when Interaction Designer is installed—if the "SOAP" option is selected. It can be used from any machine that has access to a CIC server through a Notifier connection. However, SOAP Tracer is unrelated to Interaction Designer. It is also unrelated to ISAPI Listener.

IMPORTANT—Do not run SOAP Tracer for extended periods of time. It can consume a lot of memory and may degrade performance of CIC.

Starting SOAP Tracer

The default shortcut created under *Program Files* > PureConnect > *SOAP Tracer* is:

```
C:\I3\IC\Install\Admin\IC_Admin\SOAPTracerAD.exe /notifier=localhost
```

To run this utility, press the *Start* button, then select *Programs* > PureConnect > *Soap Tracer*. SOAP Tracer optionally accepts the command line arguments listed below.

Command Line Arguments

/NOTIFIER=<hostname>

Hostname or IP address of the Notifier server. Default: default Notifier server of the user.

/USER=<username>

User name of the CIC user. Default: current user

/PASSWORD=<password>

Password of the CIC user.

/TEMPDIR=<directory>

Directory in which to store the temporary files generated by the utility. Default is the system's TEMP directory.

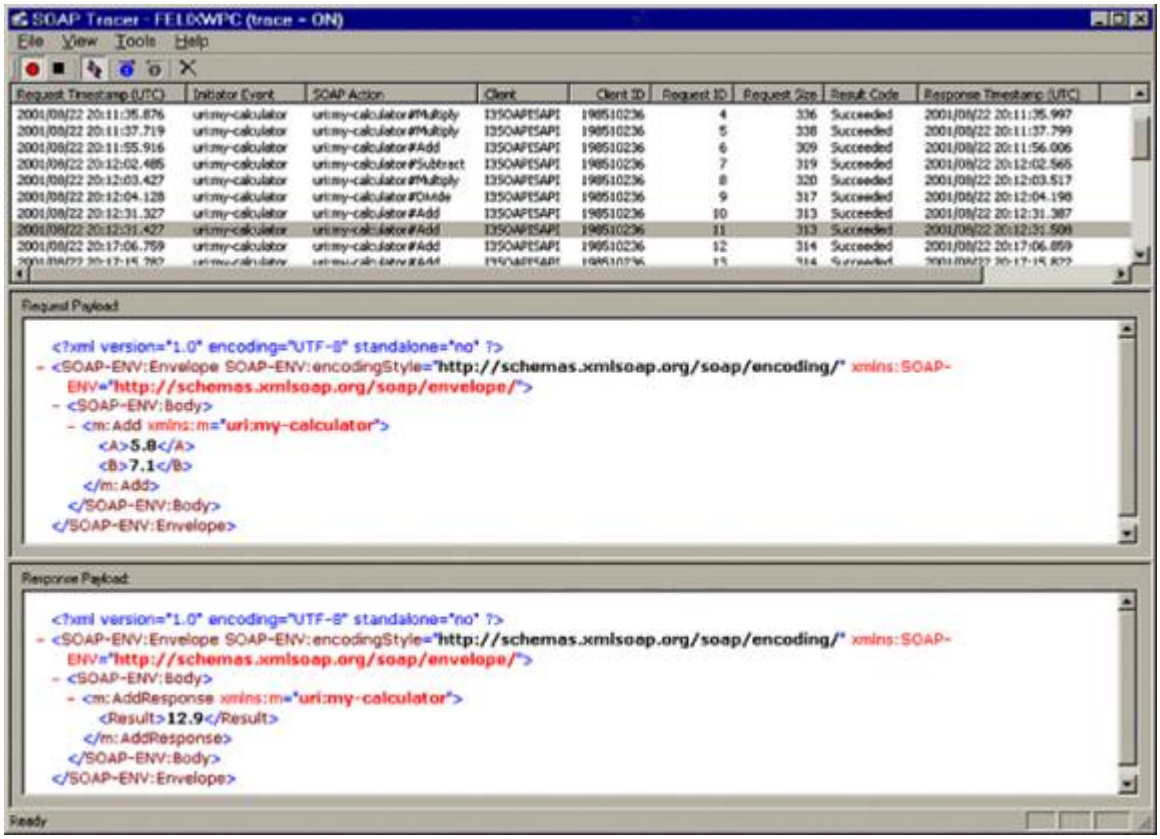
/STARTCAPTURE

If this argument is specified, the SOAP Tracer immediately starts capturing SOAP notifications. If not specified, the capture must be started by selecting Start Capture from the Tools menu, or by pressing the corresponding tool bar button for this command.

/KEEPTEMPFILES

By default, temporary files used to store the SOAP payload are deleted when the traces are cleared or the utility is exited. When this switch is specified, SOAP Tracer won't delete its temporary files automatically.

SOAP Tracer's User Interface



The SOAP Tracer window is divided into three panes. Users select a message in the top pane to display corresponding request and response messages in the other panes.

The Request List Pane

The top pane is the Request List. It displays information about SOAP requests. such as the name of the client who issued the request, the time, and whether or not the request succeeded. This pane contains the following columns:

Request Timestamp (UTC)

Date and time in UTC when request notification was recorded.

Initiator Event

Notification Event of the request (often same as SOAP Action).

SOAP Action

SOAP Action of the request.

Client

Name of the client issuing the request.

Client ID

Dynamic identifier of the client.

Request ID

Identifier of the request (generated by and scoped to client).

Request Size

Size of the request payload (SOAP envelope) in bytes.

Result Code

Result code returned by the server.

Succeeded

Request successfully processed

Failed

Request failed, server returned SOAP fault

Unhandled

Request was not handled by the server

Response Timestamp (UTC)

Date and time in UTC when response notification was recorded

Duration

Difference between Response Timestamp and Request Timestamp

Response Size

Size of the response payload data in bytes.

Key Term—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.

The Request Payload Pane

The Request Payload pane displays XML payload data that was sent to the handler for the selected request.

The Response Payload Pane

The Response Payload pane panes display payload data that was sent back to the client by the handler.

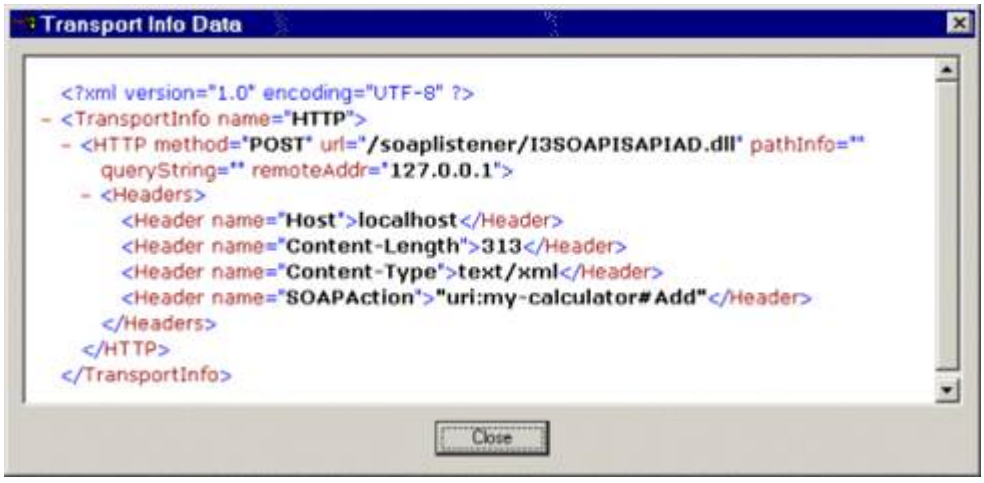
Menu Commands

File > Exit

Closes Soap Tracer.

View > Transport Info Data

Displays a dialog containing the transport info data of the request. This option is enabled if the SOAP request notification included transport information data.



View > Transport Control Data

Displays a dialog containing the transport control data of the response. This option is enabled if the SOAP request notification included transport control data.

View > Follow Requests

If checked, the selection in the request list will follow the recorded requests and always select the most recent one.

View > Toolbar

Hides or displays toolbar icons.

View > Status Bar

Hides or displays the status bar.

Tools > Start Capture

Starts recording the SOAP notification traffic.

Tools > Stop Capture

Stops recording the SOAP notification traffic.

Tools > Clear View

Clears the list of recorded SOAP notifications.

Tools > Settings

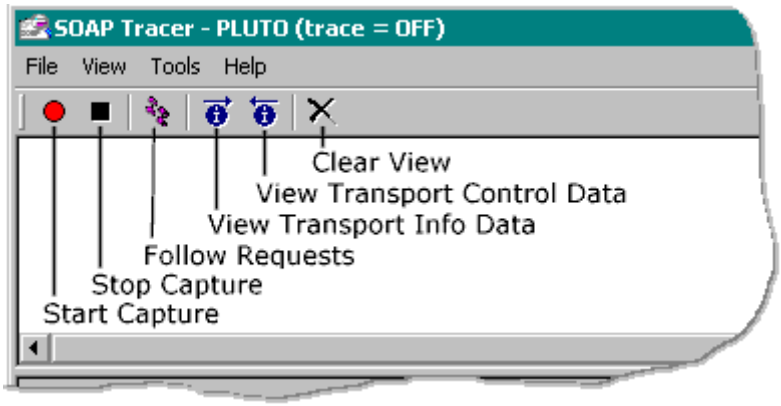
Displays dialog used to configure the application. This feature is disabled in the current release.

Help > About SOAPTracer…

Opens a dialog that displays copyright information.

Toolbar

Some SOAP Tracer commands have tool bar equivalents.



Tool bar icons in SOAP Tracer.

SOAP ISAPI Listener Task for IIS

SOAP ISAPI Listener translates HTTP SOAP packets into notifications and sends them to the CIC server. The SOAP ISAPI Listener must be installed on a machine that has IIS installed.

What is a Listener?

A *listener* receives incoming HTTP messages that contain SOAP requests for some type of service. It parses these messages, decides whether to process the request (based upon threshold values and filter configurations), and dispatches the request to the appropriate method for processing. If the service returns a response, the listener packages the response into an HTTP payload, and sends that back to the client. A listener also handles requests for WSDL information about web services.

The *SOAP ISAPI Listener* looks at incoming SOAP requests, decides whether requests should be forwarded to CIC to invoke a handler, and forwards appropriate requests to CIC's Notifier subsystem, which in turn calls Interaction Processor to invoke the handler associated with the initiator specified in the incoming message. SOAP ISAPI listener and packages return values from handlers into outgoing HTTP responses, and sends them to the client. If the listener decides not to forward a request to CIC for processing, it returns a fault message (SOAP and/or HTTP) to the requesting client application.

What is ISAPI?

The SOAP ISAPI Listener is sometimes called the *SOAP ISAPI DLL*, since it is a dynamic link library developed in conformance with Microsoft's Internet Server Application Programming Interface (ISAPI). ISAPI allows developers to extend the functionality of Microsoft's Internet Information Server (IIS). The component that implements the ISAPI Soap Listener task is I3SOAPISAPIU.DLL. This DLL is installed by the Interaction Center SOAP Listener Install to the IIS server of your choice. It translates HTTP requests into notifications and acts as a gatekeeper to prevent denial of service attacks. An ISAPI DLL is not a COM DLL. To invoke an ISAPI DLL, it must be explicitly referenced in a HTTP header. For example:

```
http://www.foo.com/virtual_directory_name/I3SOAPISAPIU.DLL
```

The virtual directory name is optional, so long as the server can resolve the location of the DLL.

What is an endpoint?

SOAP invokes methods at HTTP *endpoints*. An *endpoint* is a URL that uniquely identifies a namespace URI (Universal Resource Indicator), and the name of the method to execute (known as the NCName). Consider the following endpoint:

```
uri:my-calculator#Add"
```

The URI namespace (my-calculator) identifies the code module that contains the method to be called (Add), just as an interface name scopes a method in Java, CORBA, or COM. The namespace and the method name are separated by a pound sign.

When a SOAP request is transported to invoke the method, the endpoint name is passed in the SOAPMethodName header of the HTTP POST request. Consider the following sample HTTP header:

```
POST /objectURI HTTP/1.1

Host: www.foo.com

SOAPMethodName: urn:foo.com:my-calculator#Add

Content-Type: text/xml

Content-Length: nnnn
```

The HTTP header indicates that the **Add** method (from the **urn:foo.com:my-calculator** namespace) should be invoked against the endpoint identified by http://www.foo.com/objectURI.. The rest of the HTTP request is an XML document that contains additional information needed to invoke the request, such as parameters passed to the method. The server-side software that receives the request (e.g. the SOAP ISAPI Listener) is responsible for processing the request. Unlike other RPC protocols, SOAP doesn't define specific actions that must occur when a request is received. It leaves the implementation details to the process running at the endpoint. See http://www.w3.org/TR/REC-xml-names/

SOAP Notifier COM Objects

SOAP Notifier COM is a set of software components that allow custom applications to invoke handlers. SOAP Notifier COM objects issue SOAP notifications from automation compatible applications. Microsoft's .NET framework makes it possible for programmers to invoke a web service as if they were invoking a method of an object. SOAP Notifier COM components provide a high-performance method of calling handlers without incurring the performance penalty of HTTP-based Listener operations. Third-party applications created using the SOAP Notifier COM components can directly create and forward packets to Interaction Processor, bypassing the need to create packets received using HTTP and the Soap Listener task. These packets that are identical to those created by SOAP ISAPI Listener. However, the process is faster than HTTP-based Listener operations.

SOAP Notifier COM is appropriate for Windows client workstations that can run COM applications. It is not appropriate for operating systems (Linux, for example) that do not support COM. SOAP Notifier COM Components Setup registers SOAP Notifer COM API components on desktop PCs used to develop

or run SOAP Notifier COM API applications. Soap_Notifier_COM_API_DG.chm is the *SOAP Notifier COM API Developer's Guide*. It describes interfaces, methods, and properties of the SOAP Notifier COM API. You will find this publication in the System APIs section of the PureConnect Documentation Library.

ISoapConnector: the MSSOAP Notifier Connector

ProgId: ININ.MSSOAPNotifierConnector

The SOAP Notifier COM API provides a component named **ISoapConnector** that is used to initiate SOAP handlers. Programmers can invoke a web service as easily as invoking a method on an object. 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. Instead of the SoapClient, you may use the MSSOAPLib.SoapSerializer and MSSOAPLib.SoapReader objects with any object that uses a ISoapConnector.

```
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>...)
```

Properties

SOAP Notifier Connector supports the following 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).

Related Topics

Appendix C: Structure of IP Notification Messages

Install and Configure SOAP ISAPI Listener

Install and Configure SOAP ISAPI Listener

The components of SOAP follow the client/server model. Some components are installed when Interaction Designer is installed on the CIC Server. Other components are installed on IIS web servers and client PCs. This section explains how to install and configure SOAP Tools, SOAP Tracer, SOAP Listener, and Soap Notifier COM components.

- **SOAP Tools Installation**: When Interaction Designer is installed (as part of the CIC Admin setup), new SOAP tools are added to Interaction Designer's tool palette. SOAP tools are implemented in a DLL (SOAPToolsIDA.DLL) that is installed with Interaction Designer. Appendix B in this document also contains a summary of each SOAP Tool.
- **SOAP Tracer Installation**: The Soap Tracer Utility (SOAPTracerA.exe) is optionally installed if the "SOAP" option is selected during installation of Interaction Designer.
- **SOAP ISAPI Listener Installation**: The *Interaction Center SOAP Listener Install* installs the SOAP Listener Task on an IIS server. The SOAP Listener task is an ISAPI DLL. Installation requires preplanning on your part to address security and configuration issues, and some post-

installation work to customize the default SOAP filter configuration. The SOAP ISAPI DLL must be installed on a server running Microsoft Internet Information Server (IIS), version 5 or later.

Installation and configuration pre-planning

This section describes issues that SOAP implementers must resolve before installing ISAPI SOAP Listener on an IIS server. Security issues are particular important to consider if you plan to pass SOAP requests across the Internet.

1. **Select a server to host SOAP ISAPI Listener.**

   The SOAP Listener task is an ISAPI DLL that you must install on a computer running Microsoft's Internet Information Server (IIS) service. SOAP Listener uses IIS (version 5 or later) solely for HTTP operations. It does not consume other IIS services. You can install this task on a dedicated IIS server, or on a CIC server that is running IIS. Before choosing a platform, you should carefully consider security, performance, and capacity issues.

   SOAP Listener will work if it is installed on a CIC server running IIS. Theoretically, this could improve performance by eliminating latency between CIC and a dedicated IIS server. In practice, performance could be degraded if the CIC server becomes too highly tasked, and this configuration could compromise network security. As a rule of thumb, *do not* install SOAP ISAPI Listener on a CIC server unless:

   Port 80 HTTP traffic is tightly controlled (e.g. SOAP will be used exclusively for interactions between servers inside a firewall). This is appropriate for some corporate Intranets. Use a different port than 80 (e.g. 8080) that is blocked by the firewall. SOAP requests will not be received from the Internet, or use another port. The CIC server has the capacity to run IIS without degrading performance.

   If SOAP requests will be received from the Internet, you should install SOAP ISAPI Listener on an IIS server in a DMZ (Demilitarized Zone) between two firewalls. This can be an existing CIC/web server or a dedicated web server.

2. Open port 2633 on the firewall between the DMZ and the Intranet on which the CIC server is located, so that Notifier traffic can pass between the CIC server and the SOAP listener. Do not open port 2633 to the Internet.

What is a Demilitarized Zone?

In an Internet-connected world, any public access server, such as a web server that connects outside of an internal network is unprotected against hacking. A public access server can expose the rest of a network to potential intrusion.

Demilitarized zones (DMZ) reduce security risks by using multiple firewalls to delimit an internal network from publicly connected devices, such as web servers. A DMZ configuration protects both public servers and the internal network. The first firewall isolates essential Internet services (web, email, DNS, etc). The second firewall protects the internal network.

A DMZ is not the only solution that you might employ to protect your network. It is completely acceptable to use different security measures. The exact method is up to you—be reminded that if you connect a server to the outside world, you must manage the risk that your internal network might be penetrated through a public server.

3. **Apply service packs and hotfixes to IIS.**

Network security is a topic outside the scope of this paper. However, we strongly recommend that you keep server operating system and IIS software up-to-date. Apply Microsoft service packs and hot fixes regularly. Hackers frequently exploit known security holes that you can close by applying free software updates. You can automate this process to a limited extent. For example, Microsoft's HFNETCHK is an executable that runs on your server. It retrieves an XML file that contains information about security hot fixes that your system might need. Browse Microsoft's web site (http://www.microsoft.com/security) for security bulletins, upgrades and other information. As a rule of thumb, you should not install services that you do not need. Subscribe to "NTBugtraq" or a similar discussion list. This mailing list discusses security exploits and security bugs in Windows NT, Windows 2000, and Windows XP plus related applications. To sign up, visit http://www.ntbugtraq.com/.

4. **Decide how to configure SOAP Listener to prevent DoS Attacks**

Denial of Service (DoS) Attacks are attempts to flood a server with false requests for information, with the goal of overwhelming the system and ultimately crashing it. Not much can be done to prevent a denial of service attack. However, you can minimize the impact of DoS attacks by supplying the a couple of threshold values at installation time, and by customizing an ISAPI filter after installation is complete.

Default Request Timeout

Since DoS attacks can degrade performance of the CIC Server, ISAPI Listener can be configured (at installation time) to return a fault message (Server.RequestTimeout) if the CIC Server fails to respond within a specific time interval.

Before installing SOAP Listener, decide what value to enter into the *Default Request Timeout* field. This value sets the maximum amount of time in milliseconds that ISAPI Listener will wait for the CIC Server to respond to a SOAP request. When this interval is exceeded, ISAPI Listener

sends a fault message to the requester. The default is 60,000 milliseconds (1 minute). If your IIS server has a fast processor, and is dedicated to IIS, you may be able to reduce the default value.

This value sets the default timeout for all SOAPActions. Following installation, you can assign timeout values to specific SOAPActions, by editing a configuration file. For details, see Step 2: Set SOAPAction-Specific Timeout Values in the Post-Installation Procedures section of this document.

Maximum SOAP Payload Size

SOAP ISAPI Listener uses a threshold setting named *Maximum SOAP Payload Size* to limit the size of incoming SOAP messages. By default, the maximum SOAP payload Size is 128 KB. Larger messages are not forwarded by the Listener to the CIC Server for processing. Based upon the size of data passed to your handlers, you may be able to reduce this value significantly. This helps minimize the impact of denial of service (DoS) attacks.

Maximum Pending Requests

The *Maximum Pending Requests* threshold limits the maximum number of SOAP requests that the CIC server should process concurrently. It helps to think of this as the maximum number of pending *responses* that SOAP Listener will wait for at any given time, since SOAP Listener waits for a response to each request that it sends to CIC.

If Listener finds itself waiting for more responses that are allowed, it stops sending additional inbound request messages to the CIC Server until the number of pending requests falls below the threshold. SOAP ISAPI Listener does not queue unprocessed requests. It fails unprocessed requests with a fault message (Server.TooBusy).

Process Isolation Level

There is one last setting that you must consider before installing SOAP ISAPI Listener, and that is the level of process isolation (Low or High) that you wish to assign to the ISAPI Listener DLL. Process isolation protects the main IIS process against application faults—in this case, against potential failure of the ISAPI Listener DLL .

Process Isolation provides an additional layer of durability for your Web server. *Low* process isolation provides the best performance. *High* process isolation offers more protection against possible faults in the Listener application (unlikely). *Low* is the default.

Install SOAP Listener

If at this point, if you have IIS running with the latest service packs and hot fixes, behind an acceptable firewall configuration, and have formulated threshold values, you are ready to install SOAP Listener. This procedure explains how to run the ***SOAP Listener Setup*** to install, register, and configure the SOAP Listener task on an IIS server. The Soap Listener task is an ISAPI DLL that translates HTTP requests into

notifications. It acts as a gatekeeper to prevent denial of service attacks. Complete this procedure at your dedicated IIS Server or CIC Server running IIS. Installation requires pre-planning on your part to address security and configuration issues. If you have not read the Installation and configuration pre-planning section, we strongly recommend that you do so before performing this procedure.

1. Download the CIC 2018 R1 or later .iso file from the Genesys Product Information site at https://my.inin.com/products/Pages/Downloads.aspx.
2. Copy the .iso file to a file server (non-CIC server) with a high bandwidth connection to the server(s) on which you will be running the CIC 2018 R1 or later installs.
3. Mount the .iso file and share the contents to make them accessible to the server(s) on which you will be running the CIC 2018 R1 or later installs.
4. Navigate to the \Installs\Off-ServerComponents directory on the file server.
5. Copy the SOAP Listener .msi file, for example, SOAPListener_2018_R1.msi, to the server on which you plan to run this install and double-click to launch it.

   The welcome page appears.

6. Press Next to proceed past the welcome screen. Then press Next a second time to accept installation of default features.
7. Supply user name, domain password, and domain for a user account with administrative privileges on the CIC server. Then press **Next**.
8. Type the name of the CIC server. Then press **Next**.
9. Supply values as indicated below:

   Default Request Timeout (in seconds)

   Enter the number of seconds that the ISAPI Listener should wait for the CIC Server to respond (to a SOAP request) before timing out and returning a fault message. The default value is 0 seconds. Press **Next** to proceed.

   Maximum Pending Requests

   Specify the maximum number of SOAP requests that your CIC server should handle concurrently during peak periods. This helps protect your server from denial of service (DoS) attacks. When this value is exceeded, additional requests will be denied.

   Maximum SOAP Payload Size (in KB)

   Specify the maximum size (in kilobytes) of SOAP payloads sent by the SOAP Listener to the CIC Server. Larger XML payloads will not be forwarded, to minimize the risk of denial of service (DoS) attacks.

10. Press **Next** to proceed. The next screen prompts for a location where log files will be stored. Accept the default path, or navigate to a different path. When you are finished, click Next.
11. Click **Install** to begin installing files.
12. Press **Finish** to exit Setup.
13. Click **Yes** to restart.

14. For the SOAP Listener machine to receive updates from the Interactive Update Provider on the CIC Server, you must run the *Interactive Update Client install* following the *SOAP Listener* install. The install will prompt for the Interactive Update Provider Server (CIC Server) name or IP address.

Post-installation procedures

Following installation of ISAPI SOAP Listener, you should complete additional security steps to defend against DoS Attack. Specifically, you should limit requests to known SOAPActions, and to assign timeout values to individual SOAPActions. You will modify the default ISAPI filter configuration file. The relative path to this file is ..\soaplistener\filter\I3SOAPISAPIConfig.xml. The SOAP ISAPI endpoint listener uses I3SOAPISAPIConfig.xml to filter incoming message requests. This file acts as a gatekeeper. It affects whether or not incoming messages are forwarded to the CIC Server by ISAPI Listener. Implementers are strongly encouraged to edit I3SOAPISAPIConfig.xml immediately after SOAP ISAPI Listener is installed, and whenever new handlers implement an additional SOAPAction.

The default configuration indiscriminately forwards all SOAP requests to the Interaction Center server identified in the ISAPI Listener install. You should modify the filter file to make the following modifications:

1. Add <Rule> elements that identify the specific operations (SOAPActions) that your CIC server should process. Thereafter, SOAP ISAPI Listener will forward only those particular SOAPActions to the CIC server.
2. Set timeout thresholds for specific SOAPActions used in your environment.

These modifications are particularly important if your SOAP Listener is exposed to the Internet. If you leave the default filter unchanged, your CIC server is more venerable to DoS attacks. Before we discuss the modification procedure steps in detail, it is necessary to introduce the format of the configuration file.

I3SOAPISAPIConfig.xml Filter File Format

The ISAPI filter is just an XML file whose structure can be described as follows. Its root element, <FilterConfig> has three child elements, <ICServers>, <Defaults> and <Rules>.

<ICServers>

The <ICServers> element contains a list of Interaction Center Servers to which to route the messages. <ICServers> can have <ICServer> and <ICServer2> child elements.

<ICServer2> Uses a remote subsystem connection. GenSSLCerts must be run prior to attempting to connect to a notifier with this type of connection. In a switchover situation, use <ICServer>. <ICServer2> will not work correctly in switchover environments.

The attributes of the <ICServer> child element are:

name

The name of the CIC server, used to identify the server in filter rules.

host

Hostname or IP address of the Notifier (CIC) server.

username

Login name for the Notifier connection.

password

Password for the Notifier session.

The attributes of the <ICServer2> child element are:

name

Name of the server (used to identify it in rule action).

host

Hostname or IP address of the Notifier server.

<Defaults>

The <Defaults> element stipulates default rule actions. It has two child elements. <ForwardRequest> identifies requests that will be forwarded. <HTTPResponse> identifies requests to be rejected.

The attributes of the <ForwardRequest> child element are:

server

Name of the Interaction Center Server configured through the corresponding <ICServer> tag. This attribute is (case-sensitively) matched against the name attributes of the <ICServer> tags.

initiatorEvent

Name of the InitiatorEvent (notification event) as which the request should be forwarded to the Interaction Center server. If not explicitly specified or an empty string, the soapAction from the HTTP header will be used.

soapAction

SOAPAction string to be forwarded to IP. If not defined or "*", use same action that matched the rule.

clientName

Client name value specified in the request notification. Default = "I3SOAPISAPI". This is mainly informational for use as a trace message.

requestTimeout

Timeout value used for the request. Default as specified by ‘DefaultRequestTimeout' registry key. Time in milliseconds

includeTransportInfo

Specifies whether to include the TransportInfo data in the request sent to IP. Possible values: "1", "0", "true", "false". Default = "1".

The attributes of the <HTTPResponse> child element are:

statusCode

HTTP status code. Default = "500".

statusText

HTTP status text. Default = lookup based on statusCode (for "500": "Internal Server Error").

soapFaultcode

Value of the <faultcode> element in the <Fault> element of the response sent back to the client. Default = "Server.SOAPAction".

soapFaultstring

Value of the <faultstring> element in the <Fault> element of the response sent back to the client.. Default = "The SOAPAction is not recognized by the server!"

<Rules>

The <Rules> element contain <Rule> child elements which define the action to be performed when the rule fires. That happens when the request's SOAPAction matches the rule's soapAction attribute. The <Rule>child element has only one attribute:

soapAction

SOAPAction that triggers this rule. SOAPAction matching is case-sensitive.

Sample I3SOAPISAPIConfig File

This sample filter listed below shows how the elements fit together. The numbers are for illustration purposes and do not appear in an actual configuration file. See SOAP ISAPI Filter Schema for the schema used by I3SOAPISAPIConfig.xml.

```
<FilterConfig xmlns="urn:schemas-inin-com:soapisapi-filter-config">
  <ICServers>
     <ICServer name="localhost"
        host="localhost"
        userName=""
        password=""/>
     <ICServer name="mars"
        host="mars"
        userName="eic_admin"
        password="i3"/>
  </ICServers>
  <Defaults>
     <ForwardRequest server="localhost"
        clientName="I3SOAPISAPI"
        requestTimeout="20000"
        includeTransportInfo="1"/>
        <HTTPResponse statusCode="500"
           statusText="Internal Server Error"
           soapFaultcode="Client.SOAPAction"
           soapFaultstring="The specified method is not supported!"/>
   </Defaults>
  <Rules>
     <Rule soapAction="uri:my-calculator#Add">
        <ForwardRequest initatorEvent="uri:my-calculator"/>
     </Rule>
     <Rule soapAction="uri:my-calculator#Subtract">
        <ForwardRequest initatorEvent="uri:my-calculator"/>
      </Rule>
     <Rule soapAction="uri:my-calculator#Multiply">
        <ForwardRequest initatorEvent="uri:my-calculator"/>
```

```
31      </Rule>
32      <Rule soapAction="uri:my-calculator#Divide">
33         <ForwardRequest initatorEvent="uri:my-calculator"/>
34      </Rule>
35      <Rule soapAction="uri:test#foo">
36         <ForwardRequest server="mars"
37            soapAction="uri:test#bar"
38            requestTimeout="120000"/>
39      </Rule>
40      <Rule>
41         <HTTPResponse/>
42      </Rule>
43   </Rules>
44 </FilterConfig>
```

This sample specifies several SOAPActions that refer to a calculator service. On line 23, the SOAPActions of the calculator are forwarded with the "uri:my-calculator" InitiatorEvent, so all requests trigger the same initiator. All other attributes of that rule are inherited from the default <ForwardRequest> element (line 13). Accordingly, requests for my-calculator are sent to the "localhost" server, even though that was not explicitly defined in the rule. It is easy to specify attributes in a Rule element that override default elements. In line 35, the SOAPAction "uri:test#foo" is forwarded as "uri:test#bar" (both the SOAPAction and InitiatorEvent) to the server "mars". The request timeout for this particular request is set to 2 minutes (120,000 milliseconds).

The last rule simply rejects all other SOAPActions with the default <HTTPResponse> rule action. To forward all SOAPActions indiscriminately, the following rule could be used:

```
<Rule>
  <ForwardRequest/>
</Rule>
```

Wildcard Pattern Matching

Currently, we do not support regular expression patterns as the soapAction attribute of a rule, although that may be added in a future release. However, to simplify filters for objects with many methods, a simple wildcard pattern is supported: The soapAction value may end with an asterisk (*), which means that the SOAPAction many be followed by one or more characters, that are ignored in the match. The * wildcard is supported only if it is the last character in a soapAction attribute. For example, this technique could be used to replace all rules for the calculator with a single one, where soapAction attribute has a value of " uri:my-calculator#*". Implement wildcards with care, or not at all, since this opens the possibility for DoS attacks on the Notifier event-ID caches. We thus strongly suggest explicitly adding rules for each SOAPAction that is to be forwarded to the server.

Forward only supported SOAPActions to CIC

1. Customize the ISAPI filter file to prevent the SOAP Listener task from indiscriminately forwarding all SOAP requests to the Interaction Center Server. Filtering ensures that the CIC Server receives only those requests that match supported SOAPActions.
2. Set SOAPAction Timeout Values. You can optionally modify this file to assign SOAPAction-specific timeout values, by adding requestTimeout attributes to ForwardRequest elements. The example below shows how to set the timeout value for a SOAPAction named "bar" to 120 seconds.

   <ForwardRequest server="mars" …identifies the CIC server

   soapAction="uri:test#bar" …identifies which SOAPAction

   requestTimeout="120000"/> …action-specific timeout value in milliseconds

3. Unload the SOAP ISAPI DLL. To put a modified filter configuration into effect, you must unload the ISAPI DLL. The DLL will reload automatically the next time that a SOAP request is received.
   1. From the desktop of your IIS server, press the *Start* button. Select *Settings*, then *Control Panel*.
   2. Double-click the *Administrative Tools* folder to open it.
   3. Double-click the icon titled *Internet Services Manager*.
   4. Right-click the name of your virtual directory. Then select *Properties*.
   5. Select the *Virtual Directory* tab. Then press the *Unload* button.
   6. Press OK to close the active dialog.
   7. Close the *Internet Services Manager* window. Changes made to the SOAP filter configuration will take effect the next time that a request is received.

SOAP ISAPI Filter Schema

The ISAPI Filter Configuration file conforms to the following schema:

```
<?xml version="1.0"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
   xmlns:tns="urn:schemas-inin-com:soapisapi-filter-config"
   targetNamespace="urn:schemas-inin-com:soapisapi-filter-config"
   elementFormDefault="qualified"
   attributeFormDefault="unqualified">
   <xsd:element name="FilterConfig" type="tns:FilterConfig"/>
   <xsd:complexType name="FilterConfig">
     <xsd:sequence>
        <xsd:element name="ICServers" type="tns:ICServers" minOccurs="0"/>
```

```
        <xsd:element name="Defaults" type="tns:Defaults" minOccurs="0"/>
        <xsd:element name="Rules" type="tns:Rules" minOccurs="0"/>
      </xsd:sequence>
    </xsd:complexType>
    <xsd:complexType name="ICServers">
      <xsd:element name="ICServer" type="tns:ICServer" minOccurs="0"
        maxOccurs="unbounded"/>
    </xsd:complexType>
    <xsd:complexType name="ICServer">
      <xsd:attribute name="name" type="xsd:string" use="required"/>
      <xsd:attribute name="host" type="xsd:string" use="required"/>
      <xsd:attribute name="userName" type="xsd:int" use="optional"/>
      <xsd:attribute name="password" type="xsd:boolean" use="optional"/>
    </xsd:complexType>
    <xsd:complexType name="Defaults">
      <xsd:sequence>
        <xsd:element name="ForwardRequest" type="tns:ForwardRequest"
minOccurs="0"/>
        <xsd:element name="HTTPResponse" type="tns:HTTPResponse"
minOccurs="0"/>
      </xsd:sequence>
    </xsd:complexType>
    <xsd:complexType name="Rules">
      <xsd:element name="Rule" type="tns:Rule" minOccurs="0"
maxOccurs="unbounded"/>
    </xsd:complexType>
    <xsd:complexType name="Rule">
      <xsd:choice minOccurs="0">
        <xsd:element name="ForwardRequest" type="tns:ForwardRequest"/>
        <xsd:element name="HTTPResponse" type="tns:HTTPResponse"/>
      </xsd:choice>
      <xsd:attribute name="soapAction" type="xsd:string" use="optional"/>
    </xsd:complexType>
    <xsd:complexType name="ForwardRequest">
      <xsd:attribute name="server" type="xsd:string" use="optional"/>
      <xsd:attribute name="initatorEvent" type="xsd:string" use="optional"/>
```

```
      <xsd:attribute name="soapAction" type="xsd:string" use="optional"/>
      <xsd:attribute name="clientName" type="xsd:string" use="optional"/>
      <xsd:attribute name="requestTimeout" type="xsd:int" use="optional"/>
      <xsd:attribute name="includeTransportInfo" type="xsd:boolean"
use="optional"/>
   </xsd:complexType>
   <xsd:complexType name="HTTPResponse">
      <xsd:attribute name="statusCode" type="xsd:positiveInteger"
use="optional"/>
      <xsd:attribute name="statusText" type="xsd:string" use="optional"/>
      <xsd:attribute name="soapFaultcode" type="xsd:QName" use="optional"/>
      <xsd:attribute name="soapFaultstring" type="xsd:string" use="optional"/>
   </xsd:complexType></xsd:schema>
```

Reinstall/Uninstall SOAP Listener

If you run the *SOAP Listener Install* a second time, it provides the opportunity to change the way features are installed, repair installation errors, or remove SOAP Listener from your computer.

1. Insert your CIC installation DVD (or mount an ISO image). In many cases, the user interface application will start automatically. If it does not appear, run **autorun.exe** from the root directory.
2. Click the **Optional Installs (2)** button.
3. Click **CIC SOAP Listener**.
4. Click **Next** to dismiss the Welcome screen.
5. Click **Change**, **Repair**, or **Remove**.

Install SOAP Notifier COM

Install SOAP Notifier COM

SOAP Notifier COM objects issue SOAP notifications from automation compatible applications. SOAP Notifier COM components provide a high-performance method of initiating handlers without incurring the performance penalty of HTTP-based Listener operations.

Third-party applications created using the *SOAP Notifier COM API* directly create and forward packets to Interaction Processor, bypassing the need to create packets received using HTTP and the Soap Listener task.

What: Run **CC *SOAP Notifier COM Components Setup*** to install and register components needed to run or develop third-party SOAPNotifierCOM applications on a desktop PC. Components are installed to the destination folder specified by the user. The default folder is c:\Program Files\Interactive Intelligence. Setup registers two dynamic link libraries: SOAPNotifierCOMU.DLL and MSSOAPNotifierConnectorU.DLL. Setup optionally installs a help system that describes interfaces, methods, and properties in the Notifier

COM API. When this option is selected, setup adds a shortcut named *SOAP Notifier COM Help* to the start menu, inside the *Interactive Intelligence* folder.

Where: Install these components on any PC used to develop or run SOAP Notifier applications.

Prerequisite: The desktop PC must be running a version of Windows that supports the Component Object Model. SOAP Notifier COM API is not compatible with operating systems that do not support COM (Linux, for example).

Steps to Complete

1. Download the CIC 2018 R1 or later .iso file from the Genesys Product Information site at https://my.inin.com/products/Pages/Downloads.aspx.
2. Copy the .iso file to a file server (non-CIC server) with a high bandwidth connection to the server(s) on which you will be running the CIC 2018 R1 or later installs.
3. Mount the .iso file and share the contents to make them accessible to the server(s) on which you will be running the CIC 2018 R1 or later installs.
4. Navigate to the \Installs\Off-ServerComponents directory on the file server.
5. Copy the SOAP Notifier COM .msi file, for example, SOAPCOM_2018_R1.msi, to the server on which you plan to run this install and double-click to launch it.
6. If prompted whether to run the install program, respond **Run**.
7. Press **Next** to dismiss the welcome screen.
8. Press **Next** to accept all default features.
9. Press *Install* to begin installation.
10. Wait while files are copied.
11. Press **Finish** to exit Setup.

Reinstall/Uninstall SOAP Notifier COM Components

If you run *SOAP Notifier COM Components Setup* a second time, it provides the opportunity to modify the way features in installed, to repair installation errors, or to remove SOAP Notifier COM components from your computer.

1. Click **Next** to proceed past the startup screen.
2. Then select **Change**, **Repair**, or **Remove**.

Appendix A: SOAP Transport Information and Control

Appendix A: 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>
```

HTTP Transport

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>
```

Request (Transport Info)

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

This sample Transport Info structure adheres to 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)

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>
```

Tip—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").

Appendix B: SOAP Tools

Appendix B: SOAP Tools

This appendix provides information about the tools in Interaction Designer that process SOAP requests and responses. SOAP tools are late-bound, meaning that the structure of data processed by a SOAP handler does not have to be specified at compile time when the handler is published. SOAP tool steps can be added to any handler, to create and send SOAP requests to any server that understands SOAP. SOAP Tools do not support calls to an SSL server. In CIC 2.3 and later, the assumed namespace prefix is SOAP, rather than SOAP-ENV, for compatibility with Microsoft .NET. These tools are also documented in the Interaction Designer help.

There are 5 categories of SOAP tools:

- Initiator Tools
- Request Tools
- Payload Processing Tools
- Invocation Tools
- Helper Tools

Initiator Tools

Initiator Tools

SOAP Initiator

This initiator triggers if the ‘Notification Event' of the request matches a specified string. The Notification Event on which the Initiator triggers is specified in the property dialog.

Parameter	Dir	Type	Remarks
SOAP Request	OUT	Handle	Handle representing the SOAP request. It can subsequently be used to query additional information from the (HTTP) header.
Initiator Event	OUT	String	String of the notification event that triggered the initiator.
SOAP Action	OUT	String	SOAP Action of the request that triggered the handler.

Request Tools

Request Tools

SOAP Get Request Info

Queries some information from the request handle. Exit Paths: Success, Failure.

Parameter	Dir	Type	Remarks
SOAP Request	IN	Handle	Handle of the SOAP request.
Initiator Event	OUT	String	Notification event that caused the initiator to trigger.
SOAP Action	OUT	String	SOAP action code of that request.
Client ID	OUT	Integer	Client ID (Notifier object id).
Client Name	OUT	String	Name of the client.
Request ID	OUT	Integer	Request ID (for debugging/tracing purposes).
Payload Size	OUT	Integer	Size of the request payload in bytes.
Transport Info Size	OUT	Integer	Size of the transport information in bytes.

SOAP Abort Request

Aborts the request. If ‘Send Unhandled Response' is False, it does not send a response notification, not even an "Unhandled" response when the Request handle goes out of scope. Aborting a request is useful if a SOAP request handler is registered as Monitor handler, for example for wildcard SOAPAction. Multiple handlers may fire at the same time, but only one must send a response notification to the client.

Exit Paths: Success, Failure

Parameter	Dir	Type	Remarks
SOAP Request	IN	Handle	Handle of the SOAP request
Send Unhandled Response	IN	Boolean	Checkbox (default = False)

SOAP Get Transport Info

Returns an XML document containing transport specific (header) data. It allows the client to include any kind of out-of-band data in the request. For example, for HTTP requests, this document contains the HTTP method and a list of the header elements.

Tip—The data may be parsed every time the tool is invoked or cached. This may depend on the specified selection namespaces. The returned document is read-only.

See SOAP ISAPI Filter Schema for schema details. If there is no transport information data, an empty document is returned and the tool takes the ‘No Info' exit. If there is an error (Failure), an empty document is returned which can be queried with ‘XML Get Error Info'.

Exit Paths: Success, No Info, and Failure

Parameter	Dir	Type	Remarks
SOAP Request	IN	Handle	Handle of the SOAP request
Selection Namespaces	IN	String	Optional. Space delimited list of namespace declarations to be set as selection namespaces for the XPath queries.
Preserve Whitespace	IN	Boolean	Checkbox: False Default. Nonessential white space is ignored when parsing the payload. True Preserve nonessential white space.
Validate On Parse	IN	Boolean	Checkbox: False Default. Only verifies for well-formedness.

			True Validates against the schema during parse.
Resolve Externals	IN	Boolean	Checkbox: False Default. Do not resolve resolvable namespaces. True Resolve resolvable externals (namespaces, DTDs, entity references etc.) at parse time.
Transport Info	OUT	Node	Read-only. XML document containing transport-specific out-of-band information. Empty document if no transport information. See Appendix A: SOAP Transport Information and Control.

SOAP Expects Response

Takes a different exit path depending on whether the SOAP request requires a response (YES) or not (NO). If the request expects a response and the handler exits (the SOAP Request handle goes out of scope) without having invoked 'SOAP Send Response', a Response Notification is sent back with the 'Unhandled' flag set to true.

Exit Paths: YES, NO

Parameter	Dir	Type	Remarks
SOAP Request	IN	Handle	Handle of the SOAP request

SOAP Parse Request Payload

Parses the payload of the request into an XML document. If the 'Validate SOAPAction' parameter is True, the tool checks the SOAPAction field of the request against the payload. The payload data is parsed every time this tool is invoked (i.e. it is not cached). The document is furthermore *not* read-only and thus may be modified as needed, for example to create the response. The payload envelope node will still be returned, even if the SOAP Action does not match.

Heuristic

This tool uses a heuristic to match the action code (legend: *<NS>* = namespace of the first body element; *<MethodName>* = Name of the element [method name]):

```
<NS>
<NS> [<AnyCharacter>] <MethodName>
<MethodName>
```

[**<AnyCharacter>**] **<MethodName>**

This will catch actions such as "uri:my-uri#MyMethod", "http://soap.inin.com/e-faq", "MyMethod" etc. An empty SOAPAction matches all methods.

Parameter	Dir	Type	Remarks
SOAP Request	IN	Handle	Handle of the SOAP request
Validate SOAPAction	IN	Boolean	Checkbox: False Don't verify SOAPAction header field against payload. True Default. Check SOAPAction header field against method namespace and name.
Action Validation Mask	IN	String	Optional. Mask for validation of SOAP Action. Note: this is a future extension that has not been defined.
Selection Namespaces	IN	String	Optional. Space delimited list of namespace declarations to be set as selection namespaces the XPath queries. If this argument not specified, just "SOAP-ENV" is mapped to the envelope namespace. NOTE: The "SOAP-ENV" prefix will be used irrespective of the actual prefix in the payload. A declaration mapping "SOAP-ENV" to the envelope namespace will always be added to the declarations, unless SOAP-ENV is already declared in the argument.
Preserve Whitespace	IN	Boolean	Checkbox: False Default. Nonessential whitespace is ignored when parsing the payload. True Preserve nonessential white space
Validate On Parse	IN	Boolean	Checkbox: False Default. Only verifies for well-formedness. True Validates against the schema during parse.
Resolve Externals	IN	Boolean	Checkbox: False Default. Do not resolve resolvable namespaces. True Resolve resolvable externals (namespaces, DTDs, entity references etc.) at parse time.

Payload	OUT	Node	XML document with Envelope as document element. If there is an error, the document may be empty (but not NULL), and the 'XML Get Error Info' tool can be used to retrieve information about what failed).

Exit Paths

Success

Payload successfully parsed. SOAP Action matches.

Empty Payload

SOAP Payload is empty (XML document has no document element).

Wrong Action

SOAP Action validation enabled and action doesn't match.

Parse Error

A parse error occurred parsing the payload. Use 'XML Get Error Info'.

Failure

Some other failure. Use 'XML Get Error Info'.

SOAP Send Response

Sends the specified payload as response to the sender of the request. To support transport specific features, the 'Transport Control Data' argument takes an XML node whose content will be sent back to the client. It can be used to send transport specific out-of-band data to the client. For example, for the HTTP transport it allows to set additional header fields or specify a special status code. See SOAP ISAPI Filter Schema for schema details. The schema itself is not part of SOAP specification.

Parameter	Dir	Type	Remarks
SOAP Request	IN	Handle	Handle of the SOAP request
Payload	IN	Node	Node of the payload envelope to send back to the client. Must be document node or <Envelope> document element.
Transport Control Data	IN	Node	Optional. Node of an XML structure with additional transport specific control data. See Appendix A: SOAP Transport Information and Control.

Exit Paths

Success

Response was sent successfully.

No Response

This request does not expect a response.

Duplicate

Response for this request has already been sent.

Failure

Some other error. Check Payload node with XML Get Error Info.

Payload Processing Tools

Payload Processing Tools

SOAP Create Envelope

Creates a new SOAP envelope. To simplify composing RPC requests, where the first child element of the <Body> element is the method to invoke, the 'RPC Method Name' and 'RPC Method Namespace' argument can be used as shortcut. The same can be achieved by invoking 'SOAP Add Body Element' after creating the envelope. Therefore, this tool creates the following XML document:

```
<?xml version="1.0" encoding="{XML Encoding}" ?></{RPC Method Name}>]
   </{Envelope Prefix}:Body>
</{Envelope Prefix}:Envelope>
```

The 'Declare Namespaces' argument is used to declare namespaces in the envelope that will be used in other elements, such as the **xsd** or **xsi** prefixes for typed arguments. It keeps the size of the envelope low, as otherwise each element that uses a prefix will contain **xmlns** attributes. If the 'RPC Method Name' argument has no namespace prefix and an 'RPC Method Namespace' different than **""** (default namespace) is specified, a prefix will be synthesized, unless the local name starts with a '**:**' (which is illegal in XML, but signals to this tool *not* to add a synthesized namespace prefix). Adding a prefix can greatly reduce the size of the message if child elements are in no namespace (usually parameters are in the default namespace), as otherwise each child element would get a **xmlns=""** attribute.

Exit Paths: Success, Failure

Parameter	Dir.	Type	Remarks
XML Encoding	IN	String	Optional. Character encoding to be used for the XML document. If omitted, "UTF-8" is used. See remarks.
Envelope Prefix	IN	String	Optional. Namespace prefix for the envelope namespace. If not specified the default "SOAP-ENV" is used.
Encoding Style	IN	String	Optional. Space separated list of namespaces specifying the encoding style (value of the 'encodingStyle' attribute). If not

			specified or "STANDARD" is passed as string, "http://schemas.xmlsoap.org/soap/encoding/" is used. The encodingStyle attribute is omitted if "NONE" is specified.
RPC Method Name	IN	String	Optional. Fully qualified name of the method element (first child element of the body element). If not specified, no method element will be added. Please consult Remarks for additional details!
RPC Method Namespace	IN	String	Optional. Namespace of the method element.
Declare Namespaces	IN	String	Space delimited list of namespace declarations of the form xmlns:{prefix}='{URI}' to be declared in the envelope. See remarks.
Selection Namespaces	IN	String	Optional. Space delimited list of namespace declarations to be set as selection namespaces for the XPath queries. If argument not specified, the envelope prefix and the 'Declare Namespace' namespaces will be set as selection namespaces. NOTE: mapping for envelope prefix will always be added.
Envelope	OUT	Node	XML document with Envelope as document element.

SOAP Get Body

Retrieves the Body element from the SOAP envelope. A body must exist and if it can't be found, the tool exits through 'Failure' and attaches error information to the envelope.

Exit Paths: Success, Failure

Parameter	Dir	Type	Remarks
Envelope	IN	Node	Envelope node of the SOAP payload. Can be a document node whose document element is <SOAP-ENV:Envelope> or the node is the element itself.
Body	OUT	Node	Node of the <SOAP-ENV:Body> element.

SOAP Get Body Element

Retrieves the first body element that matches the given base name and namespace. If no namespace is specified, the first element matching 'Base Name' is returned. Returns the first element in the body if neither a name nor namespace is given.

Exit Paths: Success, Not Found, Failure

Parameter	Dir	Type	Remarks
Envelope	IN	Node	Envelope node of the SOAP payload. Can be a document node whose document element is <SOAP-ENV:Envelope> or the node is the element itself.
Base Name	IN	String	Optional. Base name of the element to return. If no name given, the first entry in the body in 'Namespace' is returned. This corresponds to the element of the method for RPC requests.
Namespace	IN	String	Optional. Namespace of the element to return
Retrieve Value	IN	Boolean	Checkbox: False Default. Do not retrieve value True Return node value
Body Element	OUT	Node	Child element of the <SOAP-ENV:Body> element that has the given base name and namespace. NULL node if the element is not in the body.
Element Base Name	OUT	String	Base name of the returned element
Element Namespace	OUT	String	Namespace URI of the returned element
Value	OUT	String	Value of the body element (if 'Retrieve Value' = True)

SOAP Add Body Element

Adds an entry to the body of the SOAP envelope. Use the XML tools on the returned 'Element' node to add rich contents to the element (not just a string).

Tip—If the 'Name' argument has no namespace prefix and a 'Namespace' different than **""** (default namespace) is specified, a prefix will be synthesized, unless the local name starts with a '**:**' (which is illegal in XML, and thus signals to this tool *not* to add a synthesized namespace prefix). Adding a prefix can greatly reduce the size of the message if child elements are in no namespace, as otherwise each child element would get an **xmlns=""** attribute.

Exit Paths: Success, Failure

Parameter	Dir	Type	Remarks
Envelope	IN	Node	Envelope node of the SOAP payload. Can be a document node whose document element is <SOAP-ENV:Envelope> or the node is the element itself.

Name	IN	String	Fully qualified name of the element to create and add to the body.
Namespace	IN	String	Optional. Namespace URI of the element. If the parameter is omitted and the name has a namespace prefix, the tool will search in the parent elements for the namespace with the same prefix and make the element a member of this namespace.
Encoding Style	IN	String	Optional. Value of the ‘encodingStyle' attribute. Attribute is omitted if not specified or "NONE". Specify "STANDARD" for standard namespace ("http://schemas.xmlsoap.org/soap/encoding/").
Value	IN	String	Optional. String value to set as content of the element.
Replace Existing Body Element	IN	Boolean	Checkbox: False Default. Add the element as last child of the body. True Replace first element in the body that has the same (local) name and namespace. If body contains multiple elements with the same name and namespace, the remaining ones are not modified.
Delete All Existing Body Elements	IN	Boolean	Checkbox: False Default. Append to the child list of the body. True Remove all existing elements from the body prior to adding the new element.
Body Element	OUT	Node	Node of the element that has just been added.

SOAP Query Encoding Style

Matches a space separated list of URIs against the ‘encodingStyle' attribute of the element. If the element doesn't have an ‘encodingStyle' attribute, the parent of the element is checked until an element with an ‘encodingStyle' attribute is found. If that attribute contains any of the specified encoding style URIs, the tool returns through ‘Found' and returns the style that was found.

Tip: If the first ‘encodingStyle' attribute found along the parent chain does not contain any of the specified styles, the search does not continue and the tool exits ‘Not Found'.

Exit Paths: Found, Not Found, Failure

Parameter	Dir	Type	Remarks
Element	IN	Node	(child) Element of the SOAP envelope to query. If document node, the document element is queried.

Encoding Styles	IN	String	Space separated list of URIs to match against the ‘encodingStyle' attributes.
First Style Found	OUT	String	Encoding style namespace that was found
Element Of Style	OUT	Node	XML node of the element in which the encoding style attribute was found.

SOAP Get Header

Retrieves the header element from the SOAP envelope if it has one.

Exit Paths: Success, No Header, Failure

Parameter	Dir	Type	Remarks
Envelope	IN	Node	Envelope node of the SOAP payload. Can be a document node whose document element is <SOAP-ENV:Envelope> or the node is the element itself.
Header	OUT	Node	Node of the <SOAP-ENV:Header> element. NULL node if the envelope contains no header.

SOAP Get Header Element

Retrieves the first header element that matches the given base name and namespace. Returns the first element in the header if neither a name nor namespace is given. Takes ‘Not Found' exit if the envelope doesn't have a header or the element can't be found.

Exit Paths: Success, Not Found, No Header, Failure

Parameter	Dir	Type	Remarks
Envelope	IN	Node	Envelope node of the SOAP payload. Can be a document node whose document element is <SOAP-ENV:Envelope> or the node is the element itself.
Base Name	IN	String	Optional. Base Name of the element to return
Namespace	IN	String	Optional. Namespace of the entry to return
Retrieve Value	IN	Boolean	Checkbox: False Do not retrieve value True Default. Return node value
Header Element	OUT	Node	Child element of the <SOAP-ENV:Header> element that has the given base name and namespace. NULL node if the envelope contains no header or the element is not in the

			header.
Element Base Name	OUT	String	Base name of the returned element
Element Namespace	OUT	String	Namespace URI of the returned element
Value	OUT	String	Value of the element (if ‘Retrieve Value' = True)

SOAP Get Header Elements

Returns iterator to a list of header elements filtered by the given arguments. Takes the ‘None' exit if envelope has no header or none of the header elements matched the filter criteria.

Exit Paths: Success, None, No Header, Failure

Parameter	Dir	Type	Remarks
Envelope	IN	Node	Envelope node of the SOAP payload. Can be a document node whose document element is <SOAP-ENV:Envelope> or the node is the element itself.
Base Name	IN	String	Optional. Only include elements with this base name.
Namespace	IN	String	Optional. Only include elements in this namespace.
Must Understand	IN	Boolean	Optional: False Return header entries whose ‘mustUnderstand' attribute is "0" (or no attribute is specified) True Return header entries whose ‘mustUnderstand' attribute is "1". Default: Don't filter on ‘mustUnderstand'
Actor URIs	IN	String	Optional. Space separated list of actor URIs. Only elements whose actor attribute has one of these namespaces is returned. If not specified, don't filter on actor namespace.
Header Elements	OUT	NodeIter	Iterator to collection of header entries. Use the ‘XML Get Next Node' tool to iterate over collection.
Count	OUT	Integer	Number of items in the Header Entries collection

SOAP Add Header Element

Creates a header element and adds it to the given envelope. If the envelope doesn't yet have a header, one will be inserted before the Body element.

If the ‘Name' argument has no namespace prefix and a ‘Namespace' different than **""** (default namespace) is specified, a prefix will be synthesized, unless the local name starts with a ‘**:**' (which is

illegal in XML, and thus signals to this tool *not* to add a synthesized namespace prefix). Adding a prefix can greatly reduce the size of the message if child elements are in no namespace, as otherwise each child element would get an **xmlns=""** attribute.

Exit Paths: Success, Failure

Parameter	Dir	Type	Remarks
Envelope	IN	Node	Envelope node of the SOAP payload. Can be a document node whose document element is <SOAP-ENV:Envelope> or the node is the element itself.
Name	IN	String	Fully qualified name of the header element to create and add to the header.
Namespace	IN	String	Optional. Namespace URI of the element. If the parameter is omitted and the name has a namespace prefix, the tool will search in the parent elements for the namespace with the same prefix and make the element a member of this namespace.
Must Understand	IN	Boolean	Optional. Specifies the value of the ‘mustUnderstand' attribute: False mustUnderstand="0" True mustUnderstand="1" Not specified: No ‘mustUnderstand' attribute is added.
Actor URI	IN	String	Optional. Value of the ‘actor' attribute.
Encoding Style	IN	String	Optional. Value of the ‘encodingStyle' attribute. Attribute is omitted if not specified or "NONE". Specify "STANDARD" for standard namespace ("http://schemas.xmlsoap.org/soap/encoding/").
Value	IN	String	Optional. String value to set as content of the element.
Replace Existing Header Element	IN	Boolean	Checkbox: False Default. Add the element as last child of the body. True Replace first element in the body that has the same (local) name and namespace. If body contains multiple elements with the same name and namespace, the remaining ones are not modified.
Delete All Existing Header Elements	IN	Boolean	Checkbox: False Default. Append to the child list of the body. True Remove all existing elements from the body prior to adding the new element.
Header	OUT	Node	Node of the element that just has been inserted.

Element			

SOAP Get Fault

Retrieves fault information from the SOAP envelope. If there is no <Fault> element in the envelope, the 'No Fault' exit is taken and NULL elements and empty strings are returned. If the envelope is read-only, the returned elements will be read-only too.

Exit Paths: Success, No Fault, Failure

Parameter	Dir	Type	Remarks
Envelope	IN	Node	Envelope node of the SOAP payload. Can be a document node whose document element is <SOAP-ENV:Envelope> or the node is the element itself.
Fault Element	OUT	Node	Node of the <Fault> element.
Fault Code	OUT	String	Value of the <faultcode> element. It provides programmatic information about the fault.
Fault String	OUT	String	Value of the <fautstring> element. It provides human readable information about the fault.
Fault Actor	OUT	String	Value of the <faultactor> element. It provides the URI of the source of the fault.
Detail Element	OUT	Node	Node of the <detail> element. It is used to transfer application specific fault information. NULL Node if there is no <detail> element.

SOAP Set Fault

Adds a <Fault> element to the envelope or replaces an existing one. If one of the mandatory fields (Fault Code, Fault Actor) is empty, the Failure path is taken and XML Get Error Info may be used on the Envelope node to query for error reasons. If the envelope already has a <Fault> element, the tool will remove the existing <Fault> element and replace it with the new element.

Parameter	Dir	Type	Remarks
Envelope	IN	Node	Envelope node of the SOAP payload. Can be a document node whose document element is <SOAP-ENV:Envelope> or the node is the element itself.
Fault Code	IN	String	String to set as value of the <faultcode> element. String must not be empty.
Fault String	IN	String	String to set as value of the <fautstring> element. Should be

			set to provide human readable information.
Fault Actor	IN	String	Optional. String to set as value of the <faultactor> element. If argument is not specified, no <faultactor> element is added.
Create Detail Element	IN	Boolean	Checkbox: *False* Don't create a <detail> element *True* Default. Create an empty <detail> element NOTE: According to the SOAP spec, a <detail> element must be present if the fault is because the <Body> could not be processed successfully.
Preserve Body Elements	IN	Boolean	Checkbox: *False* Default. Remove all existing body elements and replace with <Fault> element *True* Leave existing body elements and append <Fault> element as last child of <Body> NOTE: When sending a fault response to the client, only the <Fault> element is allowed in the body!
Detail Element	OUT	Node	Returns the node of the newly created <detail> element. If 'Create Detail Element' is False, a NULL node is returned.

Exit Paths: Success, Failure

SOAP Create Fault Response

Copies the request envelope and replaces all children of the <Body> element with a single <Fault> element. It thus combines the 'SOAP Create Envelope' and 'SOAP Set Fault' tools. The selection namespaces from the source envelope document are copied to the response envelope document as well.

Exit Paths: Success, Failure

Parameter	Dir	Type	Remarks
Envelope	IN	Node	Envelope node of the request SOAP payload. Can be a document node whose document element is <SOAP-ENV:Envelope> or the node is the element itself.
Fault Code	IN	String	String to set as value of the <faultcode> element. String must not be empty.
Fault String	IN	String	String to set as value of the <fautstring> element. Should be set to provide human readable information.
Fault Actor	IN	String	Optional. String to set as value of the <faultactor> element. If

			argument is not specified, no <faultactor> element is added.
Create Detail Element	IN	Boolean	Checkbox: *False* Don't create a <detail> element *True* Default. Create an empty <detail> element NOTE: According to the SOAP spec, a <detail> element must be present if the fault is because the <Body> could not be processed successfully.
Copy Header	IN	Boolean	Checkbox: *False* Does not copy the <Header> element from the source envelope. *True* Copies the <Header> element and its content from the source envelope.
Response Envelope	OUT	Node	Document node of the response envelope
Detail Element	OUT	Node	Node of the <detail> element of the <Fault> element. If 'Create Detail Element' is False, a NULL node is returned.

SOAP Get RPC Parameter

This is a convenience tool for examining RPC requests. It retrieves a parameter element (child) from the first element in the <Body> element (method in an RPC request). It returns the first element that matches all of the specified arguments. If 'Base Name', 'Namespace', and 'Index' are undefined, the first element will be returned.

For example, to retrieve the 2nd parameter from the 'Add' method in the calculator example presented in Listing 4, you would specify "Parameter2" as name and "" as namespace, or '1' as index.

Exit Paths: Success, Not Found, Failure

Parameter	Dir	Type	Remarks
Envelope	IN	Node	Envelope node of the SOAP payload. Can be a document node whose document element is <SOAP-ENV:Envelope> or the node is the element itself.
Base Name	IN	String	Optional. Base name of the parameter
Namespace	IN	String	Optional. Namespace of the parameter
Index	IN	Integer	Optional. Zero based index into parameters of the method. If this parameter is specified, 'Name' and 'Namespace' may be omitted, but if present must match the name and namespace

			of the parameter.
Retrieve Value	IN	Boolean	Checkbox: *False* Do not retrieve value *True* Default. Return node value Disable retrieval of value if parameter contains a large XML document and the value is not used (performance option).
Parameter Element	OUT	Node	Parameter element
Parameter Base Name	OUT	String	Base name of the parameter element
Parameter Namespace	OUT	String	Namespace URI of the parameter element
Parameter Index	OUT	Integer	Zero based index of the parameter element in the child list of the method element.
Value	OUT	String	Value of the parameter

SOAP Add RPC Parameter

This is a convenience tool for composing RPC requests or responses. It adds a parameter element to the first element in the body of the envelope, which represents the method in RPC requests. Use the XML tools to add complex data (not just a string) to the parameter by manipulating the returned 'Parameter Element' node.

The <Body> element must have a child element (method element). Otherwise this tool fails. When using 'SOAP Create Envelope', you must add a method element using 'SOAP Add Body Element'. The 'SOAP Create RPC Response' tool already adds a method element.

Exit Paths: Success, Failure

Parameter	Dir	Type	Remarks
Envelope	IN	Node	Envelope node of the SOAP payload. Can be a document node whose document element is <SOAP-ENV:Envelope> or the node is the element itself.
Name	IN	String	Qualified name of the parameter
Namespace	IN	String	Optional. Namespace URI of the element. If the parameter is omitted and the name has a namespace prefix, the tool will search in the parent elements for the namespace with the same prefix and make the element a member of that namespace.
Value	IN	String	Optional. Value of the parameter
Parameter Element	OUT	Node	Node of the element that just has been added to the method element.

SOAP Get RPC Method Info

This is a convenience tool for examining RPC requests. It retrieves the first child element of the SOAP <Body> element (Method element in RPC requests). It also returns a collection containing the child elements of the method, which constitute the method arguments. The tool exits through 'No Method' if the body does not contain an element. It returns through <Fault> if the body contains a <Fault> element.

Exit Paths: Success, Fault, No Method, Failure

Parameter	Dir	Type	Remarks
Envelope	IN	Node	Envelope node of the SOAP payload. Can be a document node whose document element is <SOAP-ENV:Envelope> or the node is the element itself.
Method Element	OUT	Node	Node of the method element (first child of the Body)
Method Base Name	OUT	String	Base name of the method element
Method Namespace	OUT	String	Namespace URI of the method element
Parameters	OUT	NodeIter	Iterator to collection of RPC parameter elements. Use the 'SOAP Get Next RPC Parameter' or 'XML Get Next Node' tool to iterate over collection.
Parameter Count	OUT	Integer	Number of items in the Parameters collection

SOAP Get Next RPC Parameter

This tool returns the element node at the current iterator position and returns an iterator to the next position. As the iterator is just a variable, you can make copies at any time to remember a certain position, for example the start position. By using the same variable as input and output iterator, you can easily iterate over the list by connecting the Success path back to this tool (after processing the node, of course). The tool takes the 'End' exit when the iterator points to an empty list or the iteration is complete (list traversed to end).

The tool will fail (take the Failure exit) if the node to which 'Parameter Iterator' points is not an element! This cannot happen if the iterator was obtained through 'SOAP Get RPC Method Info'.

Exit Paths: Success, End, Failure

Parameter	Dir	Type	Remarks

Parameter Iterator	IN	NodeIter	Iterator to collection of parameter of a method.
Retrieve Value	IN	Boolean	Checkbox: *False* Do not retrieve value *True* Default. Return node value Disable retrieval of value if value is not used and parameter may contain a large XML document.
Next Parameter	OUT	NodeIter	Iterator pointing to next parameter in the list
Parameter Element	OUT	Node	Node of the parameter element
Parameter Base Name	OUT	String	Base name of the parameter element
Parameter Namespace	OUT	String	Namespace URI of the parameter element
Value	OUT	String	Value of the parameter

SOAP Create RPC Response

This is a convenience tool for composing the response envelope for an RPC request. It copies the source envelope and replaces the method element in the body with an element that has the same name but "Response" added to its name. It also adds a <Result> element as child of the method element. Usually, the type of the return value is given by the service description and doesn't need to be included in the <Result> element. However, the service may define the type as **xsd:anyType**, for example for VARIANT types. In this case, the type must be included in the argument. The 'Return& Value Type' argument permits specifying the type of the result value. For example, if a type of "double" is specified, the <Result> element will look as follows:

```
<Result xmlns:xsd="http://www.w3.org/2001/XMLSchema"
     xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
     xsi:type="xsd:double">1234.567</Result>
```

The selection namespaces from the source envelope document are copied to the response envelope document as well. The tool fails if the request body does not contain a method element.

Exit Paths: Success, Failure

Parameter	Dir	Type	Remarks
Envelope	IN	Node	Envelope node of the request SOAP payload. Can be a

			document node whose document element is <SOAP-ENV:Envelope> or the node is the element itself.
Method Name Mask	IN	String	Optional. Mask to create the name of the response method. The string passed here may contain the following substitution tags: %1 Namespace prefix of the first child element of the <Body> element (RPC method). %2 Base name of the first child element of the <Body> element (RPC method). %{ Treat everything up to closing '}' as XPath query to be run against the 'Envelope' node and substitute the value of the first node found into element name string. %% '%' character Default: "%1:%2Response".
Method Namespace	IN	String	Optional. Namespace of the method element. If not specified, namespace of request method is used.
Result Element Name	IN	String	Optional. Name of the return value element (first child of the method element). Default: "result"
Result Element Namespace	IN	String	Optional. Namespace URI of the result element. If the parameter is omitted and the name has a namespace prefix, the tool will search in the parent elements for the namespace with the same prefix and make the element a member of that namespace.
Return Value	IN	String	Optional. Return value of the method. It will be set as content of the <Result> child element.
No Return Value (void response)	IN	Boolean	Checkbox: *False* Default. Add a <Result> element. *True* No <Result> element is added (void method).
Copy Header	IN	Boolean	Checkbox: *False* Default. Does not copy the <Header> element from the source envelope. *True* Copies the <Header> element and its content from the source envelope.

Copy Method Element Attributes	IN	Boolean	Checkbox: *False* Default. Don't copy attributes from request method element. *True* Copy all attributes of the request method element into response method element.
Response Envelope	OUT	Node	Document node of the response envelope
Method Element	OUT	Node	Node of the response method element.
Result Element	OUT	Node	Node of the <Result> element in the method element.

SOAP Set Element Type

In SOAP, the type of an argument or the return value is specified by the service description and doesn't need to be included in the payload. However, the service may define the type as **xsd:anyType**, for example for VARIANT types. In this case, the type must be included in the argument. For example, if a type of "double" is specified, an element will look as follows:

```
<Element xmlns:xsd="http://www.w3.org/2001/XMLSchema"
     xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
     xsi:type="xsd:double">1234.567</Element>.
```

The type may be a user defined (complex) type. For example:

```
<ns1:Order xmlns:ns1="uri:my-order-type"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:type="ns1:Order">
<ns1:Product>Watchmacallit</ns1:Product>
<ns1:Quantity>7</ns1:Quantity>
<ns1:Price>19.99</ns1:Price>
</ns1:Order>.
```

Please refer to http://www.w3.org/TR/xmlschema-0 or http://www.w3.org/TR/xmlschema-2 for details on the XML Schema Data types.

Parameter	Dir	Type	Remarks

Element	IN	Node	Node of an element whose Schema instance type to set
Type	IN	String	XSD type to declare for this element. The argument may either be just the type name or have schema namespace prefix, such as xsd:string. If the type argument does not contain a prefix, xsd will be used.
Type Namespace	IN	String	Optional. Namespace of the type. Default: http://www.w3.org/2001/XMLSchema
XSI Namespace	IN	String	Optional. XML Schema Instance namespace. Default: http://www.w3.org/2001/XMLSchema-instance
XSI Namespace Prefix	IN	String	Optional. Prefix of the schema instance namespace. Default: xsi
Declare Namespaces in Envelope	IN	Boolean	Checkbox: *False* Declares the XSD and XSI namespaces in the element itself. *True* Default. Declare the XSD and XSI namespaces in the Envelope element (actually, the document element is used, as this tool may be for other purposes than SOAP). If any of the parent elements already has a NS declaration for a prefix and the namespace URI is different, the declaration will be added to the element, and not the Envelope.

Exit Paths: Success, Failure

SOAP Create Array

Turns an element, for example an RPC parameter, into a SOAP array. The array is created for values supplied as list of strings or just a number of empty elements that can be populated with complex data. The following is a sample array as produced by this tool (default argument):

```
<Element xmlns:xsd="http://www.w3.org/2001/XMLSchema"
     xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
     xmlns:SOAP-ENC="http://schemas.xmlsoap.org/soap/encoding/"
     SOAP-ENC:arrayType="xsd:string[5]"
     xsi:type="SOAP-ENC:Array">
   <xsd:string>first</xsd:string>
   <xsd:string>second</xsd:string>
```

```
    <xsd:string>third</xsd:string>
    <xsd:string>fourth</xsd:string>
    <xsd:string>fifth</xsd:string>
</Element>
```

If the element already has child elements, they are all removed before the array elements are added. The array items may be user defined (complex) types. Use the 'XML Get Next Item' tool to iterate through the 'Item Elements' collection and populate the items. For example:

```
<Element xmlns:ns1="uri:my-order-type"
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xmlns:SOAP-ENC="http://schemas.xmlsoap.org/soap/encoding/"
      SOAP-ENC:arrayType="ns1:Order[3]"
      xsi:type="SOAP-ENC:Array">
    <ns1:Order>
      <ns1:Product>Watchmacallit</ns1:Product>
      <ns1:Quantity>3</ns1:Quantity>
      <ns1:Price>19.99</ns1:Price>
    </ns1:Order>
    <ns1:Order>
      <ns1:Product>Doodleany</ns1:Product>
      <ns1:Quantity>9</ns1:Quantity>
      <ns1:Price>12.49</ns1:Price>
    </ns1:Order>
    <ns1:Order>
      <ns1:Product>Ozadingdong</ns1:Product>
      <ns1:Quantity>1</ns1:Quantity>
      <ns1:Price>43.15</ns1:Price>
    </ns1:Order>
</Element>
```

For details on the XML Schema Data types, refer to http://www.w3.org/TR/xmlschema-0 or http://www.w3.org/TR/xmlschema-2.

Exit Paths: Success, Empty, Failure

Parameter	Dir	Type	Remarks

Element	IN	Node	Node of the parameter to turn into an array.
Values	IN	StringList	Optional. List of strings to set as the array items. If not specified, empty elements will be created.
Size	IN	Integer	Optional. Size of the array. If not specified, the length of the ‘Values' list specifies the size. If both a ‘Values' and ‘Size' argument are given, the ‘Size' has precedent and either not all items of the ‘Values' list are included or the array is padded with elements containing the ‘Default Value'.
Default Value	IN	String	Optional. Default array item value for padding items (if ‘Size' is larger than size of ‘Values' or no ‘Values' defined). Default: No value (padding elements will be empty)
Array Type	IN	String	Optional. Type of the array. The argument may either be just the type name or have schema namespace prefix, such as xsd:string. If the type argument does not have a prefix, xsd will be used. Default: xsd:string
Type Namespace	IN	String	Optional. Namespace of the array type. Default: http://www.w3.org/2001/XMLSchema
Encoding Prefix	IN	String	Optional. Prefix of the encoding namespace (http://schemas.xmlsoap.org/soap/encoding/). Default: SOAP-ENC
Item Element Name	IN	String	Optional. Qualified name of the array items. Default: Qualified array type (thus, the default item element name is xsd:string).
Item Element Namespace	IN	String	Optional. Namespace of the array items. Default: Namespace of the prefix of ‘Item Element Name'. If no prefix, empty namespace.
XSI Namespace	IN	String	Optional. XML Schema Instance namespace. Default: http://www.w3.org/2001/XMLSchema-instance
XSI Namespace Prefix	IN	String	Optional. Prefix of the schema instance namespace. Default: xsi
Include XSI Type Declaration	IN	String	Checkbox: *False* Do not add a type declaration for the array.

			True Default. Add XSI type declaration for the SOAP Array. If all parameters are default the declaration is: xsi:type="SOAP-ENC:Array".
Declare Namespaces in Envelope	IN	Boolean	Checkbox: *False* Declares the namespaces in the element itself. *True* Default. Declare the namespaces in the Envelope element (if they aren't already). If any of the parent elements already has a NS declaration for a prefix and the namespace URI is different, the declaration will be added to the element, and not the Envelope.
Return Item Element Collection	IN	Boolean	Checkbox: *False* Does not return collection of array items (‘Array Items' is returned as NULL). *True* Default. Return collection ‘Array Items' containing all items of the array.
Item Elements	OUT	NodeIter	Iterator pointing to first element of a collection containing the nodes of the array items.
Count	OUT	Integer	Number of items in the array. NOTE: this value is returned, even if ‘Return Item Collection' is False.

Invocation Tools

Invocation Tools

SOAP HTTP Request

This tool issues an HTTP request to the specified URL with the SOAP request envelope as payload. The response body is parsed and returned as response envelope. The URL may have the following format (also see RFC2396 at http://www.rfc.net/rfc2396.html):

```
['http://'] <host> [':' <port>] ['/' <path> ['?' <query>]]
```

The (UNICODE) string passed as URL is converted to UTF-8 and invalid characters in the resulting string are escaped according to RFC2396 (%<hexvalue>). The structure of the request sent to the host will be as follows:

```
'POST ' <path> ' HTTP/1.1' CRLF
'Host: ' <host> [':' <port>] CRLF
'Content-Type: text/xml; charset="' <charset> '"' CRLF
'Content-Length: ' <bodysize> CRLF
'SOAPAction: "' <SOAPAction> '"' CRLF
[<additional headers>]
CRLF
<SOAP envelope body>
```

The ‘Additional HTTP Headers' parameter can be used to supply additional HTTP header elements. The headers must have the form *{<name>* ' :' *<value> [CR] LF }** The header elements in this argument have precedence over the default headers generated by the tool. Thus, if the ‘Additional HTTP Headers' parameter contains a ‘Content-Length header, it will be used (with potentially unexpected results, of course).

The response body will be parsed and returned as ‘Response Envelope' if the content type is text/xml. Otherwise, the body is returned in ‘Raw Response Body' and an empty document node is returned as ‘Response Envelope'. This document node can be queried for information about what went wrong.

This tool maintains a global cache of the most recently resolved and successfully connected host addresses to improve performance. Each address resolution is kept for at most 5 minutes.

Exit Paths

Success

Request was processed successfully (2xx code) and body is valid XML.

SOAP Fault

Response body contains a <Fault> element.

EmptyResponse

Response body was empty and the HTTP status code was 2xx. Some servers use this to signal success for methods with no result (void).

Unknown Host

Invalid or unknown hostname (DNS lookup failed)

Connection Error

Unable to establish connection to server: connection failed or existing connection was lost prematurely.

HTTP Error

HTTP error (3xx, 4xx, 5xx) and it was not a SOAP Fault.

Parse Error

Error parsing the returned XML payload (status was 200 or 500).

Timeout

The request timed out.

Size Limit

The response data exceeded the size limit.

Failure

Some other failure. Use ‘XML Get Error Info' on the ‘Response Envelope' to obtain more information.

Parameter	Dir	Type	Remarks
Request Envelope	IN	Node	XML Node of the SOAP envelope to send. Can be document node or <Envelope> element node.
URL	IN	String	URL of the request. See remarks for details.
SOAP Action	IN	String	Optional. String to be passed as SOAPAction header. The string passed here may contain the following substitution tags: %1 Namespace of the first child element of the <Body> element (RPC method). %2 Base name of the first child element of the <Body> element (RPC method). %{ Treat everything up to closing ‘}' as XPath query to be run against the ‘Request Envelope' node and substitute the value of the first node found into the SOAPAction string. %% ‘%' character If this argument is not specified, the following mask will be used as default: "%1#%2". The value "NONE" may be specified to suppress addition of the SOAPAction header.

Additional HTTP Headers	IN	String	Optional. Additional HTTP Headers, separated by LF characters (\n). See remarks for details.
Selection Namespaces	IN	String	Optional. Selection namespaces to set in response envelope document. Default: Copy selection namespaces from request envelope document.
Timeout	IN	Integer	Optional. Maximum time the request may take before timing out (in milliseconds). -1 à Never timeout. Default: 60000 (60 seconds)
Max Response Size	IN	Integer	Optional. Size limit of the response data. If the data returned by the server exceeds this limit, the data is not processed and the tool fails. This prevents denial of service attacks. Default: 1MB.
Escape URL	IN	Boolean	Checkbox: *False* URL is already escaped. *True* Default. Escape invalid characters in the URL with %<hexvalue> according to RFC2396.
Always Return Raw Response Body	IN	Boolean	Checkbox: *False* Default. Do not return raw response body. *True* Returns the raw data of the response body as string (‘Raw Response Body).
Response Envelope	OUT	Node	Document node of the response envelope. If an error occurred, an empty document is returned which can be queried using ‘XML Get Error Info'.
Status Code	OUT	Integer	HTTP status code of the response (e.g. 200, 500, etc).
Status Text	OUT	String	HTTP status text of the response (e.g. "OK", "Internal Server Error", etc.)
Response Headers	OUT	String	HTTP Headers returned by the server, separated by a LF (\n).
Raw Response Body	OUT	String	Raw data of the response body (data that is parsed as response envelope). This string is only returned if the ‘Always Return Raw Response Body' parameter is True, an error occurs, or the response content type is not XML.

Helper Tools

Helper Tools

SOAP Base64 Encode

Converts the string (which is UNICODE) into the specified character set (default = UTF-8) and encodes the resulting data into a Base64 string. Characters that cannot be translated to the destination character set will be represented as '?'. Wide character sets, such as UTF-16 are currently not supported. SOAP does not mandate a maximum line width for base64 encoded data. Some other protocols, such as MIME, do.

Exit Paths: Success, Failure

Parameter	Dir	Type	Remarks
Data	IN	String	String to encode Base64
Character Set	IN	String	Optional. Character set to convert data into before encoding. Default: 'UTF-8'
Max Line Width	IN	Integer	Optional. Maximum width of a line in characters. –1 = unlimited (default).
Line Separator	IN	String	Optional. String inserted as line separator. Default: "\r\n" (CR/LF)
Encoded Data	OUT	String	String after encoding data Base64

SOAP Base64 Decode

Decodes the base64 encoded string into the binary representation and converts it to UNICODE based on the specified character set. Thus, the character set argument specifies the character set of the base-64 encoded data.

Exit Paths: Success, Failure

Parameter	Dir	Type	Remarks
Encoded Data	IN	String	Base64 encoded data
Character Set	IN	String	Optional. Character set of the base64 encoded data. Default: 'UTF-8'
Decoded	OUT	String	Data after decoding from Base64 and transforming from

Data			'Character Set' to UNICODE.

SOAP Base64 Encode File

Reads the specified file as binary data and encodes it into a base64 string. Encoding a file prepares it for transport inside a SOAP payload. For example, a SOAP request might encode a wave file, and send it to CIC server. SOAP does not mandate a maximum line width for base64 encoded data. Some other protocols, such as MIME, do. This tool can be used to send any kind of data through SOAP requests. For example, you could encode a wave file.

Exit Paths: Success, File Not Found, Access Denied, Failure

Parameter	Dir	Type	Remarks
Filename	IN	String	Filename and path of the file to encode
Max Line Width	IN	Integer	Optional. Maximum width of a line in characters. –1 = unlimited (default).
Line Separator	IN	String	Optional. String inserted as line separator. Default: "\r\n" (CR/LF)
Encoded Data	OUT	String	Base64 encoded content of the file

SOAP Base64 Decode To File

Decodes the base64 encoded string into the binary representation and writes the data to the specified file as binary data.

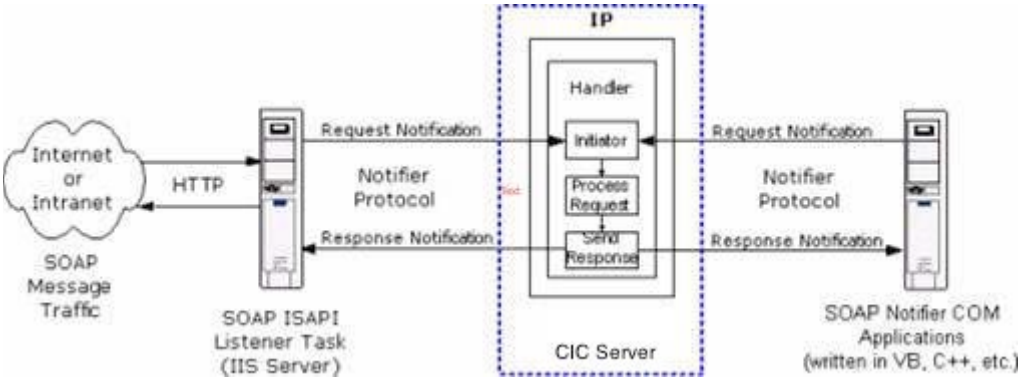
Exit Paths: Success, Access Denied, Failure

Parameter	Dir	Type	Remarks
Encoded Data	IN	String	Base64 encoded data
Filename	IN	String	Filename and path of the file to which to write the decoded data.
Append To Existing File	IN	Boolean	Checkbox: *False* Default. Create new file or truncate existing file. *True* Create new file or append to existing file.

Appendix C: Structure of IP Notification Messages

Appendix C: Structure of IP Notification Messages

For the purpose of the 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 (see SOAP ISAPI Listener Task for IIS). However, any kind of client who "talks" Notifier could issue SOAP requests. For example, a COM object that allows to directly send SOAP packets to CIC.



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

Since Interaction Processor does not directly support Notifier requests, notifications are used to emulate the request/response mechanism. The SOAP request notifications use CIC's eSOAP_REQUEST_OBJECT object type and an object ID that identifies the client. The notification event ("Initiator Event") can either be explicitly specified or the SOAPAction is will be used as default. 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.

Request Message 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 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.
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). See SOAP ISAPI Filter Schema for schema details.
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

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.

Appendix D: SOAP ISAPI Listener Fault Messages

This appendix lists fault messages returned by the SOAP ISAPI Listener. For general information about SOAP Faults, refer to section 4.4 of the SOAP Specification at W3C. The URL is http://www.w3.org/TR/SOAP/. SOAP ISAPI Listener may return the following codes:

Client.ContentType

Unsupported Content-Type specified. Expecting "text/xml" or "application/xml".

Client.ContentLength

The 'Content-Length' field of the HTTP header does not match the length of the data sent by client.

Client.SOAPAction

The HTTP header does not contain a 'SOAPAction' header field.

Client.PayloadSize

The SOAP payload exceeds the maximum size limit configured for the server.

Server.TooBusy

Server is too busy—too many requests are currently pending.

Server.SOAPAction

The SOAPAction is not recognized by the server (e.g. it doesn't match any filter rules).

Server.NotifierConnection

SOAP ISAPI Listener was unable to establish a Notifier connection with the CIC server to forward the request.

Server.RequestTimeout

The request was not processed by the CIC server in the allotted time.

Server.NotifierConnectionLost

The SOAP ISAPI Listener lost the Notifier connection while waiting for the request to be processed by the CIC server.

Server.Switchover

A Switchover was initiated while waiting for the request to be processed. The response was lost.

Server.Error

A general error occurred while server was waiting for request to be processed.

Server.Unhandled

The request was not processed by the CIC server (i.e. a handler was initiated but did not send a response with the SOAP Send Response tool).

Server.Shutdown

The web server was shut down (ISAPI unloaded) while the request was being processed by the CIC server.

Glossary

This section explains special terms used in this documentation.

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 CIC modules.

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.

Customer Interaction Center (CIC)

Customer Interaction Center offers comprehensive interaction management covering not only telephone calls, faxes, and e-mail messages, but also Internet text chats, Web callback requests, and voice over IP calls. Using CIC and the PureConnect platform,, enterprises, contact centers, and service

providers can centralize the processing of all customer interactions and provide a new level of service and consistency.

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:

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.

HTML

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

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.

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.

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.

Revisions

CIC 2018 R2

1. Added procedure, Configuring IC SOAP Listener to work with IC 4.0 and 2015 or later.
2. Added procedure, Additional configuration steps required for SOAP Listener when using IIS7.

CIC 2018 R1

1. Rebranded this document to apply Genesys terminology. Colorized source code. Updated formatting, copyright and trademarks.
2. Deprecated the procedure titled "Install Microsoft SOAP Install Toolkit". Installing the toolkit is no longer necessary. All SOAP Toolkits were replaced by the Microsoft .NET Framework. SOAP Toolkits are no longer supported.

CIC 2015 R4

Added information about new <ICServer2> element in configuration file.

CIC 2015 R1

Updated documentation to reflect changes required in the transition from version 4.0 SU# to CIC 2015 R1, such as updates to product version numbers, system requirements, installation procedures, references to Interactive Intelligence Product Information site URLs, and copyright and trademark information.

CIC 4.0 SU1 and SU2

No revisions were made to this document.

CIC 4.0 GA

1. Installation should be performed using the CIC 4.0 GA DVD. Do not use an CIC 3.x DVD.
2. Updated copyrights and trademarks in this document.
3. The Installing and Using SOAP Functionality Technical Reference Guide was renamed to CIC and SOAP API Developer's Guide. The filename was changed from soap.chm to Soap_API_DG.chm.
4. The *SOAP Notifier COM API Developer Guide* was renamed to SOAP Notifier COM API Developer's Guide. The file name was changed from soapnotifiercom.chm to Soap_Notifier_COM_API_DG.chm.
5. Updated setup instructions for minor changes made to installs.

Copyright and Trademark Information

DISCLAIMER

Genesys Telecommunications Laboratories, Inc.
2001 Junipero Serra Boulevard
Daly City, CA 94014
Telephone/Fax (844) 274-5992
www.genesys.com