Fine-Tuning a Web Client in InterSystems IRIS

Version 2024.1
2024-05-02

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

Fine-Tuning a Web Client in InterSystems IRIS
InterSystems IRIS Data Platform Version 2024.1 2024-05-02

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

Modifying a Web Client

After you generate an InterSystems IRIS® data platform web client class, you do not usually edit the class. Instead you write code that creates an instance of that class and that provides provide client-side error handling. This topic discusses various ways to fine-tune the InterSystems IRIS web client, either by modifying the web client instance or (less commonly) by modifying the generated class.

Note: Do not create a subclass of the generated web client class. The compiler will not generate the supporting classes that it would need in order to run properly, and your subclass would not be usable.

2

Disabling Keep-Alive for a Web Client

By default, if you reuse an InterSystems IRIS® data platform web client instance to send multiple request messages, InterSystems IRIS sends all the messages in a single HTTP transmission (using a HTTP 1.1 keep-alive connection). Specifically, InterSystems IRIS keeps the TCP/IP socket open so that InterSystems IRIS does not need to close and reopen it. To disable this keep-alive behavior, do either of the following:

- Kill the web client instance and create and use a new one.
- Set the client's HttpRequest.SocketTimeout property to 0 after you send the first message. For example:

```
Set client.HttpRequest.SocketTimeout=0
```

Note: If you are using WS-ReliableMessaging and you are using SSL/TLS to communicate with the web service, do not disable keep-alive. For information on WS-ReliableMessaging, see *Securing Web Services*.

3

Controlling the Form of Null String Arguments

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

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

4

Controlling the Client Timeout

You can control two separate timeout periods for an InterSystems IRIS® data platform web client:

- The Timeout property of the web client is the read timeout. This specifies how long, in seconds, the web client waits for a response.

  If this property is not specified, the web client uses the default value specified for the Timeout property of the %Net.HttpRequest class. This default is 30 seconds.

  If you are using a proxy server, this property controls how long the client waits for a response from the proxy.

- The OpenTimeout property specifies the open timeout, which is the number of seconds to wait for the TCP/IP connection to open. If this property is not specified, the value specified by Timeout is used instead.

5

Using a Proxy Server

A InterSystems IRIS® data platform web client can communicate with a web service via a proxy server. In order to set this up, specify properties of the web client instance to indicate the proxy server to use. These properties are as follows:

HttpProxyServer

Specifies the host name of the proxy server to use. If this property is not null, the HTTP request is directed to this machine.

For information on specifying the default proxy server, see Using a Proxy Server.

HttpProxyPort

Specifies the port to connect to, on the proxy server.

For information on specifying the default proxy port, see Using a Proxy Server.

HttpProxyHTTPS

Specify this as true if you are using a proxy server and if that proxy server supports HTTPS.

Note that if you are using HTTPS, you must also set the SSLConfiguration property of the client equal to the name of the SSL/TLS configuration; for more details, see Configuring the Client to Use SSL.

HttpProxyAuthorization

If the web client should authenticate itself with the proxy server, specify this as the required `Proxy-Authorization` header field.

HttpProxyTunnel

Specify this as true if the web client should establish a tunnel through the proxy to the target HTTP server. If true, the request uses the HTTP CONNECT command to establish a tunnel. The address of the proxy server is taken from the HttpProxyServer and HttpProxyPort properties. If the endpoint URL has the https: protocol, then once the tunnel is established, InterSystems IRIS negotiates the SSL connection. In this case, the HttpProxyHTTPS property is ignored because the tunnel establishes a direct connection with the target system.

6

Setting HTTP Headers

If you need further control over the HTTP headers sent by a web client, you can use the following methods of %SOAP.WebClient:

SetHttpHeader()

Adds a header to the HTTP request. Note that the `Content-Type`, `Content-Encoding`, and `Content-Length` headers are part of the entity body rather than the HTTP main headers. You cannot set the `Content-Length` header, which is read-only. Nor can you set the `Connection` header, because this class does not support persistent connections.

ResetHttpHeaders()

Clear all HTTP headers.

Also see Using HTTP User Authentication.

7

Specifying the HTTP Version to Use

By default, an InterSystems IRIS web client uses HTTP/1.1. You can instead use HTTP/1.0. To do so, set the HttpVersion property of the client to `"1.0"`.

8

Disabling SSL Server Name Checking

By default, when an InterSystems IRIS® data platform web client connects to a server via SSL, it checks that the certificate server name matches the DNS name used to connect to the server. (This checking is described in RFC 2818 section 3.1. Wildcard support is described in RFC 2595, but browsers generally do check the server name, and InterSystems has chosen to do the same.)

To disable this checking, set SSLCheckServerIdentity property of the client to `0`.

9

Controlling Use of the xsi:type Attribute

By default, InterSystems IRIS® data platform 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, do either of the following:

- Set the OutputTypeAttribute property equal to 1 in the web client instance.
- Set the *OUTPUTTYPEATTRIBUTE* parameter equal to 1 in the web client class.

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>
...
```

The property takes precedence over the parameter.

10

Controlling Use of Inline References for Encoded Format

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, in your web client, specify the *REFERENCESINLINE* parameter or the ReferencesInline property as 1. The property takes precedence over the parameter.

11

Specifying the Envelope Prefix

By default, an InterSystems IRIS® data platform web client 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 client class. For example, if you set this parameter equal to `MYENV`, the web client includes this prefix in its messages instead of `SOAP-ENV`.

12

Adding Namespace Declarations to the SOAP Envelope

To add a namespace declaration to the SOAP envelope (<SOAP-ENV:Envelope> element) of the SOAP responses returned by a given web client, call the **%AddEnvelopeNamespace()** method of the web client before calling the web method. 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.

13

Sending Responses Compressed by gzip

An InterSystems IRIS® data platform web client 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 client to do so, do either of t he following:

- Set the GzipOutput property equal to 1 in the web client instance.
- Set the *GZIPOUTPUT* parameter equal to 1 in the web client class.

If you do so, be sure that the web service can automatically decompress the message with gunzip, the corresponding decompression program. (If the web service is an InterSystems IRIS web service, note that the Web Gateway automatically decompresses inbound messages before sending them to the web service.)

14

Quoting the SOAP Action (SOAP 1.1 Only)

In SOAP 1.1 request messages, the HTTP header includes a `SOAPAction` line as follows:

```
POST /csp/gsop/GSOP.DivideWS.cls HTTP/1.1
User-Agent: Mozilla/4.0 (compatible; InterSystems IRIS;)
Host: localhost:8080
Connection: Close
Accept-Encoding: gzip
SOAPAction: http://www.mynamespace.org/GSOAP.DivideWS.Divide
Content-Length: 397
Content-Type: text/xml; charset=UTF-8

...
```

By default, the value for `SOAPAction` is not quoted. To place this value in quotes, specify *SOAPACTIONQUOTED* as 1 in the web client class. Then the HTTP header of the request message would be as follows:

```
POST /csp/gsop/GSOP.DivideWS.cls HTTP/1.1
User-Agent: Mozilla/4.0 (compatible; InterSystems IRIS;)
Host: localhost:8080
Connection: Close
Accept-Encoding: gzip
SOAPAction: "http://www.mynamespace.org/GSOAP.DivideWS.Divide"
Content-Length: 397
Content-Type: text/xml; charset=UTF-8

...
```

In SOAP 1.2, the *SOAPACTIONQUOTED* parameter has no effect. This is because the request messages do not have a `SOAPAction` line. Instead the SOAP action is always quoted and is included within the `Content-Type` line as follows:

```
Content-Type: application/soap+xml;
   charset=UTF-8; action="http://www.mynamespace.org/GSOAP.DivideWS.Divide"
```

Note: This example shows an artificial line break, to ensure that the line fits on the page when this content is formatted as PDF.

15

Treating HTTP Status 202 Like Status 200

By default, an InterSystems IRIS® data platform web client follows the standard WS-I Basic Profile, which uses the HTTP response status 202 only if the HTTP response does not contain a SOAP envelope.

If you want to treat HTTP status 202 in the same way as HTTP status 200, set the HttpAccept202 property of the client to 1. To see the actual return status, check the HttpResponse.StatusCode property of the client.

The WS-I Basic Profile supports but does not encourage this practice: “The Profile accepts both status codes because some SOAP implementations have little control over the HTTP protocol implementation and cannot control which of these response status codes is sent.”

16

Defining a One-Way Web Method

Normally, when a web client calls a web service, a SOAP message is returned, even if the method has no return type and returns nothing when executed in InterSystems IRIS® data platform.

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 message.

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.

To define a one-way web method, define the return type of the method as %SOAP.OneWay. The WSDL does not define output defined for this web method, and the web service does not return a SOAP message.

17

Adding Line Breaks to Binary Data

You can cause the InterSystems IRIS® data platform 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.

18

Adding a Byte-Order Mark to the SOAP Messages

By default, a message sent by an InterSystems IRIS® data platform web client 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 client, set the RequestMessageStart property of the client. 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"`

19

Using Process-Private Globals When Parsing

By default, an InterSystems IRIS® data platform web client 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 client 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 client always uses process-private globals when it parses requests or responses. If this parameter is 0, then the web client always uses local array memory for these purposes. If this parameter is not set, then the web client uses the default, which is usually local array memory.

You can override this parameter at runtime. To do so, set the *UsePPGHandler* property of the web client instance.

20

Creating Custom SOAP Messages

In special cases, you may want a web client to send a custom SOAP message. The essential requirements are as follows:

1. Create a subclass of %SOAP.WebRequest and set its *LOCATION* parameter or Location property.
2. In this subclass, create a method to send a SOAP message. This method must create an instance of %Library.CharacterStream and place into it the SOAP message you want to send. It is your responsibility to ensure that the message is correctly formed.
3. The method must next invoke the **SendSOAPBody()** method:

```
method SendSOAPBody(Action As %String,
                    OneWay As %Boolean = 0,
                    Request As %CharacterStream,
                    ByRef Response) as %Status
```

   - *Action* is a string that gives the name of SOAP action to perform.
   - *OneWay* is a true/false flag that controls whether the message is one way.
   - *Request* is an instance of %Library.CharacterStream that contains the body of the SOAP request in the character set of the current locale.
   - *Response* is the response, returned by reference either as a character stream or an instance of %XML.Node.

     If *Response* is null when you invoke **SendSOAPBody()**, then the method sets *Response* equal to an instance of %Library.CharacterStream. This stream contains the body of the SOAP response in the character set of the current locale.

     If *Response* is an instance of %Library.CharacterStream when you invoke **SendSOAPBody()**, then the method updates *Response* to contain the body of the SOAP response in the character set of the current locale.

     If *Response* is an instance of %XML.Node when you invoke **SendSOAPBody()**, then the method updates *Response* to point to the body DOM.

%SOAP.WebRequest is a subclass of %SOAP.WebClient, so you may want to set other parameters and properties. You can also add SOAP headers as described elsewhere. See the class documentation for %SOAP.WebRequest for further notes.

21

Specifying Custom HTTP Requests

By default, an InterSystems IRIS® data platform web client uses HTTP to transport the SOAP message to the web service and to receive the response. The web client automatically creates and sends an HTTP request, but you can create a custom HTTP request. To do, you use the following procedure:

1. Create an instance of %Net.HttpRequest and set properties as needed. For information on this class, see *Using Internet Utilities* or the class documentation for %Net.HttpRequest.
2. Set the HttpRequest property of your web client equal to this instance.

This is useful in particular if you want to support multiple calls to a SOAP service within the same session. By default, the InterSystems IRIS web client does not support multiple calls to a SOAP service using the same session. To work around this, create a new instance of %Net.HttpRequest and use it as the HttpRequest property of your web client. This change forces the same HTTP request to be reused for all calls, which returns all cookies in a response to the next request.

22

Customizing Callbacks of a Web Client

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

%OnSOAPRequest()

```
Method %OnSOAPRequest(mode As %String,
                      client As %SOAP.WebClient,
                      action As %String,
                      oneWay As %Boolean,
                      method As %String,
                      requestStream As %BinaryStream)
```

Called just before the web client invokes the **DoSOAPRequest()** method of the transport class (which makes the actual SOAP request). The default DoSOAPRequest() method is included in %SOAP.WebClient and uses HTTP for request/response.

- *mode* specifies the type of SOAP request (`"SOAP"` or `"binary"`).
- *client* is the OREF of the web client instance.
- *action* contains the value of the SOAPAction header.
- *oneWay* is true if no body is to be sent.
- *method* argument is the name of the web method that is being invoked.
- *requestStream* argument contains the SOAP request message in a stream.

%OnSOAPResponse()

```
Method %OnSOAPResponse(mode As %String,
                       client As %SOAP.WebClient,
                       action As %String,
                       oneWay As %Boolean,
                       method As %String,
                       requestStream As %BinaryStream,
                       responseStream As %BinaryStream,
                       sc As %Status)
```

Called after the web client has invoked the **DoSOAPRequest()** method of the transport class. The *sc* argument is the status returned by the **DoSOAPRequest()** method of the transport class. The other arguments are the same as for **%OnSOAPRequest()**.

%OnSOAPFinished()

```
Method %OnSOAPFinished(mode As %String, client As %SOAP.WebClient, method As %String, sc As
%Status)
```

Called after the web client has performed all its processing. The *sc* argument is the status returned by the web method that was invoked. The *mode*, *client*, and *method* arguments are the same as for the other callback methods.

23

Specifying Custom Transport from a Web Client

By default, if you use an InterSystems IRIS® data platform web client, the web client uses HTTP to transport the SOAP message to the web service and to receive the response. You can define and use your own transport class.

23.1 Background

To communicate with the web service that it uses, an InterSystems IRIS web client requires a transport class. The transport class contains parameters, properties, and methods related to communication. The overall communication works as follows:

1. When a web client proxy method is run, the web client instance checks the value of its Transport property.

   If this property is null, the web client instance uses itself as the transport class instance. You can instead set the Transport property equal to an instance of some other suitable class, if you have defined such a class.

2. The web client instance executes the **DoSOAPRequest()** method of the transport class, passing the following arguments:

   a. The OREF of the web client class.

   b. A string that specifies the SOAP action.

   c. A stream containing the request encoded in UTF-8.

   d. (By reference) A stream containing the response.

3. The web client instance checks the status of the result and acts accordingly.

For HTTP transport, the **DoSOAPRequest()** method includes the following logic:

1. Create a request object (an instance of %Net.HttpRequest) and set its properties. Here, the method uses values of the properties of the web client instance, in particular HttpRequestHeaderCharset and other HTTP-related properties.
2. Go through the headers in the SOAP request and initialize the headers in the request object.
3. Execute the **Post()** method of the request object, which is a suitable action for HTTP transport.
4. Get the response and return that.

Important: Do not directly use the **DoSOAPRequest()** method of %SOAP.WebClient. No guarantee is made about its behavior or future operation. The preceding summary is provided only as a general tip.

23.2 Defining Custom Transport for an InterSystems IRIS Web Client

To enable an InterSystems IRIS web client to use custom transport, define a custom transport class. Then after you create an instance of the web client, set its Transport property equal to an instance of the transport class.

The requirements for the transport class are as follows:

- The class must be instantiable (that is, non-abstract).
- The class must implement the **DoSOAPRequest()** method as described below.

The **DoSOAPRequest()** method should transport the request to the web service and obtain the response. The signature of this method must be as follows:

```
Method DoSOAPRequest(webClient,action,requestStream, responseStream) As %Status
```

- *webClient* is the OREF of the web client class.
- *action* is a %String that specifies the SOAP action.
- *requestStream* is a stream containing the request encoded in UTF-8.
- *responseStream* is a %FileBinaryStream argument that **DoSOAPRequest()** uses to write the response. This stream must contain data in the character set specified by the encoding attribute of the `?xml` directive. UTF-8 is recommended.

24

Specifying Flags for the SAX Parser

When an InterSystems IRIS® data platform web client invokes a web service, it internally uses the SAX parser, a third-party product that is shipped with InterSystems IRIS. You can set the SAXFlags property of the web client in order to set the flags for the parser to use.

For information on the parser flags themselves, see *Using XML Tools*.

25

Using the WS-Security Login Feature

If your InterSystems IRIS® data platform web client is using a web service that requires authentication, and if do not want to use the newer WS-Security features, you can use the older and simpler WS-Security login feature.

To use the WS-Security login feature:

1. Ensure that you are using SSL between the web client and the web server that hosts the web service. The WS-Security header is sent in clear text, so this technique is not secure unless SSL is used. See Configuring the Client to Use SSL.
2. Invoke the **WSSecurityLogin()** method of the web client. This method accepts the username and password, generates a WS-Security username token with clear text password, and adds a WS-Security header to the web request.
3. Invoke the web method.

This technique adds the security token only to the next SOAP message.

For information on the newer WS-Security features, see *Securing Web Services*.

26

Using HTTP Authentication

Some web services require HTTP authentication instead of using WS-Security (which is described in *Securing Web Services*). For these web services, InterSystems IRIS® data platform supports the following HTTP authentication schemes:

1. Negotiate (SPNEGO and Kerberos, per RFC 4559 and RFC 4178)
2. NTLM (NT LAN Manager Authentication Protocol)
3. Basic (Basic Access Authentication as described in RFC 2617)

Note that on HTTP 1.0, only Basic authentication is used; the other authentication schemes require multiple round trips within a single connection, which is not permitted in HTTP 1.0.

To use HTTP authentication:

- Set the HttpUsername and HttpPassword properties of the web client before invoking the web method.
- If you want the client to send an initial header indicating the scheme to use (and you know that the server permits the scheme), set the HttpInitiateAuthentication property before invoking the web method. For the value of this property, specify an authentication scheme name, as given in Providing Login Credentials in Sending HTTP Requests.
- If you want to customize the list of schemes that the client tries, set the HttpInitiateAuthentication property before invoking the web method. For the value of the property, use a comma-separated list of names, as given in Providing Login Credentials in Sending HTTP Requests.

Important: If there is a chance that Basic authentication will be used, ensure that you are using SSL between the web client and the web server that hosts the web service. In Basic authentication, the credentials are sent in base-64 encoded form and thus can be easily read. See Configuring the Client to Use SSL.