Fine-Tuning a Web Service in InterSystems IRIS

Version 2023.3
2024-05-16

InterSystems Corporation 1 Memorial Drive Cambridge MA 02142 www.intersystems.com

Fine-Tuning a Web Service in InterSystems IRIS
InterSystems IRIS Data Platform Version 2023.3 2024-05-16

For Support questions about any InterSystems products, contact:

InterSystems Worldwide Response Center (WRC)
Tel: +1-617-621-0700
Tel: +44 (0) 844 854 2917
Email: support@InterSystems.com

Table of Contents

1

Disabling Access to the Online WSDL

By default, it is possible to view the WSDL for an InterSystems IRIS® web service via a URL of the following form:

```
base/csp/app/web_serv.cls?WSDL
```

Here *base* is the base URL for your web server (including port if necessary), */csp/app* is the name of the web application in which the web service resides, and *web_serv* is the class name of the web service.

To disable the ability to access the WSDL in this way, specify the *SOAPDISABLEWSDL* parameter of the web service as 1. Note that even with *SOAPDISABLEWSDL* equal to 1, it is possible to use the **FileWSDL()** method to generate the WSDL as a static file.

For more basic information about InterSystems IRIS® web services, see Basic Settings of the Web Service.

2

Requiring a Username and Password

To configure an InterSystems IRIS® web service to require a password, you configure its parent web application to use password authentication, and to disallow unauthenticated access.

3

Controlling the XML Types

The WSDL defines the XML types for the arguments and return values of all methods of the web service. For an InterSystems IRIS® web service, the types are determined as follows:

- If the InterSystems IRIS type corresponds to a simple type (such as `%String`), an appropriate corresponding XML type is used.
- If the InterSystems IRIS type corresponds to an XML-enabled class, the *XMLTYPE* parameter of that class specifies the name of the XML type. If that parameter is not specified, the class name (without the package) is used as the XML type name.

  Also, the WSDL defines this type, by using the information in the corresponding class definition.
- If the InterSystems IRIS type corresponds to some other class, the class name (without the package) is used as the XML type name. Also, the WSDL does not define this type.

For further details, see *Projecting Objects to XML*.

Also see WSDL Support in InterSystems IRIS.

4

Controlling the Namespaces of the Schema and Types

This topic describes how to control the namespace for the schema of the WSDL for an InterSystems IRIS® web service, as well as the namespaces for any types defined within it.

4.1 Controlling the Namespace of the Schema

The *TYPENAMESPACE* parameter (of your web service) controls the target namespace for the schema of your web service.

If *TYPENAMESPACE* is null, the schema is in the namespace given by the *NAMESPACE* parameter of the web service. The WSDL might look as follows:

```
<?xml version='1.0' encoding='UTF-8' ?>
...
<types>
<s:schema elementFormDefault='qualified'
targetNamespace = 'http://www.myapp.org'>
...
```

If you set *TYPENAMESPACE* to a URI, that URI is used as the namespace for the types. In this case, the WSDL might look as follows:

```
<?xml version='1.0' encoding='UTF-8' ?>
...
<types>
<s:schema elementFormDefault='qualified'
targetNamespace = 'http://www.mytypes.org'>
...
```

4.2 Controlling the Namespace of the Types

For any types referenced within the schema, the following rules govern how they are assigned to namespaces:

- If the *USECLASSNAMESPACES* parameter of the web service is 0 (the default), then the types are in the same namespace as the schema; see the previous section.
- If the *USECLASSNAMESPACES* parameter of the web service is 1 (and if the web service uses the document binding style), then each type is in the namespace given by the *NAMESPACE* parameter of the corresponding type class.

For a given type, if the *NAMESPACE* parameter is null for the type class, then the type is in the same namespace as the schema; see the previous section.

For information on binding styles, see Specifying the Binding Style for the SOAP Messages.

5

Including Documentation for the Types

By default, the WSDL for an InterSystems IRIS® web service does not include documentation for the types used by the web service.

To include the class documentation for the types within `<annotation>` elements in the schema of the WSDL, specify the *INCLUDEDOCUMENTATION* parameter of the web service as 1.

This parameter does not cause the WSDL to include comments for the web service and its web methods; there is no option to automatically include these comments in the WSDL.

6

Adding Namespace Declarations to the SOAP Envelope

To add a namespace declaration to the SOAP envelope (`<SOAP-ENV:Envelope>` element) of a SOAP message sent by a given web service, modify each web method of that web service so that it invokes the **%AddEnvelopeNamespace()** method of the web service. This method has the following signature:

```
Method %AddEnvelopeNamespace(namespace As %String,
                             prefix As %String,
                             schemaLocation As %String,
                             allowMultiplePrefixes As %Boolean) As %Status
```

Where:

- *namespace* is the namespace to add.
- *prefix* is the optional prefix to use for this namespace. If you omit this argument, a prefix is generated.
- *schemaLocation* is the optional schema location for this namespace.
- *allowMultiplePrefixes* controls whether a given namespace can be declared multiple times with different prefixes. If this argument is 1, then a given namespace can be declared multiple times with different prefixes. If this argument is 0, then if you add multiple declarations for the same namespace with different prefixes, only the last supplied prefix is used.

7

Checking for Required Elements and Attributes

By default, an InterSystems IRIS® web service does not check for the existence of elements and attributes that correspond to properties that are marked as Required. To cause a web service to check for the existence of such elements and attributes, set the *SOAPCHECKREQUIRED* parameter of the web service to 1. The default value for this parameter is 0, for compatibility reasons.

8

Controlling the Form of Null String Arguments

Normally, if an argument is omitted, an InterSystems IRIS® web service omits the corresponding element in the SOAP message that it sends. To change this, set the *XMLIGNORENULL* parameter to 1 in the web service class; in this case, the SOAP message includes an empty element.

Note: This parameter affects only web method arguments of type `%String`.

9

Controlling the Message Name of the SOAP Response

In an InterSystems IRIS® web service, you can control the message name used in the response received from a web method. By default, this message name is the name of the web method with `Response` appended to the end. The following example shows a response from a web method called `Divide`; the response message name is `DivideResponse`.

XML

```
<SOAP-ENV:Body>
   <DivideResponse xmlns="http://www.myapp.org">
       <DivideResult>.5</DivideResult>
   </DivideResponse>
</SOAP-ENV:Body>
```

To specify a different response message name, set the SoapMessageName keyword within the web method definition.

Note that you cannot change the name of the SOAP message that invokes a given web method; this name of this message is the name of the method. You can, however, override the SOAP action as given in the HTTP request; see Overriding the Default HTTP SOAP Action.

10

Overriding the HTTP SOAP Action and Request Message Name

When you invoke a web method via HTTP, the HTTP headers must include the SOAP action, which is a URI that indicates the intent of the SOAP HTTP request. For SOAP 1.1, the SOAP action is included as the `SOAPAction` HTTP header. For SOAP 1.2, it is included within the `Content-Type` HTTP header.

The SOAP action indicates the intent of the SOAP HTTP request. The value is a URI identifying the intent; it is generally used to route the inbound SOAP message. For example, a firewall could use this header to appropriately filter SOAP request messages in HTTP.

For a web method in an InterSystems IRIS® web service, the `SOAPAction` HTTP header has the following form by default (for SOAP 1.1):

```
SOAPAction: NAMESPACE/Package.Class.Method
```

Where *NAMESPACE* is the value of the *NAMESPACE* parameter for the web service, and *Package.Class.Method* is the name of the method that you are using as a web method. For example:

```
SOAPAction: http://www.myapp.org/GSOAP.WebService.GetPerson
```

To override this, specify a value for the SoapAction method keyword, within the definition of the web method. Specify a quoted string that indicates that identifies the intent of the SOAP request. In the typical scenario, each web method in the web service specifies a unique value (if any) for SoapAction.

If SoapAction is not unique within this web service, each method must have a unique value of the SoapRequestMessage method keyword. This keyword specifies the name of the top element in the SOAP body of the request message. Note that SoapRequestMessage has an effect only for wrapped document/literal messages.

11

Specifying Whether Elements Are Qualified

The *ELEMENTQUALIFIED* parameter (of your web service) controls the value of the `elementFormDefault` attribute in the schema of the WSDL. Specifically:

- If *ELEMENTQUALIFIED* is 1, then `elementFormDefault` is `"qualified"`.
- If *ELEMENTQUALIFIED* is 0, then `elementFormDefault` is `"unqualified"`.

The default value for this parameter depends on the value of the SoapBodyUse class keyword. See the *Class Definition Reference*. Normally SoapBodyUse is `"literal"`, which means that *ELEMENTQUALIFIED* is 1.

For information on the differences between qualified and unqualified elements, as well as examples, see *Projecting Objects to XML*.

12

Controlling Whether Message Parts Use Elements or Types

Your web service has a parameter (*XMLELEMENT*) that controls the precise form of the message parts of the SOAP messages. Specifically:

- If *XMLELEMENT* is 1, then the `<part>` element has attributes called `name` and `element`. In this case, the WSDL contains a sample `<message>` element as follows:

  XML

  ```
  <message name="GetPersonSoapOut">
    <part name="GetPersonResult" element="s0:Person" />
  </message>
  ```

- If *XMLELEMENT* is 0, then the `<part>` element has attributes called `name` and `type`. In this case, the WSDL contains a sample `<message>` element as follows:

  XML

  ```
  <message name="GetPersonSoapOut">
    <part name="GetPersonResult" type="s0:Person" />
  </message>
  ```

The default value for this parameter depends on the value of the SoapBodyUse class keyword. See the *Class Definition Reference*. Normally SoapBodyUse is `"literal"`, which means that *XMLELEMENT* is 1.

13

Controlling Use of the xsi:type Attribute

By default, InterSystems IRIS® SOAP messages include the `xsi:type` attribute only for the top-level types. For example:

```
<?xml version="1.0" encoding="UTF-8" ?>
...
<types:GetPersonResponse>
<GetPersonResult href="#id1" />
</types:GetPersonResponse>
<types:Person id="id1" xsi:type="types:Person">
<Name>Yeats,Clint C.</Name>
<DOB>1944-12-04</DOB>
</types:Person>
...
```

In these examples, line breaks have been added for readability. To use this attribute for *all* types in the SOAP messages, set the *OUTPUTTYPEATTRIBUTE* parameter or the OutputTypeAttribute property to 1. The same output would look like this:

```
<?xml version="1.0" encoding="UTF-8" ?>
...
<types:GetPersonResponse>
<GetPersonResult href="#id1" />
</types:GetPersonResponse>
<types:Person id="id1" xsi:type="types:Person">
<Name xsi:type="s:string">Yeats,Clint C.</Name>
<DOB xsi:type="s:date">1944-12-04</DOB>
</types:Person>
...
```

This parameter has no effect on the WSDL of the web service.

14

Controlling Use of Inline References in Encoded Format

In an InterSystems IRIS® web service, with encoded format, any object-valued property is included as a reference, and the referenced object is written as a separate element in the SOAP message.

To instead write the encoded objects inline, specify the *REFERENCESINLINE* parameter or the ReferencesInline property as 1.

The property takes precedence over the parameter.

15

Specifying the SOAP Envelope Prefix

By default, an InterSystems IRIS® web service uses the prefix `SOAP-ENV` in the envelope of the SOAP messages it sends. You can specify a different prefix. To do so, set the *SOAPPREFIX* parameter of the web service. For example, if you set this parameter equal to `MYENV`, the web service includes this prefix in its messages, as follows:

XML

```
<?xml version="1.0" encoding="UTF-8" ?>
<MYENV:Envelope xmlns:MYENV='http://schemas.xmlsoap.org/soap/envelope/'
xmlns:xsi='http://www.w3.org/2001/XMLSchema-instance'
xmlns:s='http://www.w3.org/2001/XMLSchema'>
  <MYENV:Body>
   <DivideResponse xmlns="http://www.myapp.org">
      <DivideResult>.5</DivideResult>
   </DivideResponse>
  </MYENV:Body>
</MYENV:Envelope>
```

The *SOAPPREFIX* parameter also affects the prefix used in any SOAP faults generated by the web service.

This parameter has no effect on the WSDL of the web service.

16

Restricting the SOAP Versions Handled by a Web Service

By default, an InterSystems IRIS® web service can handle SOAP requests that use SOAP version 1.1 or 1.2. To modify the web service so that it can handle only SOAP requests for a specific SOAP version, set the *REQUESTVERSION* parameter. This parameter can equal `"1.1"`, `"1.2"`, or `""`. If this parameter is `""`, the web service has the default behavior.

Note that the *SOAPVERSION* parameter does not affect the versions supported by the web service; it only controls which versions are advertised in the WSDL.

17

Sending Responses Compressed by gzip

An InterSystems IRIS® web service can compress its response messages with gzip, a free compression program that is widely available on the Internet. This compression occurs after any other message packaging (such as creating MTOM packages). To cause a web service to do so, set the *GZIPOUTPUT* parameter equal to 1.

This parameter has no effect on the WSDL of the web service.

If you make this change, be sure that the web client can automatically decompress the message with gunzip, the corresponding decompression program.

If the web client is an InterSystems IRIS web client, note that the Web Gateway automatically decompresses inbound messages before sending them to the web client.

18

Defining a One-Way Web Method

For an InterSystems IRIS® web service, normally, when you execute a web method, a SOAP message is returned, even if the method has no return type and returns nothing. This SOAP response message has the following general form:

XML

```
<?xml version="1.0" encoding="UTF-8" ?>
<SOAP-ENV:Envelope
xmlns:SOAP-ENV='http://schemas.xmlsoap.org/soap/envelope/'
xmlns:xsi='http://www.w3.org/2001/XMLSchema-instance'
xmlns:s='http://www.w3.org/2001/XMLSchema'>
  <SOAP-ENV:Body>
   <MethodNameResponse xmlns="http://www.myapp.org"></MethodNameResponse>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

In rare cases, you might need to define a web method as being one-way. Such a method must return no value, and no SOAP response is expected to the request message. To define a one-way web method, define the return type of the method as %SOAP.OneWay. In this case:

- The WSDL does not define output defined for this web method.
- The web service does not return a SOAP message (unless the service adds a header element; see the subsection). That is, the HTTP response message does not include any XML content.

Note: One-way methods should normally not be used. A request-response pair is much more common, supported, and expected — even for a method that has no return type.

See WSDL Differences for One-Way Web Methods.

18.1 One-Way Web Methods and SOAP Headers

If the web method adds a header element, then the HTTP response does include XML content as follows:

```
<?xml version="1.0" encoding="UTF-8" ?>
<SOAP-ENV:Envelope xmlns:SOAP-ENV='http://schemas.xmlsoap.org/soap/envelope/' ...
  <SOAP-ENV:Header>
     header elements as set by the web service
 </SOAP-ENV:Header>
  <SOAP-ENV:Body></SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

18.2 Dynamically Making a Web Method One Way

You can also dynamically redefine a web method to be one way. To do so, invoke the **ReturnOneWay()** of the web service within the definition of the web method. For example:

Class Member

```
Method HelloWorldDynamic(oneway as %Boolean = 0) As %String [ WebMethod ]
{
  If oneway {Do ..ReturnOneWay() }
  Quit "Hello world "
}
```

If the argument is 0, this web method returns a SOAP response whose body contains `Hello world`. If the argument is 1, this method does not return a SOAP response.

19

Adding Line Breaks to Binary Data

You can cause an InterSystems IRIS® web service to include automatic line breaks for properties of type `%Binary` or `%xsd.base64Binary`. To do so, do either of the following:

- Set the *BASE64LINEBREAKS* parameter to 1 in the web service class.
- Set the `Base64LineBreaks` property to 1, for the web service class instance. The value of this property takes precedence over the value set by the *BASE64LINEBREAKS* parameter.

For the parameter and the property, the default value is 0; by default, an InterSystems IRIS web service does not include automatic line breaks for properties of type `%Binary` or `%xsd.base64Binary`.

20

Adding a Byte-Order Mark to the SOAP Messages

By default, a message sent by an InterSystems IRIS® web service does not start with a BOM (byte-order mark).

The BOM is usually not needed because the message is encoded as UTF-8, which does not have byte order issues. However, in some cases, it is necessary or desirable to include a BOM in a SOAP message; this BOM merely indicates that the message is UTF-8.

To add a BOM to the messages sent by an InterSystems IRIS web service, set the RequestMessageStart property of the service. This property must equal a comma-separated list of the parts to include at the start of a message. These parts are as follows:

- `DCL` is the XML declaration:

  ```
  <?xml version="1.0" encoding="UTF-8" ?>
  ```

- `BOM` is the UTF-8 BOM.

The default is `"DCL"`.

In practice, RequestMessageStart can equal any of the following values:

- `"DCL"`
- `"BOM"`
- `"BOM,DCL"`

21

Customizing the Timeout Period

The Web Gateway waits for a fixed length of time for an InterSystems IRIS® web service to send a response message. For information on setting the timeout period, see Configuring the Default Parameters for Web Gateway in the *Web Gateway Guide*.

In some cases, you might know that a given web method requires a longer period before it can complete. If so, you can specify the timeout period for that method. To do so, near the start of the definition of that web method, add a line to set the *Timeout* property of the web service. Specify a timeout period in seconds. For example, if the default timeout period is three minutes and you need the timeout period to be five minutes, you might do the following:

```
Method LongRunningMethod(Input) as %Status [ WebMethod ]
{
   set ..Timeout=300; this method will not time out until 5 minutes
   //method implementation here
}
```

22

Using Process-Private Globals to Support Very Large Messages

By default, an InterSystems IRIS® web service usually uses local array memory when it parses requests or responses. You can force it to use process-private globals instead; this enables the web service to process very large messages.

To do so, specify the *USEPPGHANDLER* parameter of the web service class as follows:

```
Parameter USEPPGHANDLER = 1;
```

If this parameter is 1, then the web service always uses process-private globals when it parses requests or responses. If this parameter is 0, then the web service always uses local array memory for these purposes. If this parameter is not set, then the web service uses the default, which is usually local array memory.

23

Customizing Callbacks of a Web Service

You can customize the behavior of an InterSystems IRIS® web service by overriding its callback methods:

OnRequestMessage()

Called when the web service receives a request message, if there is no security error; this callback is not invoked in the case of a security error. The system invokes this callback after performing security processing, after checking the envelope for errors, and after processing the actions specified in the WS-Addressing header (if any). This callback is useful for tasks such as logging raw SOAP requests.

This method has the following signature:

```
Method OnRequestMessage(mode As %String, action As %String, request As %Stream.Object)
```

Where:

- *mode* specifies the type of SOAP request. This is either `"SOAP"` or `"binary"`.
- *action* contains the value of SOAPAction header.
- *request* contains the SOAP request message in a stream.

This method can use the object *%request*, which is an instance of %CSP.Session. In this object:

- The Content property contains the raw request message.
- The **NextMimeData()** instance method enables you to retrieve individual MIME parts (if this is a MIME SOAP request).

This method can also use properties of the web service instance. The following properties are set during initialization:

- The ImportHandler property contains the DOM for parsed SOAP request message.
- The SecurityIn property contains the WS-Security header element. For details, see *Securing Web Services*.
- The SecurityNamespace property contains the namespace for the WS-Security header element.
- The SoapFault property is set if SOAP fault has been generated.

To return a fault within **OnRequestMessage()**, set the SoapFault property. Do not call the **ReturnFault()** method.

OnPreWebMethod()

Called just before a web method is executed; does nothing by default. This method takes no arguments and cannot return a value. This method therefore cannot change the execution of the web service except by returning a SOAP fault in the same way that a web method would do.

This method can use *%request*, *%session*, and the web service properties. Note that the MsgClass property of the web service is the message descriptor class that contains the web method arguments.

OnPostWebMethod()

Called just after a web method is executed; does nothing by default. This method takes no arguments and cannot return a value. This method therefore cannot change the execution or return value of the web method. You customize this method primarily to clean up required structures created by **OnPreWebMethod()**.

24

Specifying Custom Transport for a Web Service

By default, an InterSystems IRIS® web service responds to transport in a specific way, described here. You can customize this behavior.

24.1 Background

When an InterSystems IRIS web service receives a SOAP message, it executes its **OnSOAPRequest()** class method. By default, this method does the following:

1. Initializes the web service instance by calling its **Initialize()** method. This method parses the inbound SOAP message, returns several pieces of information by reference, and processes the security header. See the documentation for the %SOAP.WebService class.
2. Sets properties of the web service instance, such as SoapFault and others.
3. Initializes the response stream.
4. Invokes the **Process()** method of the web service, passing to it the SOAP action and the method to invoke.
5. Resets the web service instance by calling its **Reset()** method.
6. Copies the result into the response stream.

24.2 Defining Custom Transport for a Web Service

To implement a web service using your own transport, get the SOAP message as a stream using your transport, instantiate the web service class and call its **OnSOAPRequest()** class method.

The **OnSOAPRequest()** method must transport the request to the web service and obtain the response. To indicate an error, it must return a SOAP fault in the response stream. The signature of this method must be as follows:

```
Method OnSOAPRequest(action,requestStream, responseStream)
```

Here:

1. *action* is a %String that specifies the SOAP action. The piece of the action string after the last "." is used as the method name for using the correct descriptor class. If action is null, then the element name from the first element (the wrapping element) in the SOAP body is used as the method name.
2. *requestStream* is a stream that contains the SOAP request message encoded according to the encoding attribute of the XML directive.
3. *responseStream* is a character stream produced as the SOAP response that contains the response SOAP message encoded in UTF-8. You can create this argument before calling **OnSOAPRequest()** and passed it in with the method call. Or this argument can be a variable passed by reference. In this case, **OnSOAPRequest()** must set it equal to an instance of %FileCharacterStream that contains the response.

25

Defining Custom Processing in a Web Service

In rare scenarios, it may be useful to define an InterSystems IRIS® web service that uses custom processing to handle inbound messages and to build response messages. In these scenarios, you implement either the **ProcessBodyNode()** method or the **ProcessBody()** method in the web service. This topic provides the details.

25.1 Overview

In custom processing, you parse the inbound message and construct the response manually. The requirements are as follows:

- In the web service, you define web methods that have the desired signatures. You do this to establish the WSDL of the web service. These web methods (or some of them) can be stubs. A method is executed only if **ProcessBodyNode()** or **ProcessBody()** returns 0.
- Also in the web service, you implement one of the following methods:
  - **ProcessBodyNode()** — This method receives the SOAP body as an instance of %XML.Node. You can use InterSystems IRIS XML tools to work with this instance and build the response message. The SOAP envelope is available in the Document property of this instance of %XML.Node.
  - **ProcessBody()** — This method receives the SOAP Body as a stream. Because the SOAP body is an XML fragment rather than an XML document, you cannot use the InterSystems IRIS XML tools to read it. Instead, you parse the stream with ObjectScript functions and extract the needed parts.

  If you define both of these methods, the **ProcessBodyNode()** method is ignored.

In either case, the response message that you construct must be consistent with the WSDL of the web service.

25.2 Implementing ProcessBodyNode()

The **ProcessBodyNode()** method has the following signature:

```
method ProcessBodyNode(action As %String, body As %XML.Node,
       ByRef responseBody As %CharacterStream) as %Boolean
```

Where:

- *action* is the SOAP action specified in the inbound message.
- *body* is an instance of %XML.Node that contains the SOAP `<Body>`.
- *responseBody* is the response body serialized as an instance of %Library.CharacterStream. This stream is passed by reference and is initially empty.

If you implement this method in a web service, the method should do the following:

1. Examine the *action* and branch accordingly. For example:

   ObjectScript

   ```
   if action["action1" {
    //details
   }
   ```

2. If you need to access the SOAP `<Envelope>` (for example, to access its namespace declarations), use the Document property of *body*. This equals an instance of %XML.Document, which represents the SOAP envelope as a DOM (Document Object Model).

   Otherwise, use *body* directly.

3. Now you have the following options:

   - Use %XML.Writer to write the body as a string, which you can then manipulate. For example:

     ObjectScript

     ```
     set writer=##class(%XML.Writer).%New()
     do writer.OutputToString()
     do writer.DocumentNode(body)
     set request=writer.GetXMLString(.sc)
     // check returned status and continue
     ```

   - Use methods of %XML.Document or %XML.Node, as appropriate, to navigate through the document. Similarly, use properties of %XML.Document or %XML.Node to access information about the current part of the document.
   - Use XPath expressions to extract data.
   - Perform XSLT transformations.

   For details, see *Using XML Tools*. Be sure to check the status returned by methods in these classes, to simplify troubleshooting in the case of an error.

4. If an error occurs during the processing of the request, return a fault in the usual way using the **ReturnFault()** method.

5. Use the **Write()** method of the response stream to write the XML fragment which will become the child element of `<Body>`.

6. If a response stream is created, return 1. Otherwise, return 0, which causes InterSystems IRIS to run the web method associated with the given action.

   For example:

   ObjectScript

   ```
    if action["action1" {
      //no custom processing for this branch
      quit 0
     } elseif action["action2" {
        //details
        //quit 1
     }
   ```

25.3 Implementing ProcessBody()

The **ProcessBody()** method has the following signature:

```
method ProcessBody(action As %String, requestBody As %CharacterStream,
                   ByRef responseBody As %CharacterStream) as %Boolean
```

Where:

- *action* is the SOAP action specified in the inbound message.
- *requestBody* is an instance of %Library.CharacterStream that contains the SOAP `<Body>` element. The stream contains an XML fragment, not a complete XML document.
- *responseBody*, is the response body serialized as an instance of %Library.CharacterStream. This stream is passed by reference and is initially empty.

If you implement this method in a web service, the method should do the following:

1. Examine the *action* and branch accordingly. For example:

   ObjectScript

   ```
    if action["action1" {
     //details
    }
   ```

2. Use the **Read()** method of *requestBody* to obtain the SOAP `<Body>`. For example:

   ObjectScript

   ```
    set request=requestBody.Read()
   ```

3. Parse this stream by using tools such as **$EXTRACT**. For example:

   ObjectScript

   ```
    set in1="<echoString xmlns=""http://soapinterop.org/xsd""><inputString>"
    set in2="</inputString></echoString>"
    set contents=$extract(request,$length(in1)+1,*-$length(in2))
   ```

4. If an error occurs during the processing of the request, return a fault in the usual way using the **ReturnFault()** method.
5. Use the **Write()** method of the response stream to write the XML fragment that will become the child element of `<Body>`. For example:

   ObjectScript

   ```
    set in1="<echoString xmlns=""http://soapinterop.org/xsd""><inputString>"
    set in2="</inputString></echoString>"
    set request=requestBody.Read()
    if ($extract(request,1,$length(in1))'=in1) || ($extract(request,*-$length(in2)+1,*)'=in2) {
      do responseBody.Write("Bad Request: "_request)
      quit 1
    }

    set out1="<echoStringResponse xmlns=""http://soapinterop.org/xsd""><echoStringResult>"
    set out2="</echoStringResult></echoStringResponse>"
    do responseBody.Write(out1)
    do responseBody.Write($extract(request,$length(in1)+1,*-$length(in2)))
    do responseBody.Write(out2)
   ```

6. If a response stream is created, return 1. Otherwise, return 0, which causes InterSystems IRIS to run the web method associated with the given action.