



TIBCO® Data Virtualization

Web Based Data Sources Adapter Guide

Version 8.7.0 | October 2023

Contents

Contents	2
TDV Web-Based Data Sources	4
About OAuth Configuration for SOAP and REST Data Sources	4
About WSDL Bare and Wrapper Parameter Styles	5
WSDL and SOAP Data Sources	7
Adding a WSDL or SOAP Data Source	8
Enabling Tube Logs for SOAP Data Sources	14
Subscribe to a WSDL Offered as a JMS Data Source	14
Defining SOAP Message Pipelines	15
Message Level Security (Pipelines) for Legacy Web Services Imported from CAR Files	18
REST Data Sources	30
Adding a REST Data Source	31
Setting a Value for NULL JSON Values in REST Responses	42
Passing a Full XML Document in a POST Request to a REST Service	42
Using an HTTP Header Parameter to Specify the Content Type	43
Using Design By Example to Infer Parameter Data Types	44
Cross-Origin Resource Sharing (CORS)	47
Configuring XML/HTTP Data Sources	51
Creating an XML Definition Set	52
Adding an XML/HTTP Data Source	53
Retrieving XML Data Over HTTP	55
Proxy Configuration Limitations	56
Client Authentication for Web Data Sources	57
TDV SOAP and REST OAUTH Examples	58
Google OAuth Example	59
Facebook OAuth Example	61
Linkedin OAUTH Example	63

Github OAuth Example	64
Salesforce OAuth Example	65
Foursquare OAuth Example	67
TDV OAuth Tab XML Processors Field Reference	68
Authorization Element Reference	71
AccessToken XML Element Reference	75
RefreshToken Element	80
QueryTokenName Element	84
Partial Example of Using OAuth Customized Flow for WSDL, SOAP, and REST	84
TIBCO Product Documentation and Support Services	89
How to Access TIBCO Documentation	89
How to Contact TIBCO Support	90
Release Version Support	90
How to Join TIBCO Community	91
Legal and Third-Party Notices	92

TDV Web-Based Data Sources

This topic describes how to configure Web-based data sources including WSDL (Web Services Definition Language) and REST (Representational State Transfer) data sources.

- [About OAuth Configuration for SOAP and REST Data Sources](#)
- [About WSDL Bare and Wrapper Parameter Styles](#)
- [WSDL and SOAP Data Sources](#)
- [REST Data Sources](#)
- [Configuring XML/HTTP Data Sources](#)
- [Proxy Configuration Limitations](#)
- [Client Authentication for Web Data Sources](#)
- [TDV SOAP and REST OAUTH Examples](#)
- [TDV OAuth Tab XML Processors Field Reference](#)
- [Partial Example of Using OAuth Customized Flow for WSDL, SOAP, and REST](#)

About OAuth Configuration for SOAP and REST Data Sources

OAuth is a standard method for obtaining secure authorization from the Web. The OAuth authorization framework enables a third-party application to obtain limited access to an HTTP service. You are expected to be familiar with the OAuth 2.0 Authorization Framework (RFC 6749).

If 5 seconds is not the appropriate time for the data source to wait for a web page execution to occur, you can modify the Default OAuth Page Execution Timeout configuration parameter value.

The OAuth grant-flows between the client and the resource owner are:

- Authorization code grant—OAuth uses an authorization server as an intermediary to obtain an authorization code. The authorization server authenticates the resource

owner and obtains authorization. The resource owner's credentials are not shared with the client.

- **Implicit grant**—This simplified flow is optimized for browser clients using a scripting language. The client is issued an access token directly; the authorization server does not authenticate the client. However, the access token may be exposed to the resource owner or other applications with access to the resource server.
- **Resource owner password credentials grant**—The resource owner credentials (username and password) can be used directly as an authorization-code grant to obtain an access token. The credentials should only be used when there is a high degree of trust between the resource owner and the client, and when other authorization grant types are not available. With this grant type, the client can use a long-lived access token or refresh token instead of storing the resource owner credentials for future use.
- **Client credentials grant**—The client credentials (or other forms of client authentication) can be used as an authorization grant when the authorization scope is limited to protected resources either under the control of the client or previously arranged with the authorization server.
- **Custom Grant**—You specify the JavaScript or other process to use to obtain the access token to connect to the resource. For example, the client can request an access token using a Security Assertion Markup Language (SAML) 2.0 bearer assertion grant type.

About WSDL Bare and Wrapper Parameter Styles

WSDL bare and wrapped parameter styles control the consumption of data that is returned in a response. Typically if you have a parameter where you expect the response values to be small, you can put the parameter in the WSDL header. If, however, you expect there to be significant volumes of data returned in the response for the parameter, then it is best to place the parameter or set of parameters in the body of the WSDL.

TDV conforms to the open source standard for XML-based web services. Oracle provides useful reference content that describes the standards fully. This section attempt to summarize how TDV interprets those standards for using Bare and Wrapped parameters for your WSDL.

Style	Description
WRAPPED	<p>For multiple parameters that use the BODY location.</p> <p>The wrapper element is the child of the SOAP BODY and the parameter elements are children of the wrapper element.</p>
BARE	<p>For exactly one parameter with a location of BODY and its element is the only child of the SOAP BODY.</p> <p>All other parameters of the same direction must be mapped to another location, for example, HEADER.</p>

WRAPPED

A parameter style of wrapped means that all of the input parameters are wrapped into a single element on a request message and that all of the output parameters are wrapped into a single element in the response message. If you set the style to RPC you must use the wrapped parameter style. The wrapped style tells the Web service provider that the root element of the message represents the name of the operation and that children of the root element must map directly to parameters of the operation's signature.

With wrapped all sub-parameters are in the first level, and the client automatically wraps them in another top element.

Wrapped requests contain properties for each in and in/out non-header parameter. The properties for the method return values of each out non-header parameter, and in/out non-header parameter. The order of the properties in the request is the same as the order of parameters in the method signature. The order of the properties in the response is the property corresponding to the return value followed by the properties for the parameters in the same order as the parameters in the method signature.

Most Web services engines use positional parameter binding with wrapper style. If a service signature changes, clients must also change. With the wrapper style, all parameters have to be present in the XML message, even if the element corresponding to the parameter can be defined as optional in the schema.

The wrapped request:

- Must be named the same as the method and the wrapper response bean class must be named the same as the method with a “Response” suffix.

- Can only have one message part in WSDL, which guarantees that the message (method element with parameters) is represented by one XML document with a single schema.
- Must have parameters present in the XML message, even if the element corresponding to the parameter can be defined as optional in the schema.

BARE

A parameter style of BARE means that each parameter is placed into the message body as a child element of the message root. With BARE there is only a top element parameter with sub-parameters inside. This one BARE parameter is sent directly.

Whether a given SOAP message is valid is determined by the WSDL that it conforms to. Web services using the bare style can have multiple parameters. It is only the actual invocation of the web service where a single parameter is passed.

The Java API for XML-based Web Services requires that parameters for bare mapping must meet all the following criteria:

- It must have at most one in or in/out non-header parameter.
- If it has a return type other than void it must have no in/out or out non-header parameters.
- If it has a return type of void it must have at most one in/out or out non-header parameter.

Adding a new optional element does not affect clients, because binding is always name-based. To get around having to have a contract defined by the schema of the web service, you can define all child elements of the wrapper root element as required. Optional elements must be made nillable.

WSDL and SOAP Data Sources

Web Services Description Language (WSDL) is an XML-based language that describes a Web service. A WSDL file specifies how clients can submit inputs to an operation and what kind of output the client can expect in return.

Web services are Web-based applications that dynamically interact with other Web applications using an XML message protocol such as SOAP 1.1 or 1.2. Web services can be “bound” to any message format and protocol, but three bindings are popular: SOAP, HTTP,

and MIME. TDV supports the following profiles for SOAP binding: SOAP over HTTP, SOAP over WSIF JMS, and SOAP over TIBCO JMS. The binding profile allows specification of the transport protocol, encoding scheme, and message style.

The transport protocol defines what mechanism is used for transporting the request.

- TDV supports HTTP and JMS transport.

The encoding scheme defines how the message is encoded for the transport.

- TDV supports the literal encoding scheme, which uses an XML schema as the definition for how data should be encoded.
- TDV also provides limited support for the SOAP encoding scheme as defined in the SOAP specification.
- TDV does not support multi-dimensional or sparse SOAP arrays.

The message style refers to how the request itself is structured.

- TDV supports the Document Literal message style for WSDL and SOAP. Document Literal style messages have a single part whose schema defines the message payload.

This section contains the following topics:

- [Adding a WSDL or SOAP Data Source](#)
- [Enabling Tube Logs for SOAP Data Sources](#)
- [Subscribe to a WSDL Offered as a JMS Data Source](#)
- [Defining SOAP Message Pipelines](#)
- [Message Level Security \(Pipelines\) for Legacy Web Services Imported from CAR Files](#)

Adding a WSDL or SOAP Data Source

This section describes how to add a WSDL or SOAP data source. Redefining existing WSDL data sources as SOAP data sources might increase your success in integrating those data sources with TDV and other client applications.

For more information on adding data sources, Refer the *User Guide*, section *Adding a Data Source*.

Refer the *User Guide*, Chapter *Retrieving Data Source Metadata* for how to introspect a data source.

To add a WSDL or SOAP data source

In Studio, right-click the folder in which you want to add the WSDL or SOAP data source and choose New Data Source.

1. Select WSDL or SOAP as the data source adapter and click **Next**.
2. For Name, enter a name for your data source.
3. On the Basic tab, provide this information for a data source.

Field	Description
URL	<p>URL to the WSDL or SOAP data source. WSDLs can be available by URL over any accessible network. A locally mapped WSDL can be introspected using a URL format like the following:</p> <pre>file:///Z:/test.wsdl</pre>
Login	Optionally, provide a valid username to the data source.
Password	Optionally, provide a password to the data source.
Save Password	This check box is enabled only if Pass-through Login is enabled. Refer the <i>User Guide</i> , Section <i>About Pass-Through Login</i> for more information.
Pass-through Login	<p>Disabled—The default setting. This allows automated provisioning of a connection pool. If Pass-through Login is disabled, the Save Password check box is not available.</p> <p>Enabled—A new connection to the data source uses the credentials supplied by the client when data is requested from that data source for the first time. If Pass-through Login is enabled, the Save Password check box becomes available.</p> <p>Refer the <i>User Guide</i>, Chapter <i>About Pass-Through Login</i> for more information.</p>
Authentication	<p>Choose the method of authentication for this data source: BASIC, NTLM, or NEGOTIATE, OAuth, Digest.</p> <p>When selecting OAuth as the authentication mode, another tab will</p>

Field	Description
	display. Authentication for the data source must be designated as OAuth 2.0 when the physical data source was first added.
Domain	For NTLM authentication only, enter the domain. See “Configuring NTLM Authentication” in the <i>TDV Administration Guide</i> for more information.
Service Principal Name	For NEGOTIATE authentication with Kerberos only, enter the service principal name. See “Configuring Kerberos Single Sign-On” in the <i>TDV Administration Guide</i> for more information.
SAML Header Class	SAML assertions are defined in the headers of a class. Type the class name that owns the SAML header.

4. If the data source requires client authentication, click the Advanced tab. See [Client Authentication for Web Data Sources](#) for how to configure client authentication.

If the data source requires OAuth, select the OAuth 2.0 tab. Specify the values appropriate to the OAuth flow you want to use. For examples, see [TDV SOAP and REST OAUTH Examples](#). (This tab and the fields are available to edit after creation of the data source.) The following table describes the values the user is to provide on the OAuth 2.0 tab:

Field	Description
OAuth Flow	These OAuth flows: <ul style="list-style-type: none"> • AUTHORIZATION_CODE • IMPLICIT—Client Secret and Access Token URI are disabled. • CLIENT_CREDENTIALS—Resource Owner Authentication fields are disabled. • RESOURCE_OWNER_PASSWORD_CREDENTIALS—Client Authentication fields are disabled. • CUSTOMIZED—User-specified flow.

Field	Description
Client Identification	Used in the request-body of token requests. A unique string representing the identifier issued to the client during registration. It is exposed to the resource owner. Format: string of printable characters.
Client Secret	Used in the request-body of token requests. Enabled only for AUTHORIZATION_CODE, and OAuth flow. Format: string of printable characters.
Authorization URI	URI to use for establishing trust and obtaining the required client properties.
Access Token URI	URI to use for communicating access and refresh tokens to the client or resource server. Disabled only for IMPLICIT OAuth flow.
Redirect URI	Client's redirection end point, established during client registration or when making an authorization request. Must be an absolute URI, and must not contain a fragment component. The authorization server redirects the resource owner to this end point.
Scope	Authorization scope, which is typically limited to the protected resources under the control of the client or as arranged with the authorization server. Limited scope is necessary for an authorization grant made using client credentials or other forms of client authentication. Format: one or more strings, separated by spaces.
State	A request parameter used in lieu of a complete redirection URI (if that is not available), or to achieve per-request customization
Expiration Time (Sec)	The lifetime of the access token.
Use Refresh Token To Get Access Token	Checking this box enables the use of refresh tokens to obtain access tokens, rather than obtaining them manually.
Login and Password	User credentials with which the client or resource owner registers with the authentication server before initiating the OAuth protocol.

Field	Description
Pass-through Login	Enabled or Disabled
Authentication	BASIC, DIGEST, NTLM, or NEGOTIATE—The usual collection of authentication methods, used for registration with the authentication server.
Domain	Domain to which the client or resource owner belongs; for example, composite.
Service Principal Name	SPN of the client or resource owner.
Access Token Type	<p>Bearer—User of a bearer token does not need to prove possession of cryptographic key material. Simple inclusion of the access token in the request is sufficient.</p> <p>Query—The query string “?access_token=<token>” is appended to the URL. Not available for SOAP data sources.</p>
Access Token	Credentials used to access protected resources, with a specific scope and duration. Usually opaque to the client.
Get Token button	Initiates acquisition of an access token. Proper information must be configured for this request to succeed.
Refresh Token	Credentials used to obtain access tokens when they become invalid or have expired. Intended for use with authorization servers, not resource servers.
Custom Flow	The name of the custom flow for a data source with an OAuth flow of CUSTOMIZED.
Using Processors check box and editable text field	Check this box to use processors. The editable text field allows you to enter JavaScript and XML. You can use this field to add JavaScripts that log in automatically or use XML to customize any part of the authorization or access token that does not conform to specifications.

Field	Description
	<h3>Editable text field</h3> <p>The editable text field underneath the Using Processors check box can be used to type the XML elements necessary to establish authorization and access tokens. For example:</p> <pre><Authorization> <AuthorizationProcessors> <AuthorizationProcessor> document.getElementById ('email').value='queenbeeza@gmail.com'; document.getElementById('pass').value='jellypassword'; document.getElementById('loginbutton').click(); </AuthorizationProcessor> </AuthorizationProcessors> </Authorization></pre> <pre><AccessToken> <RequestMsgStyle>QUERY</RequestMsgStyle> <ResponseMsgStyle>FORM</ResponseMsgStyle> <ExpireTime>1000</ExpireTime> </AccessToken></pre>
Sensitive Keyword in JavaScript	<p>Click the plus-sign icon one or more times to add tag-keyword pairs to substitute into the JavaScript specified for a custom authorization flow. The values sent to the server are encrypted, and then replaced with their decrypted values where the tags are found in the JavaScript.</p> <p>These pairs are used for the user name, email ID, password, and other sensitive information.</p> <p>Tag—The name of the tag to find in the JavaScript.</p> <p>Keyword—The encrypted value to decrypt and substitute for the tag in the JavaScript.</p>

5. Click Create & Introspect to initiate introspection.

The Data Source Introspection dialog displays, for the specified WSDL, the available services, ports, and operations. Expand the nodes of the Web service and ports to see the operations within each service.

Enabling Tube Logs for SOAP Data Sources

A tube is a basic SOAP message processing unit. You can determine when to collect log information on those packets of information using the TDV configuration parameters.

To enable tube logs for SOAP data sources

Open and log into Studio.

1. From the Administration menu, choose Configuration.
2. Search for or select one of the following parameters (under Data Sources > SOAP Sources):

Parameter	Description
Enable Tube log after executing	Enable logging of SOAP messages or set to ALL for all tubes. Valid values are Terminal, Handler, Validation, MustUnderstand, Monitoring, Addressing, At, Rm, Mc, Security, PacketFilter, Transport, and Customized.
Enable Tube log before executing	Enable logging of SOAP messages or set to ALL for all tubes. Valid values are Terminal, Handler, Validation, MustUnderstand, Monitoring, Addressing, At, Rm, Mc, Security, PacketFilter, Transport, and Customized.

3. Click OK to save your changes and exit the window.

Subscribe to a WSDL Offered as a JMS Data Source

Introspection properties are available for both JMS and HTTP ports.

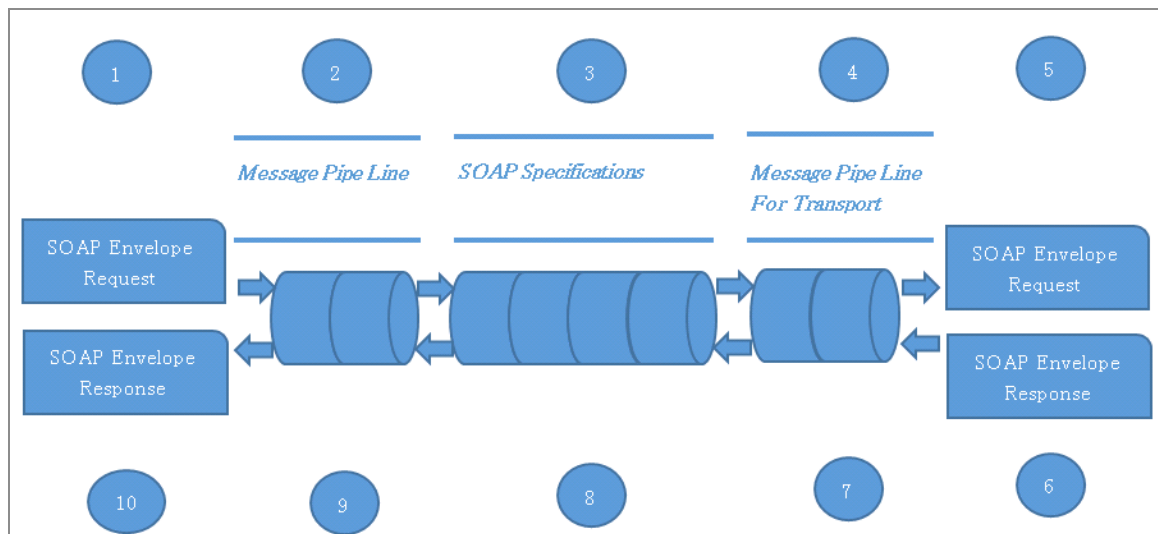
- For HTTP ports and operations only, you can specify the timeout parameter in milliseconds, or type zero for no timeout, or leave the entry null for an operation where the port setting is to take precedence.
- For JMS ports:
 - The Connector should be specified to ensure functionality of the JMS data source connection pools. JMS connectors should be installed by an administrator. See “Configuring TDV for Using a JMS Broker” in the *TDV Administration Guide*.
 - The specified JMS Destination can be changed to take advantage of different queue destination aliases that offer the same service.
 - Delivery Mode can be set to persistent so that messages are written to disk as a safeguard against broker failure. Non-persistent messages are not written to disk before acknowledgment of receipt.
 - Message Expiry specifies the period of message validity in the broker queue. An entry of 0 specifies no expiration, while a null entry for an operation specifies that the port setting is to take precedence.
 - Operations or messaging priority can be set to an integer of 1 through 9, where 9 is the highest priority.
 - Default Timeout is a setting for the consuming client, and it can be set to some duration (in milliseconds). An entry of zero means no timeout, and a null entry specifies that the default takes precedence.
 - Individual JMS operations under the port can be configured with a Message Type of Bytes or Text and with specific time-outs tailored to the operation.

If you need to review or change configurations for a Web service, open the Web service from Studio, click the Add/Remove Resources button, make required changes, and reintrospect the data source. For more details about introspection, refer the *User Guide*, Chapter *Retrieving Data Source Metadata*.

Defining SOAP Message Pipelines

The TDV SOAP data source has a Message Pipeline panel that can be used to configure how request and response messages are handled.

What Happens When a SOAP Message Pipeline is Run

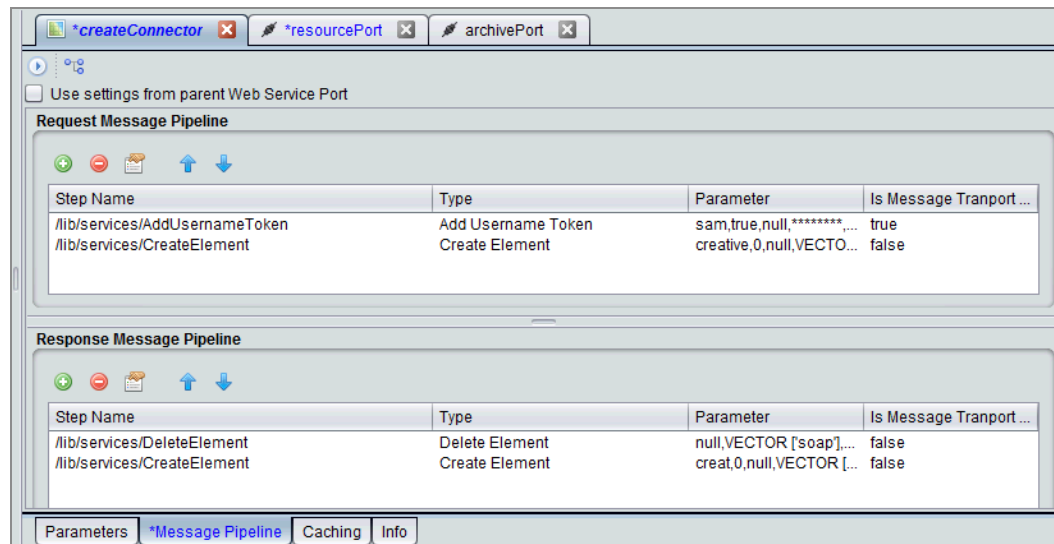


1. Create a SOAP request by executing the operation.	6. The SOAP response message is received from the service side.
2. Message pipelines that are defined as request.	7. Message pipelines that are defined as response.
3. The SOAP request message is handled according to the SOAP specifications that are defined in the contract.	8. The SOAP response message is handled according to the SOAP specifications that are defined in the contract.
4. Message pipelines that are defined as request..	9. Message pipelines that are defined as response.
5. The SOAP request message is sent to the service side.	10. The SOAP message is received.

To define a SOAP Message Pipeline

After creating and introspecting your SOAP data source, open the editor for the port or operation for which you want to define the message pipeline.

1. Select the Message Pipeline tab.



2. Use the green plus buttons to add steps to the Request Message Pipeline or Response Message Pipeline. The following types can be added:

Request Message Step Types	Response Message Step Types
Add Username Token	Create Element
Create Element	Custom
Custom	Delete Element
Delete Element	Log Message to File
Encrypt Element	Process Security Header
Log Message to File	Set Environment From Node
Set Environment From Node	Set Node From Environment
Set Node From Environment	
Sign Element	

3. Follow the prompts on the screens for what is required for each of the different step types.
4. You can delete, edit, and reorder the steps at anytime after adding them.

Message Level Security (Pipelines) for Legacy Web Services Imported from CAR Files

If messaging passes through intermediate sources in the transport layer (indirect messaging), you must define message-level security. For indirect messaging, or for multiple message receivers or senders, the message needs to be secured at different levels to ensure data integrity.

A pipeline defines multiple instructions that are processed simultaneously. Except for the Custom step, each pipeline step corresponds to a system built-in procedure available at `/lib/services/` in the server.

- [Viewing Message Pipeline Steps](#)
- [Creating a Log Message to File Pipeline Step](#)
- [Adding a Username Token Pipeline Step](#)
- [Creating an Element Pipeline Step](#)
- [Creating a Pipeline Step to Process a Procedure](#)
- [Creating a Pipeline Step to Delete an Element](#)
- [Creating a Pipeline Step to Encrypt an Element](#)
- [Creating a Pipeline Step to Process a Security Header](#)
- [Creating a Pipeline Step to Set Environment From Node](#)
- [Creating a Pipeline Step to Set Node From Environment](#)
- [Creating a Pipeline Step to Sign an Element](#)

Viewing Message Pipeline Steps

You can provide message-level security by defining multiple steps of well-defined pipeline processing for request-response messages sent through Studio. Specifically, you can provide message-level security at the following locations of a SOAP or WSDL data source:

- SOAP operation
Operation-level security settings override the security settings of the port.
- WSDL operation
Operation-level security settings override the security settings of the port.

The message pipeline editor has the following overall characteristics:

- The Request Message Pipeline and Response Message Pipeline sections allow you to add message processing steps to the pipeline.
- You can click the upper Add button to view a drop-down list of the pipeline steps available to process the Request Message Pipeline:
Create Element, Custom, Delete Element, Log Message To File, Process Security Header, Set Environment From Node, Set Node From Environment
- You can click the lower Add button to view a drop-down list of the available Response Message Pipeline steps:
Add Username Token, Create Element, Custom, Delete Element, Encrypt Element, Log Message To File, Set Environment From Node, Set Node From Environment, Sign Element
- You can click a Delete button to remove a step.
- You can click an Edit button to change what you specified when you added the step.
- You can change the processing order of a step by selecting it and clicking the Move up or Move down button.

To view the available pipeline steps in Studio

Open the operations folder for the Web service that is to be secured.

1. In the editor that opens on the right, the Message Pipeline panel opens.
2. Click the Add button to view the pipeline steps in a drop-down list.

Creating a Log Message to File Pipeline Step

This pipeline step writes SOAP message contents to a file in the specified path. Create this pipeline step for a request message, response message, or both. The outputs from other pipeline steps can be written to the log file created by this step.

This pipeline step corresponds to the system procedure `LogMessageToFile` available in `/lib/services/`.

To create the pipeline step named Log Message To File

Open the pipeline editor for the resource that is to be secured.

3. Click the **Add** button in the Request Message Pipeline section or the Response Message Pipeline, as needed, and select **Log Message To File**.
4. In the File Path field, specify a file where the messages are to be logged. The file path is relative to the TDV Server log directory. If the file or the directory does not yet exist, it is created on the local file system of the server on execution.
5. In the File Mode field, specify how the message should be logged.
APPEND—Adds new messages to the end of the log file.
OVERWRITE—Causes new messages to overwrite the log file.
6. (optional) Text entered into the Header field is added to the given text in front of the new SOAP envelope message. The text supplies a header note, which is written to the file right before the message contents. This value can be null.
7. (optional) Text entered into the Footer field is added to the end of the processed message. This value can be null.
8. In the Pretty Print drop-down list, select **true** (default) if you want the message to be formatted with tabbed indents, and **false** if you do not want the message to be formatted.

Formatting the message can make it easier to read.

9. Click **OK**, and save the step.

The output of this pipeline step is the modified XML document or element. For example:

Adding a Username Token Pipeline Step

You can use this pipeline step to authenticate a response message. If the server hosting the Web service requires that the message has a username token, use this pipeline step.

This pipeline step adds a WS-Security `UsernameToken` to a SOAP envelope. The `UsernameToken` is added to the SOAP header that is identified by the Actor and Must Understand arguments. If the SOAP message does not contain a SOAP header with the specified Actor and Must Understand values, a header is created with those values.

This pipeline step corresponds to the system procedure `AddUsernameToken`, which is available in `/lib/services/`.

To create the pipeline step named Add Username Token

Open the pipeline editor for the resource that is to be secured.

10. Click the Add button in the Request Message Pipeline section, and select Add Username Token.

11. Values for the following fields need to be supplied in the Add Username Token window:

Field	Description of values	Example
Actor	<p>Type a Uniform Resource Identifier (URI) used to specify the recipient of a header element.</p> <p>Specifies the SOAP <code>Actor</code> attribute. If it is null, the recipient is the ultimate destination of the SOAP message.</p> <p>The SOAP actor attribute can be used to address the Header element to a particular endpoint. A SOAP message can travel from a sender to a receiver by passing different endpoints along the message path. Not all parts of the SOAP message are intended for the ultimate endpoint of the SOAP message but, instead, are intended for one or more of the endpoints on the message path.</p>	<p>http://www.w3schools.com/appml</p> <p>Actor1</p>
Must Understand	<p>Select true or false.</p> <p>Indicates whether a header entry is mandatory or optional for the recipient to process.</p> <p>If you specify Must Understand=true to a child element of the Header element, it indicates that the receiver processing the Header must recognize the element. If the receiver does not recognize the element it must fail when processing the</p>	true

Field	Description of values	Example
	Header. If you specify Must Understand=false to a child element of the Header element, it indicates that the receiver processing the Header need not recognize the element.	
Username	Valid user name to access the Web service server.	joeuser
Password	Valid password associated with Username.	password
Password Type	Specify the password type, to determine how the password is encoded in the Username Token: DIGEST—Password is rendered in a digested text in the message TEXT—Password is rendered in clear text in the message	DIGEST

12. Click **OK** after supplying all the required information, and save the step.

The `UsernameToken` is added to the SOAP header that is identified by the entry supplied in the Actor field and Must Understand field. If the SOAP message does not contain a SOAP header with the specified Actor and Must Understand values, a header is created with those values.

The following shows sample output:

The digested password is rendered as follows:

```
<wsse:Password
```

```
Type="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-username-token-profile-1.0#PasswordDigest"
```

```
xmlns:wss="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-
wssecurity-secext-1.0.xsd">APy0l8XZkRsJH7qiAwC6W11D+gs=</wss:Password>
```

Creating an Element Pipeline Step

This pipeline step creates a child element in an XML document or within another element.

This pipeline step corresponds to the system procedure `CreateElement`, which is available in

```
/lib/services/
```

To create the pipeline step named Create Element

Open the pipeline editor for the resource that is to be secured.

13. Click the Add button in the Request Message Pipeline section, and select Create Element.
14. Supply the fully qualified name of the element in the Element Name field.
This value cannot be null.
15. Specify the position of the element in the Element Position field.
This position is relative to the element's siblings. The default value is 0 (zero). A value of zero indicates that the element should be created before any existing children. A value of -1 (negative one) indicates that the element should be created after all existing children.
16. Type the path to the parent of the element in the Parent XPath field.
The Parent XPath value is used to select the parent element, relative to the root of the element to be created. This entry cannot be null.
17. In the Prefix field, specify the namespace prefix used in the Parent XPath field.
18. In the Namespace field, specify the namespace URIs used in the Parent XPath expression.
19. Optionally, click the Add button next to Declare namespaces used in XPath expressions to define an additional prefix-namespace pair.
20. Optionally, click the Delete button next to a prefix-namespace pair to delete the pair.

21. Click **OK**.

Creating a Pipeline Step to Process a Procedure

This pipeline step processes a custom procedure supplied to it. The signature of a custom procedure supplied to a message pipeline should follow these rules:

- It must not have any OUT parameters.
- It must have only IN or INOUT parameter. There can be more than one IN or INOUT parameter in the signature.
- The first parameter must be of type XML, which can be defined as an IN or INOUT parameter.

The example here uses the following custom procedure (`CustomPipelineStep`) which calls the system procedure `/lib/debug/Log`:

```
PROCEDURE CustomPipelineStep(IN document XML, IN param1 INT)
```

```
BEGIN
```

```
CALL /lib/debug/Log(CAST(document AS VARCHAR(4096)));
```

```
END
```

The first parameter is an input parameter of type XML.

To create the pipeline step named Custom

Open the pipeline editor for the resource that is to be secured.

22. Click the Add button in the Request Message Pipeline or Response Message Pipeline section, and select Custom.
23. Use the Browse button to locate and select the custom pipeline procedure.
24. Supply the input parameter value in the Parameter Values field.
25. Optionally, select a parameter value and click Edit to change the value.
26. Click **OK**.

The following shows a sample log message:

Creating a Pipeline Step to Delete an Element

This pipeline step deletes one or more nodes from an XML document or element.

The example used here deletes the Header element named `MyCustomHeader` which was created in the section [Creating an Element Pipeline Step](#).

This pipeline step corresponds to the system procedure `DeleteElement`, which is available in */lib/services/*.

To delete one or more nodes from an XML document or element

Open the pipeline editor for the resource that is to be secured.

27. Click the Add button in the Request Message Pipeline or Response Message Pipeline section, and select Delete Element. Type the path to the parent of the element in the Parent XPath field.

This path is used to select the nodes to be deleted. The path is evaluated against the root node. All resulting nodes are deleted. This entry should not contain any namespaces.

28. Optionally, click the Add button next to Declare namespaces used in XPath expressions to define an additional prefix-namespace pair.
29. Optionally, click the Delete button next to a prefix-namespace pair to delete the pair.
30. In the Prefix field, specify the namespace prefix used in the XPath field.
31. In the Namespace field, specify the namespace URIs used in the XPath field.
32. Click **OK** and save the step.

Creating a Pipeline Step to Encrypt an Element

This pipeline step is used to encrypt the BODY element in the specified SOAP envelope using a symmetric key that is encrypted by a certificate or public key.

This pipeline step corresponds to the system procedure `EncryptElement`, which is available in */lib/services/*.

To create the pipeline step named Encrypt Element

Open the pipeline editor for the resource that is to be secured.

33. Click the Add button in the Response Message Pipeline section, and select Encrypt Element.

UI Element	Description
Procedure path	The path to the pipeline step procedure (by default, <i>/lib/services/EncryptElement</i>).
Actor	Type a URI. See To create the pipeline step named Add Username Token .
Must Understand	Select true or false.
Element Name	<p>The default value of the Element Name field specifies the schema of the SOAP message that is encrypted. This procedure can be used to encrypt the SOAP message body.</p> <p>The Element Name field can be null, but the default value of</p> <pre>{http://schemas.xmlsoap.org/soap/envelope/}Body</pre> <p>is used to specify the message body for encryption:</p> <pre>{http://schemas.xmlsoap.org/soap/envelope/}Body</pre>
Encryption Algorithm	<p>Accept the default algorithm AES_128 or select a different one if you have installed an unrestricted Java Cryptography Extension (JCE) policy file in the server's JVM.</p> <p>Determines the method of encryption. The default value of AES_128 is sufficient for most purposes. Stronger encryption algorithms such as AES_192 or AES_256 require that an unrestricted Java Cryptography Extension (JCE) policy file be installed in the server's JVM.</p>
Certificate Alias	<p>Select or type an appropriate certificate alias.</p> <p>If the WSDL data source or Web service has a list of certificates associated with it, the aliases of those certificates are listed here.</p> <p>The alias of a certificate or public key in the key store that is used to</p>

UI Element	Description
	encrypt the symmetric key that is used to encrypt the element. It cannot be null.
	Certificates are associated with a data source using the Data Source connection properties. To access the certificates, open the editor for the WSDL data source and click the Advanced tab in the Connection Information section. Any Alias listed in the Certificates table can be used as the Certificate Alias in a pipeline step.

34. Supply all required values and click OK.

Creating a Pipeline Step to Process a Security Header

This pipeline step is used to process a WS Security SOAP header in a SOAP envelope, as follows:

- If the envelope contains a WS Security header with the specified actor, the header is processed.
- All security elements in the header are evaluated.
- If any header security element indicates that the envelope contains signed elements, the signatures of those elements are verified.
- If any header security element indicates that the envelope contains encrypted elements, those encrypted elements are decrypted.

This pipeline step corresponds to the system procedure `ProcessSecurityHeader`, which is available in `/lib/services/`.

To create the pipeline step named Process Security Header

Open the pipeline editor for the resource that is to be secured.

35. Click the Add button in the Request Message Pipeline section, and select Process Security Header.
36. Supply the path to the pipeline step procedure if it is different from `/lib/services/ProcessSecurityHeader`.
37. Supply a value in the Actor field, if necessary.

The Actor determines which WS Security header to process. It can be null.

38. Click **OK** to save the step.

Creating a Pipeline Step to Set Environment From Node

This pipeline step saves an element or attribute value as an environment variable. It evaluates the given xpath expression against the envelope, and stores the result in the environment in a variable with the specified name. The result of the xpath expression is interpreted as a single string.

This pipeline step corresponds to the system procedure `SetEnvironmentFromNodeValue`, which is available in `/lib/services/`.

To create the pipeline step named Set Environment From Node

In the pipeline editor for the resource that is to be secured, click the Add button in the Request Message Pipeline or Response Message Pipeline section and select Set Environment From Node.

39. Type the path and step name in the Procedure Path field if it is different from `/lib/services/SetEnvironmentFromNodeValue`.
40. Type the XPath value in the XPath field.
The XPath entry is evaluated to some text which is stored in the variable name provided in the Variable Name field.
41. Type the variable name in the Variable Name field, without any spaces in the name string.
Variable Name is the name of an environment variable. It is an arbitrary string, and it is not case-sensitive. (Both `sample` and `SAMPLE` are considered the same.)
42. In the Prefix field, specify the namespace prefix used in the XPath field.
43. In the Namespace field, specify the namespace URIs used in the XPath expression.
44. Optionally, click the Add button next to Declare namespaces used in XPath expressions to define an additional prefix-namespace pair.
45. Optionally, click the Delete button next to a prefix-namespace pair to delete the pair.
46. Click **OK**.

Creating a Pipeline Step to Set Node From Environment

This pipeline step sets an element or attribute value from an environment variable. The given xpath expression is used to select a node from the envelope. The node value is then set to the value of the specified environment variable.

This pipeline step corresponds to the system procedure `SetNodeValueFromEnvironment`, which is available in `/lib/services/`.

To create the pipeline step named Set Node From Environment

In the pipeline editor for the resource that is to be secured, click the Add button in the Request Message Pipeline or Response Message Pipeline section and select Set Node From Environment.

47. Type the path and step name in the Procedure Path field if it is different from `/lib/services/SetNodeValueFromEnvironment`.

48. Type the XPath value in the XPath field.

The XPath expression is evaluated to an element.

49. Type the name of an environment variable in the Variable Name field, without a space in the name string.

Variable Name is an arbitrary string, and is not case-sensitive. (Both `sample` and `SAMPLE` are considered the same.)

50. In the Attribute Name field, specify the attribute whose value is to be set from the environment variable.

51. In the Prefix field, specify the namespace prefix used in the XPath field.

52. In the Namespace field, specify the namespace URIs used in the XPath expression.

53. Optionally, click the Add button next to Declare namespaces used in XPath expressions to define an additional prefix-namespace pair.

54. Optionally, click the Delete button next to a prefix-namespace pair to delete the pair.

55. Click **OK**, and save the step.

Creating a Pipeline Step to Sign an Element

This pipeline step is the simplified version of `EncryptElement`, and is used to sign an element in the specified SOAP envelope using a private key.

This pipeline step corresponds to the system procedure `SignElement` which is available in `/lib/services/`.

To create the pipeline step named Sign Element

In the pipeline editor for the resource that is to be secured, click the Add button in the Response Message Pipeline section and select Sign Element.

56. Type the path and step name in the Procedure Path field if it is different from `/lib/services/SignElement`.

57. For details on Actor, Must Understand, Element Name, and Certificate Alias, see [To create the pipeline step named Encrypt Element](#).

58. Supply all required values and click **OK**.

REST Data Sources

Representational State Transfer (REST) defines a set of architectural principles by which you can design Web services that focus on systems' resources and how these resource states are addressed and transferred over HTTP by client applications written in different languages.

The REST implementation in TDV establishes a one-to-one mapping between these basic operations and HTTP methods.

Operation	HTTP/REST	SQL Equivalent
Create a resource on the server	POST	INSERT
Retrieve a resource	GET	SELECT
Update or change a resource state	PUT	UPDATE
Remove or delete a resource	DELETE	DELETE

When you define a REST data source, you define its URL connection information and its operations which include the input and output parameters. After definition, the REST data source in Studio contains a Web Service Operation for each defined REST operation. The

Web Service Operations in Studio are similar to a stored procedure and can be used in the same way. Refer the *User Guide*, Chapter *Procedures* for more information.

This section contains the following topics:

- [Adding a REST Data Source](#)
- [Setting a Value for NULL JSON Values in REST Responses](#)
- [Passing a Full XML Document in a POST Request to a REST Service](#)
- [Using an HTTP Header Parameter to Specify the Content Type](#)
- [Using Design By Example to Infer Parameter Data Types](#)
- [Cross-Origin Resource Sharing \(CORS\)](#)

Adding a REST Data Source

This section describes how to add a REST data source.

About POST and Multi-Part/Form

Multi-Part/Form supports the transfer of multiple binary types of data for POST operations. For example, you could use this to transfer multiple GIF files.

- For POST, the Multi-Part/Form option is enabled for an HTTP 'multipart/form-data' request.
- If the Multi-Part/Form option is chosen, each parameter that uses Body and 'IN' or 'IN/OUT' will be consumed as a string part of the request.
- If the Content-type parameter is defined for an HTTP header then the charset defined in the parameter will be applied to all parts for the request. The default charset is ISO-8859-1.

To add a REST data source

In Studio, right-click the folder in which you want to add the REST data source and choose New Data Source.

1. Select REST as the data source adapter and click Next.

Studio displays the New Physical Data Source dialog for you to define the REST data source.

2. For Name, enter a name for your REST data source.
3. On the Basic tab, provide this information for a REST data source:

Field	Description
Base URL	<p>The base URL to access the REST data source. This is in the form:</p> <p style="text-align: center;">http://<web site name></p> <p>Example: http://search.twitter.com</p>
Use Metadata	Check this option if you want to Introspect an Open API metadata document.
Metadata URL	<p>When the Use Metadata option is selected, this field needs to be filled. Enter the location of the OpenAPI Metadata document that will be introspected.</p> <p>Note:- Only primitive datatypes (non-structured) are supported.</p>
Login	Optionally, provide a valid username to access the REST data source.
Password	Optionally, provide a valid password to access the REST data source.
Save Password	This check box is enabled only if Pass-through Login is enabled.
Pass-through Login	<p>Disabled—The default setting. This allows automated provisioning of a connection pool. If Pass-through Login is disabled, the Save Password check box is not available.</p> <p>Enabled—A new connection to the data source uses the credentials supplied by the client when data is requested from that data source for the first time. If Pass-through Login is enabled, the Save Password check box becomes available.</p> <p>Refer the <i>User Guide</i>, Section <i>About Pass-Through Login</i>, for more information.</p>
Authentication	Choose the method of authentication for this data source: BASIC, NTLM, or NEGOTIATE, OAuth, Digest.

Field	Description
	When selecting OAuth as the authentication mode, another tab will display. Authentication for the data source must be designated as OAuth 2.0 when the physical data source was first added. For more information, see Client Authentication for Web Data Sources .
Domain	<p>For NTLM authentication only. Enter the domain name.</p> <p>See “Configuring NTLM Authentication” in the <i>TDV Administration Guide</i> for more information.</p>
Service Principal Name	<p>For NEGOTIATE authentication with Kerberos only: enter the service principal name.</p> <p>See “Configuring Kerberos Single Sign-On” in the <i>TDV Administration Guide</i> for more information.</p>
Access Token	For OAUTH 2.0 authentication, type the access token.
Automatically Retrieve Access Token	Retrieve the access token automatically from the access token URL we mentioned in the OAuth tab
JSON Format	Check this box if you want to use the JSON (JavaScript Object Notation) standard for data interchange. When this box is checked, each HTTP request-response pair is encoded and decoded in the JSON format.
BadgerFish Enabled	<p>Check this box to enable BadgerFish if the REST services outside of TDV use the BadgerFish convention to translate XML into the JSON format.</p> <p>JSON Format must also be checked.</p>
Primitive Value Format	<p>Check this box to read and send the value in its primitive presentation.</p> <p>JSON Format must also be checked.</p>

Field	Description
Package Name	Prefix for each element to make the service name unique. The package name plays the same role as namespace in the XML format. Package name can consist of any characters that are legal in Java class names. JSON Format must also be checked.
Wrapper of JSON Bare Response	Type in the wrapper name. It can be the wrapper key of the whole JSON response. This value makes it possible to convert JSON responses to well-formed XML value types. JSON Format must also be checked.

- On the bottom part of the Basic tab, define the REST operations.
Under Operations, click the Add Operation button to add an operation.
In the New Operation dialog box, enter a name for the operation and click OK.
For each operation, you must define an HTTP Verb and Parse an Operation URL.

Field	Description
HTTP Verb	Choose the operation type: GET, POST, PUT, or DELETE.
Operation Name	Automatically filled in with the operation name you entered and selected in the Operations box.
Operation URL	Enter the URL for the REST operation in the format: <div style="background-color: #e6f2ff; padding: 10px; margin: 10px 0;"> <code><operation_api_name>?<query_parameters></code> </div> <ul style="list-style-type: none"> <code><operation_api_name></code>—The API service operation name as defined by the URL. Examples: addfeed.atom and search.atom from Twitter. You need to know the operation names for your Web data source and follow their conventions. <code><query_parameters></code>—Separate multiple entries with & (ampersand). Can be one or more of the following: <code><constant></code>—Define one or more constants in the form

Field	Description
	<p>constant=value. Example: count=20</p> <ul style="list-style-type: none"> • <{URL_parameter}>—Define one or more URL parameters in curly brackets. Example: tweet={mytweet} <p>At execution time, the Base URL is combined with the Operation URL defined for each operation to form the data request.</p>

Parse	<p>Click Parse to add all URL parameters specified in curly brackets in the Operation URL to the URL Parameters list.</p> <p>If the parser finds the syntax correct, it adds it to the URL Parameters list. For example, if you enter tweet={mytweet} in the Operation URL field and click Parse, mytweet appears under Param Name in the URL Parameters section.</p>
-------	---

For Request/Response Style, select one of the following:

Radio Button	Description
Bare	To support one parameter.
Wrapped	<p>To support multiple parameters within a named wrapper.</p> <p>For Wrapped only, enter:</p> <ul style="list-style-type: none"> • Request Wrapper QName—Specify a unique name for the parent element that is to contain the input parameters that you are wrapping. • Response Wrapper QName—Specify a unique name for the parent element that is to contain the output parameters that you are wrapping.
Multi-Part/Form	To support the transfer of multiple binary types of data for POST operations. For example, you could use this to transfer multiple GIF files.

For URL parameters, you can edit parameters and their data types for those parameters that you are passing through the operation URL.

For Header/Body Parameters, define the header and body parameters and their information. If you want to use the design by example feature, you must save your data source definition to activate the Design By Example button. For more information on how to use this feature, see [Using Design By Example to Infer Parameter Data Types](#).

- Click Add Operation to add a parameter. A new row appears in the Header/Body Parameters section with no name but with a default location (Body), data type (VARCHAR) and in/out direction (IN).
- Double-click under Param Name and type the name of the parameter.
- Click under Location, and from the drop-down list choose one of these options:
 HTTP Header—Parameter is located in the HTTP header.
 Body—Parameter is located in the body of the XML file.
- Click under Data Type, and from the drop-down list choose one of these options:

Option	Description
Decimal	DECIMAL, DOUBLE, FLOAT, or NUMERIC.
Integer	BIGINT, BIT, INTEGER, SMALLINT, or TINYINT.
String	CHAR, CLOB, LONGVARCHAR, or VARCHAR.
Time	DATE, TIME, or TIMESTAMP.
Complex	< >XML Complex XML is any XML defined outside of the W3C XML standard. If your XML has customized elements or requires and XSD file to interpret the meaning of the XML elements within it, the XML is considered complex.
Browse	Choose this to browse for an XSD file that defines the data type being used.

- Click under In/Out, and from the drop-down list choose IN for input parameters or OUT for output parameters.
5. If the data source requires client authentication, click the Advanced tab. See [Client Authentication for Web Data Sources](#) for how to configure client authentication.

6. If the data source requires OAuth, select the OAuth 2.0 tab. Specify the values appropriate to the OAuth flow you want to use.

The following table describes the values the user is to provide on the OAuth 2.0 tab:

Field	Description
OAuth Flow	<p>These OAuth flows:</p> <ul style="list-style-type: none"> • AUTHORIZATION_CODE • IMPLICIT—Client Secret and Access Token URI are disabled. • CLIENT_CREDENTIALS—Resource Owner Authentication fields are disabled. • RESOURCE_OWNER_PASSWORD_CREDENTIALS—Client Authentication fields are disabled. • CUSTOMIZED—User-specified flow.
Client Identification	Used in the request-body of token requests. A unique string representing the identifier issued to the client during registration. It is exposed to the resource owner. Format: string of printable characters.
Client Secret	Used in the request-body of token requests. Enabled only for AUTHORIZATION_CODE, and OAuth flow. Format: string of printable characters.
Authorization URI	URI to use for establishing trust and obtaining the required client properties.
Access Token URI	URI to use for communicating access and refresh tokens to the client or resource server. Disabled only for IMPLICIT OAuth flow.
Redirect URI	Client's redirection end point, established during client registration or when making an authorization request. Must be an absolute URI, and must not contain a fragment component. The authorization server redirects the resource owner to this end point.
Scope	Authorization scope, which is typically limited to the protected resources under the control of the client or as arranged with the authorization server. Limited scope is necessary for an authorization

Field	Description
	grant made using client credentials or other forms of client authentication. Format: one or more strings, separated by spaces.
State	A request parameter used in lieu of a complete redirection URI (if that is not available), or to achieve per-request customization
Expiration Time (Sec)	The lifetime of the access token.
Use Refresh Token To Get Access Token	Checking this box enables the use of refresh tokens to obtain access tokens, rather than obtaining them manually.
Login and Password	User credentials with which the client or resource owner registers with the authentication server before initiating the OAuth protocol.
Pass-through Login	Enabled or Disabled
Authentication	BASIC, DIGEST, NTLM, or NEGOTIATE—The usual collection of authentication methods, used for registration with the authentication server.
Domain	Domain to which the client or resource owner belongs; for example, composite.
Service Principal Name	SPN of the client or resource owner.
Access Token Type	<p>Bearer—User of a bearer token does not need to prove possession of cryptographic key material. Simple inclusion of the access token in the request is sufficient.</p> <p>Query—The query string “?access_token=<token>” is appended to the URL. Not available for SOAP data sources.</p>
Access Token	Credentials used to access protected resources, with a specific scope

Field	Description
	and duration. Usually opaque to the client.
Get Token button	Initiates acquisition of an access token. Proper information must be configured for this request to succeed.
Refresh Token	Credentials used to obtain access tokens when they become invalid or have expired. Intended for use with authorization servers, not resource servers.
Custom Flow	The name of the custom flow for a data source with an OAuth flow of CUSTOMIZED.
Using Processors check box and editable text field	Check this box to use processors. The editable text field allows you to enter JavaScript and XML. You can use this field to add JavaScripts that log in automatically or use XML to customize any part of the authorization or access token that does not conform to specifications.

Editable text field

The editable text field underneath the Using Processors check box can be used to type the XML elements necessary to establish authorization and access tokens. For example:

```
<Authorization>  <AuthorizationProcessors>
```

```
<AuthorizationProcessor>    document.getElementById
('email').value='queenbeeza@gmail.com';
document.getElementById('pass').value='jellypassword';
document.getElementById('loginbutton').click();
</AuthorizationProcessor>  </AuthorizationProcessors>
</Authorization>
```

```
<AccessToken>  <RequestMsgStyle>QUERY</RequestMsgStyle>
<ResponseMsgStyle>FORM</ResponseMsgStyle>
```

Field	Description
	<pre><ExpireTime>1000</ExpireTime> </AccessToken></pre>
Sensitive Keyword in JavaScript	<p>Click the plus-sign icon one or more times to add tag-keyword pairs to substitute into the JavaScript specified for a custom authorization flow. The values sent to the server are encrypted, and then replaced with their decrypted values where the tags are found in the JavaScript.</p> <p>These pairs are used for the user name, email ID, password, and other sensitive information.</p> <p>Tag—The name of the tag to find in the JavaScript.</p> <p>Keyword—The encrypted value to decrypt and substitute for the tag in the JavaScript.</p>

7. Click Create & Close to save the REST data source.

Studio adds the REST data source to the resource tree and opens the Data Source Introspection window for you to introspect the new REST resource.

Passing JSON data to POST REST data source

The POST REST data source is most often configured to accept a set of the scalar parameter as inputs. Those inputs are then used to populate a POST request document. Using the REST data source, typically with a POST verb, parameters can be specified that encodes into the body of the request. Typically, the parameter name is used to delimit the parameter.

To pass a JSON data to POST REST data source:
Define your REST data source.

8. Select the checkbox next to the JSON Format label.
9. In the Basics tab, scroll further down to the Operations section.
10. Define the details for your operation, including the HTTP Verb, Operation Name, and Operation URL.

11. In the Header/Body Parameters table, add a Param Name named [rawdata] with a Data Type of varchar. This is a Body parameter of direction IN. [rawdata] is used to hold the JSON input payload.
12. In the Header/Body Parameters table, add a Param Name named Content-Type with a Data Type of varchar. This is a Body parameter of direction IN. When you run operation, set "Content-Type" to "application/json" which is the MIME type for json
13. In the Header/Body Parameters table, add a Param Name named response with a Data Type of the desired type. This is a Body parameter of direction OUT.

Examples

1. To create a datasource for a specific database adapter, path and properties:

JSON parameters:

```
data source bean list = [{"parentPath":"/shared/examples",
"name":"ds1", "adapterName":"PostgreSQL 9.1", "annotation":"This is a
postgres datasource created using REST api", "nativeProperties":
{"urlIP":"localhost", "urlPort":5432, "urlDatabaseName":"orders",
"login":"tutorial", "password":"A49A5A0FAFF13F4A"} }]
```

Curl Invocation:

```
curl -X POST -u admin:admin "http://localhost:9400/rest/datasource/v1" -
H "Content-Type:application/json" -d "
[{"parentPath":"/shared/examples", "name":"ds1",
"adapterName":"PostgreSQL 9.1", "annotation":"This is a postgres
datasource created using REST api", "nativeProperties":
{"urlIP":"localhost", "urlPort":5432,
"urlDatabaseName":"orders", "login":"tutorial",
"password":"A49A5A0FAFF13F4A"} }]"
```

2. To create a sql script procedure with specified definition script:

JSON Parameters:

```
script beans = [{"parentPath":"/shared/examples", "name":"script1",
"script":"PROCEDURE script1() BEGIN CREATE TABLE /shared/examples/ds_
inventory/tutorial/sampleTable as select OrderId, ProductID, Discount,
OrderDate, CompanyName, CustomerContactFirstName,
CustomerContactLastName, CustomerContactPhone FROM
/shared/examples/ViewOrder; END", "annotation":"This script is created
using REST api"}]
```

Curl Invocation:

```
curl -X POST -u admin:admin "http://localhost:9400/rest/script/v1" -H
"Content-Type:application/json" -d "[{"parentPath\":\"/shared/examples\", \"name\":\"script1\",
\"script\":\"PROCEDURE script1() BEGIN CREATE TABLE /shared/examples/ds_
inventory/tutorial/sampleTable as select OrderId, ProductID, Discount,
OrderDate, CompanyName, CustomerContactFirstName,
CustomerContactLastName, CustomerContactPhone FROM
/shared/examples/ViewOrder; END\", \"annotation\":\"This script is
created using REST api\" }]"
```

Setting a Value for NULL JSON Values in REST Responses

There is a configuration parameter that you can use to set the value of a JSON null the occurs in a REST response. The default value for the parameter is "json_key": NULL. If that value is acceptable, there is nothing to do. If you would like to customize that value, you can use the following steps.

To set a value for NULL JSON values in REST response

Within Studio, select Administration > Configuration.

1. In the Configuration window, navigate to JSON NULL Value in REST Response.
2. Accept the value of null or type a value that you want to assign to all NULL JSON values in a REST response.
3. Click OK.

Passing a Full XML Document in a POST Request to a REST Service

If your REST data source is configured to accept a set of scalar parameters and inputs, being able to pass the whole XML document to the service might be required. When passing an XML document in a POST request to a REST service, if the parameter name is "[rawdata]" then the submitted data is not wrapped but submitted as-is (and not wrapped as a value for an XML element).

To interface CDVP with a REST Service

Define your REST data source.

1. On the Basic tab, scroll down to the Operations section.
2. Define the details for your operation, including the HTTP Verb, Operation Name, and Operation URL.
3. In the Header/Body Parameters table, add an OUT Param Name named response with a Data Type of XML. This is a Body parameter with the direction of OUT.
4. In the Header/Body Parameters table, add an IN Param Name named [rawdata] with a Data Type of XML. This is a Body parameter with the direction of IN.
5. Make sure that the XML <-> JSON check boxes are clear for all the parameters.
6. Save your data source.
7. Invoke the service and provide a full request document as the input parameter.

Using an HTTP Header Parameter to Specify the Content Type

It can be a useful to use a Content-Type HTTP header parameter especially where the REST service requires an XML request but returns a JSON response. When the JSON Format is selected, the REST data source adapter assumes that the format of the request and the format of the response will match. The REST data source automatically generates a header "Content-Type: application/json" in the request. This must be overridden with the value of "application/xml" so that the service can correctly interpret the request.

To use an HTTP header parameter to submit an XML request and get a JSON response

Define your REST data source.

1. On the Basic tab, select the check box next to the JSON Format label.
2. On the Basics tab, scroll down to the Operations section.
3. Define the details for your operation, including the HTTP Verb, Operation Name, and Operation URL.
4. In the Header/Body Parameters table, add an OUT Param Name named response with a Data Type of XML. This is a Body parameter with the direction of OUT.

5. In the Header/Body Parameters table, add an IN Param Name named [rawdata] with a Data Type of XML. This is a Body parameter with the direction of IN.
6. In the Header/Body Parameters table, add an IN Param Name named Content-Type with a Data Type of string. This is a Header parameter with the direction of IN.
7. Clear all the XML <-> JSON check boxes for the [rawdata] parameter.
8. Select the XML <-> JSON check boxes for the response parameter.
9. Save your data source.
10. Invoke the service and provide a full request document for the [rawdata] input parameter, and providing the string application or XML for the Content-Type parameter.

Using Design By Example to Infer Parameter Data Types

Design by example is a feature that provides automated inference of the definition set (XML schema) based on the response from the REST data sources. After you create your REST data source, you can open the data source editor and infer the data types of the parameters by using the Design By Example button, and then you can edit the type if necessary. Typically, design by example is used to infer the data type of body parameters.

There are several ways to access the design by example functionality, including:

- [Creating a Definition Set by Example Using the Data Source Editor](#)
- [Creating a Definition Set by Example Using a Web Service Operation Editor](#)

Creating a Definition Set by Example Using the Data Source Editor

Whether you are creating a new REST data source or editing an existing one, the way you interact with the editor to gain access to the design by example feature is the same.

To create a definition set by example from the Data Source editor

Make sure that all the input parameters defined for your REST data source are defined according to REST operation requirements.

Because the definition set is created from the REST service response, correct input is important, so that the response is correct.

1. Create your REST data source following the instructions in [Adding a REST Data Source](#).
2. Make sure that the following is true for the data source:
 - Operations are listed.
 - Operations have been assigned HTTP verbs.
 - If the Operation URL includes a variable, it is in curly brackets ({}). The URL parameters must be parsed and listed in the URL Parameters portion of the screen.
 - It has been saved and introspected.
3. Open the REST data source editor if it is not already open.
4. Select the Configuration tab.
5. Scroll down to the Connection Information portion of the tab and select the Basic tab.
6. Scroll down to the Operations portion of the screen.
7. Under the Header/Body Parameters portion of the screen, click Design By Example.
If an XML schema file was previously created or specified for this data source, a message pops up. Click OK to continue and overwrite that file with the new information.
8. If your operation has input parameters, type values for the URL Parameters in the Input Values for <op_variable> window. The value that you type must be consistent with the data type of the URL parameter.
The Add Definition Type dialog appears.
9. In the tree pane of the screen, navigate to a sample definition set. TDV uses the sample definition set to automatically create a new definition set from the JSON or XML responses for your REST operation. After the definition set has been created, you can edit it.
Typically, design by example logic is used to suggest metadata for body parameter types, which are usually OUT parameters.
10. In the right-side pane, select an Element or Type from the definition set for the Show field, and then select one of the values in the list in the other portion of the screen.
After you make a selection, the full Studio tree path to that sample definition appears in the Type field.

11. Click OK.

A new row appears in the Header/Body Parameters section with no name, but with default location (Body), data type, and in/out direction (IN). A new definition set resource is created in Studio under the REST data source with the name of the operation TDV used to suggest the schema.

12. Double-click under Param Name and type the name of the parameter.

13. Validate that the values listed for the other columns are consistent with your design.

14. For JSON formatted data, make your XML <-> JSON choice. Select the check box to convert the result set to XML. Clear the check box to retain the JSON formatting.

15. Save your edited data source.

Creating a Definition Set by Example Using a Web Service Operation Editor

To create a definition set by example from a Web service operation editor

Make sure that all the input parameters defined for your REST data source are defined according to REST operation requirements.

Because the definition set will be created from the REST service response, only correct input of the request could get correct response back.

16. Create your REST data source following the instructions in [Adding a REST Data Source](#).

17. Make sure that the following is true for the data source:

- Operations are listed.
- Operations have been assigned HTTP verbs.
- If the Operation URL includes a variable, it is in curly brackets ({}), and the URL parameters are parsed and listed in the URL Parameters portion of the screen.
- It has been saved and introspected.

18. From the Studio resource tree, expand the REST data source node.

19. Select an operation and open its editor.

20. Click Design By Example.

If an XML schema file was previously created or specified for this data source, a message pops up. Click OK to continue and overwrite that file with the new information.

21. If your operation has input parameters, type values for the URL Parameters in the Input Values for <op_variable> window. The value that you type must be consistent with the data type of the URL parameter.

The Add Definition Type dialog appears.

22. In the tree pane of the screen, navigate to a sample definition set. TDV uses the sample definition set to automatically create a new definition set from the JSON or XML responses to the REST operation. After the definition set has been created, you can edit it.

Typically, design by example logic is used to suggest metadata for Body parameter types, which are usually OUT parameters.

23. In the right-side pane, select an Element or Type from the definition set for the Show field, and then select one of the values in the list in the other portion of the screen.

After you make a selection, the full Studio tree path to that sample definition appears in the Type field.

24. Click OK.

A new row appears in the Header/Body Parameters section with no name, but with default location (Body), data type, and in/out direction (IN). A new definition set resource is created in Studio under the REST data source with the name of the operation TDV used to suggest the schema.

25. Double-click under Param Name and type the name of the parameter.

26. Validate that the values listed for the other columns are consistent with your design.

27. Save your changes.

Cross-Origin Resource Sharing (CORS)

For security reasons, browsers restrict cross-origin HTTP requests initiated within scripts. A web page can typically embed many kinds of content from other domains. However, W3C recommends that cross-origin requests for certain items (“restricted resources”) be made only using secure means. Such resources include embedded web fonts and AJAX (XMLHttpRequest) APIs.

Cross-origin resource sharing (CORS) is a mechanism that allows a browser from one domain to request restricted resources from another domain, within well-defined and secure boundaries. TDV supports CORS under Chrome and Firefox browsers, and follows the recommendation for CORS on the Web at <http://www.w3.org/TR/cors/>.

The administrator of a public REST service in TDV configures the CORS environment as described in this section:

- [CORS Configuration](#)

Typically the client makes two kinds of requests, preflight requests and actual requests:

- [Preflight Requests](#)
- [Testing Preflight Requests](#)
- [Actual Requests](#)
- [Testing an Actual Request](#)

CORS Configuration

The TDV configures CORS using configuration parameters. The parameters are in the Administration > Configuration window under Server > Web Services Interface > CORS. These parameters are described in the following table.

Configuration Parameter	Comments
Allow Credentials	A boolean indicating whether the resource allows requests with credentials. Default value is true, because TDV always allows credentials. When this is false, CORS is disabled.
Allowed Headers	A comma-separated list of HTTP headers that may be specified when accessing the resources. Default value is X-Requested-With, Content-Type, Accept, Origin.
Allowed Methods	A comma-separated list of HTTP methods that can be used when accessing the resources. Default list is GET, POST, HEAD.

Configuration Parameter	Comments
Allowed Origins	A comma-separated list of the origins that may access the resources. Default value is * (all origins).
Chain Preflight	If true (default), preflight requests are chained to their target resources for normal handling (that is, handling as OPTION requests). Otherwise, the filter responds to the preflight.
Exposed Headers	A comma-separated list of HTTP headers that may be exposed on the client. Default value is an empty list.
Preflight Max Age	The number of seconds that the client is allowed to cache preflight requests. Default value is 1800 seconds (30 minutes).

Preflight Requests

A preflight request asks the other domain's server which HTTP request methods it supports. This request also asks for an indication of whether cookies or other authentication data should be sent with the actual request.

The preflight request sends its HTTP request using the OPTIONS method, to determine whether the actual request is safe to send. A preflight request is required if the actual request wants to:

- Use a PUT or DELETE method
- Set custom headers in the request—for example, a header such as bb

Testing Preflight Requests

You can debug CORS preflight requests using cURL.

To test a CORS preflight request

Send a preflight request using cUrl:

```
curl -H "Origin: http://example.com" \
-H "Access-Control-Request-Method: POST" \
-H "Access-Control-Request-Headers: X-Requested-With" \
-X OPTIONS --verbose \
https://www.googleapis.com/discovery/v1/apis?fields=
```

This is similar to a regular CORS request, with a few additions:

- The -H flags send additional preflight request headers to the server.
- The -X OPTIONS flag indicates that this is an HTTP OPTIONS request.

1. Optionally, use the -H flag to specify additional headers, such as User-Agent.

If the preflight request is successful, the response should include these response headers:

- Access-Control-Allow-Origin
- Access-Control-Allow-Methods
- Access-Control-Allow-Headers

If the preflight request is not successful, one of the following occurs:

- The three response headers listed above do not appear.
- The HTTP response is not 200.

Actual Requests

An actual request includes an HTTP request method and any cookies or other authentication the other domain's server requires. If the actual request qualifies (see list below) as a simple cross-site request, it does not need to be preceded by a preflight request.

A simple cross-site request has the following characteristics:

- It uses only GET, HEAD or POST to send data to the server.
- It does not set custom headers with the HTTP request.

- The data sent to the other domain's server with the HTTP request is of one of these content types:
 - application/x-www-form-urlencoded
 - multipart/form-data
 - text/plain

If an actual request does not qualify as a simple request, it must be preceded by a preflight request. (See [Preflight Requests](#).)

Testing an Actual Request

You can debug actual CORS requests using cURL, using the URL you are testing in place of the sample Google API, which supports CORS.

To test an actual CORS request

Send a regular CORS request using cURL:

```
curl -H "Origin: http://example.com" \  
  
https://www.googleapis.com/discovery/v1/apis?fields=
```

The -H "Origin: http://example.com" flag is the name of the third-party domain making the request. Substitute the name of your domain.

2. Optionally, use the --verbose flag to print the entire response so you can see the request and response headers.

The response should include the Access-Control-Allow-Origin response header.

Configuring XML/HTTP Data Sources

An XML/HTTP data source collects data from raw XML over HTTP. It collects the information for a single XML/HTTP operation, writes the WSDL for you, and establishes the WSDL data source instance.

The following steps are involved in configuring an XML/HTTP data source for use:

- [Creating an XML Definition Set](#)

- [Adding an XML/HTTP Data Source](#)
- [Retrieving XML Data Over HTTP](#)

Creating an XML Definition Set

An XML definition set provides the output document definition for retrieving XML data over HTTP.

This section gives quick steps for creating an XML-type definition set. Quick steps for creating an XML definition set are given in this section. For more details, see refer to the *User Guide*, Chapter *Definition Sets*.

To create an XML-type definition set

Right-click at an appropriate location in the resource tree, and select New Definition Set.

1. Type a name for the definition set.
2. Select XML in the Type drop-down list.
3. Click OK.

The definition set is added to the resource tree, and the editor for the definition set opens in the right pane.

4. In the editor, select the XML Schema tab to define your XML schema. You can do this in one of three ways.
 - You can type your XML schema in this panel.
 - You can upload an existing XML schema by clicking the Import XML Schema Definitions From File(s) toolbar button, navigating to the file location, and selecting the XML schema file to import. To show or hide the connection properties and alternate location mappings, check or uncheck the Show Advanced Options box.
 - You can base your schema on an existing XML instance. Click the Create XML Schema Definitions from XML Instance toolbar button and navigate to the XML file. When you click Open, the XML file appears on the XML Schema panel of the editor.

Note: The XML instance should be an actual XML file, not a schema.

5. To format the definition set with hierarchical indentation, and with keywords in orange type and literal definitions in blue type, click the Format XML Schema Definitions toolbar button.
6. Click the Validate XML Schema Definitions toolbar button.

If the schema is valid, an information box appears with the message, The definition set is valid. If not, an error dialog box opens with a message describing the line and column where a problem was encountered.

Note: A field in the lower right corner of the XML Schema panel indicates the current line and column position of the cursor, separated by a colon; for example, 20:34 for line 20, column 34.

7. Save the new XML schema using Studio's File > Save command.

Adding an XML/HTTP Data Source

Use the steps in this section to add an XML/HTTP data source. After adding the data source, see [Retrieving XML Data Over HTTP](#), for details on how to access the data.

To create an XML/HTTP data source

Right-click at a location in the Studio resource tree where you want this data source to reside, and select New Data Source.

1. In the New Physical Data Source dialog, select XML/HTTP and click Next.
2. Supply this connection information for an XML/HTTP data source:

- Name—Name for the data source.

On the Basic tab, provide this information for an XML/HTTP data source:

- URL—URL to the HTTP server hosting the XML source.
- Method—GET or POST

Use GET to request data by sending the request information to the HTTP server. Typically, GET is used to retrieve a static resource. However, a query string or extra path information can be appended as a text string after a question mark (?) in the URL of a GET request to trigger server-side processing.

Use POST to send data to the HTTP server to be processed, making sure that the data format of the sending and the receiving programs are compatible.

- Login—Valid username and password to access the HTTP server.

- Password—Password for the sign-in username to access the HTTP server.
- Save Password—Check box that is enabled only if Pass-through Login is enabled.
- Pass-through Login— Refer the *User Guide, About Pass-Through Login*, for more information.
- Authentication—Choose the method of authentication for this data source.
- Domain—For NTLM authentication only, enter the domain.
- Service Principal Name—For NEGOTIATE authentication only, enter the service principal name.
- Access Token—For OAUTH, type the value of the token. For example:

```
e72e16c7e42f292c6912e7710c838347ae178b4a&scope=user%2Cgist&token_
type=bearer
```

- Automatically Retrieve Access Token—For OAUTH.
- No Input—Select if no input is required.
- Input in URL—Select if the input is in the URL.
- Input Document Definition—To specify the input document definition, use the Browse button to locate the definition set you created as described in [Creating an XML Definition Set](#).
- Output Document Definition—To specify the output document definition, browse to the path of the output document definition that supplies the schema for retrieving XML data, as in this example:

```
/<path to definition set>/xmlhttp_ds_schema/GetVotersResponse
```

When you click the Browse button for the Input Document Definition or the Output Document Definition, the Choose Schema window opens.

In the left pane, select the XML definition set in the resource tree.

In the right pane, make sure that Element is selected in the Show drop-down list, and select the definition.

When Element is selected, a specific schema response is expected by the XML/HTTP service. The fully qualified name of the document element must be the same as the element chosen from the schema.

When a Type is selected, the document element must be of the same type.

When you make your selection in the right pane, the Schema field is automatically filled with the complete path to the input or output document definition.

Click OK.

You return to the New Physical Data Source window.

3. Click the Advanced tab.
4. On the Advanced tab, enter this information:
 - Certificates—Enter information about the security certificates used for the XML/HTTP data source. You can click the buttons above the table to:
 - Import Certificate Key Store From File—Type or browse to the path, and select the type and password for the key store.
 - Export Certificate Key Store To File—Export a key store to a file.
 - Clear Certificate Key Store—Clear the key store.
 - Channel Pass-through—Identifies the name of the HTTP request header property that is to be passed through to the XML or HTTP data source. Multiple values can be passed through to the data source by specifying the names as a comma-separated list.
 - Environment Pass-through—Enter one or many environment variables to pass to the XML or HTTP data source for login authentication or other purposes.
5. Click one of these buttons:
 - Create & Introspect—To proceed immediately with introspection.
 - Create & Close—To create the data source; you can introspect at a later time.
6. Refer the *User Guide*, Chapter *Retrieving Data Source Metadata*, for how to introspect now or later.

Retrieving XML Data Over HTTP

After adding the XML/HTTP data source, if you want to have the XML data in tabular form to use it in SQL queries, you need to transform the XML data.

After creating an XML/HTTP data source, you can use it as follows to get XML data:

- Execute the XML source within the data source, and view the data in XML format.

- Create a transformation of the data source, and execute the transformation to view the XML data in tabular form.

To execute the XML source within the data source

Double-click the XML source (also called Web service operation) within the XML/HTTP data source.

Or, you can first select the XML/HTTP data source, and then select the File > Open <data source> menu option.

1. In the editor that opens on the right, click the Execute toolbar button.
The result is displayed in the Result panel.
2. Click the Details button to view the XML data.

To create a transformation of an XML/HTTP data source

Right-click at an appropriate location in the resource tree, and select New Transformation.

3. In the Create Transformation window, select a non-XQuery transformation type. For example, select Basic XML to Tabular Mapping.
4. Click Next.
5. In the tree display, select the XML source in the XML/HTTP data source for the transformation.
6. Type a name for the transformation in the Transformation Name field.
7. Click Finish.
8. For further details on transformations, Refer the *User Guide*, Chapter *Transformations*.

Proxy Configuration Limitations

Following are the limitations for Proxy configuration for web based datasources like SOAP, WSDL, REST, XML/HTTP:

1. TDV only supports http/https basic authentication for proxy setting and not for Kerberos.
2. Passthrough with proxy server is not supported.

3. Proxy authentication is not supported as a per-datasource basis. Username or Password need to be set in /Server/Configuration/Network/Proxy Settings.

Client Authentication for Web Data Sources

When Web data sources require client authentication, a keystore must be specified to identify the TDV Server to the provider. The TDV Server configuration keystore key alias has a default value that names a sample keystore, so that you can use client authentication immediately upon installation.

If the TDV configuration settings for keystore alias are set to null, the method described below to comply with client authentication requirements is used for Web data sources. The TDV configuration to use a specific keystore key alias overrides keystore specification defined on individual data sources.

To specify a keystore to comply with client authentication requirements

Open the Web data source in Studio.

1. Click the Advanced tab in the New Physical Data Source dialog to enable import of a certificate key.
2. Click Import Certificate Key Store from File to import the certificate.

Studio displays a dialog to specify the certificate.

You can choose a jks or pkcs12 certificate key store for authentication between TDV and any Web data source that requires a trusted certificate.

3. Optionally, select a keystore from the list and click Clear Certificate Key Store to remove it.
4. Optionally, click Export Certificate Key Store to File to export the current certificate keystore to a jks or pkcs12 file.
5. Optionally, on the Advanced tab, set the Channel Pass-through field to a name or names that correspond to values passed in the HTTP request header for login authentication or for other purposes.

The Channel Pass-through identifies the name of the HTTP request header property that is to be passed through to the WSDL, XML, or HTTP data source. You can specify multiple values, separated by commas.

If the data source expects a property with a name different from what was originally sent in the HTTP request header, you can change the property name if you want. The name expected by the data source is put on the left side of the “=” operand, and the original property name is put on the right.

6. Optionally, on the Advanced tab, set the Environment Pass-through field to one or more environment variables to pass through to the WSDL, XML, or HTTP data source for login authentication or other purposes.

Procedures can set an environment variable name and value by calling the SetEnvironment procedure. See the Info tab for /lib/util/SetEnvironment in the Studio resource tree for more information.

Property names in the Environment Pass-through field can be renamed before they are passed to the data source, just as they can with channel pass-through.

7. Optionally, for REST data sources, specify the Execution Timeout (msec) period.

Execution Timeout is the number of milliseconds an execution query on the data source is allowed to run before it is canceled. A value of zero milliseconds (default) disables the execution timeout. This can be used, for example, for resource-intensive cache updates set to run at non-peak processing hours.

TDV SOAP and REST OAUTH Examples

The use cases focus on how you would use TDV to customize sign-in automation, and configuration of the parts that do not conform to RFC 6749. These are examples—not exact, and not likely to run outside of one or two specialized environments. OAuth service providers occasionally change their sign-in process, which would require that you analyze the new sign-in process and design accordingly.

- [Google OAuth Example](#)
- [Facebook OAuth Example](#)
- [Linkedin OAUTH Example](#)
- [Salesforce OAuth Example](#)
- [Github OAuth Example](#)
- [Foursquare OAuth Example](#)

Google OAuth Example

With Google, every request and response follows RFC 6749, so the only thing to add is authorization.

If you do not want to show the password in clear text, create a tag like Testpassword instead of what is shown in the XML code for the editable text field below the Using Processors check box:

```
document.getElementById('Passwd').value=' Testpassword';
document.getElementById('gaia_loginform').submit();
```

The sensitive tag is now Testpassword, and the sensitive keyword is the real password (xxxxxx).

OAUTH Tab Field	Example Value
OAuth Flow	AUTHORIZATION_CODE
Authorization URI	https://accounts.google.com/o/oauth2/auth

OAUTH Tab Field	Example Value
AccessToken URI	https://accounts.google.com/o/oauth2/token
Text field below the Using Processors check box	<Authorization>
	<AuthorizationProcessors>
	<AuthorizationProcessor>
	<pre> <![CDATA[document.getElementById ('Email').value='test@gmail.com'; document.getElementById('gaia_ loginform').submit(); </pre>
	<pre> //document.getElementById('next').click();]]> </pre>
	</AuthorizationProcessor>
	<AuthorizationProcessor>
	<pre> <![CDATA[document.getElementById ('Passwd').value='xxxxx'; document.getElementById('gaia_ loginform').submit(); </pre>
	<pre> //document.getElementById('signIn').click();]]> </pre>
	</AuthorizationProcessor>

OAuth Tab Field	Example Value
	<AuthorizationProcessor>
	<![CDATA[document.getElementById('submit_approve_access').click();]]>
	</AuthorizationProcessor>
	</AuthorizationProcessors>
	</Authorization>

Facebook OAuth Example

Facebook does not conform to RFC 6749. According to RFC 6749, the request for the access token should be in FORM format, and the response should be in JSON format. With Facebook, the request is in QUERY format, and the response is in FORM format, so you need to use processors to configure them correctly.

In this example, Facebook's ExpireTime is named 'expires' instead of 'expires_in' as called for in RFC 6749, so the user should directly specify an expiry time.

Another way to get expire time is to use TokenProcessor, which can handle the input data and return standard JSON data. In this case, MessageValue is the value to retrieve from the response body, because the valid response is in FORM format. By retrieving access token and expire time from MessageValue, the token processor can return standard parameters that conform to RFC 6749 and JSON format.

OAuth Tab Field	Sample Values
Authorization URI	https://graph.facebook.com/oauth/authorize

OAuth Tab Field	Sample Values
AccessToken URI	https://graph.facebook.com/oauth/access_token
Text field below the Using Processors check box	<pre> <Authorization> <AuthorizationProcessors> <AuthorizationProcessor> document.getElementById ('email').value='test@gmail.com'; document.getElementById ('pass').value='xxxxxx'; document.getElementById('loginbutton').click (); </AuthorizationProcessor> </AuthorizationProcessors> </Authorization> <AccessToken> <RequestMsgStyle>QUERY</RequestMsgStyle> <ResponseMsgStyle>FORM</ResponseMsgStyle> <ExpireTime>1000</ExpireTime> </pre>

OAuth Tab Field	Sample Values
	<code></AccessToken></code>
	<code><TokenProcessor></code>
	<code>VAR accesstoken;</code>
	<code>VAR expires;</code>
	<code>...//Get access token and expire-time value from MessageValue</code>
	<code>MessageValue = "{access_token:" + accesstoken + " expires_in:" + expires+ "}";</code>
	<code></TokenProcessor></code>

Linkedin OAUTH Example

OAuth Tab Field	Sample Values
Authorization URI	<code>https://www.linkedin.com/uas/oauth2/authorization</code>
AccessToken URI	<code>https://www.linkedin.com/uas/oauth2/accessToken</code>
Text field below the Using Processors check box	<div><code><Authorization></code></div> <div><code><AuthorizationProcessors></code></div>

OAuth Tab Field	Sample Values
	<code><AuthorizationProcessor></code>
	<pre> document.getElementById('session_key- oauth2SAuthorizeForm').value='test@gmail.com'; document.getElementById('session_password- oauth2SAuthorizeForm').value='xxxxxx'; document.getElementsByTagName('form') [0].submit(); </pre>
	<code></AuthorizationProcessor></code>
	<code></AuthorizationProcessors></code>
	<code></Authorization></code>

Github OAuth Example

Github does not conform to RFC 6749 because the request is in QUERY format and the response is in FORM format.

OAuth Tab Field	Sample Values
Authorization URI	<code>https://github.com/login/oauth/authorize</code>
AccessToken URI	<code>https://github.com/login/oauth/access_token</code>
Text field below the Using Processors	<code><Authorization></code>

OAuth Tab Field	Sample Values
check box	<AuthorizationProcessors>
	<AuthorizationProcessor>
	document.getElementById('login_field').value='test@gmail.com';
	document.getElementById('password').value='xxxxxx'; document.getElementsByTagName('form')[1].submit();
	</AuthorizationProcessor>
	</AuthorizationProcessors>
	</Authorization>
	<AccessToken>
	<RequestMsgStyle>QUERY</RequestMsgStyle>
	<ResponseMsgStyle>FORM</ResponseMsgStyle>
	</AccessToken>

Salesforce OAuth Example

Salesforce has some special requirements:

- When logging for an authorization code, Salesforce always sends email to a registered email box with a verification code, and asks for this code during the process. You need to handle this manually or use CustomFlow.
- Check Use Refresh Token To Get Access Token after obtaining the access_token and refresh_token the first time. If you do this, the token can be retrieved automatically each time, without needing to check the verification code manually in the email box.

The screenshot shows the configuration interface for a Salesforce OAuth2 data source. The 'OAuth Flow' section is configured with 'AUTHORIZATION_CODE' as the flow. The 'Automation Process To Get Access Token' section has 'Using Processors' checked, and the custom flow contains JavaScript code to extract the authorization code from the response. The 'Resource Owner Authentication' and 'Client Authentication' sections are also visible, showing fields for login, password, and other authentication details. The 'Access Token' section shows the 'Bearer' type selected and a placeholder for the access token.

In the example, using the JavaScript regular expression to fetch the authorization code is just for demonstration purposes.

OAuth Tab Field	Example Values
Authorization URI	https://login.salesforce.com/services/oauth2/authorize
AccessToken URI	https://login.salesforce.com/services/oauth2/token
Text field below the Using Processors check box	<div><Authorization></div> <div><ResponseMsgStyle>RAWBODY</ResponseMsgStyle></div>

OAuth Tab Field	Example Values
	<code><AuthorizationProcessors></code>
	<code><AuthorizationProcessor></code>
	<pre>document.getElementById ('username').value='test@gmail.com'; document.getElementById ('password').value='xxxxxx'; document.getElementById ('Login').click();</pre>
	<code></AuthorizationProcessor></code>
	<code></AuthorizationProcessors></code>
	<code><TokenProcessor></code>
	<pre>var pattern=/\?code=.+/i; var ans = pattern.exec(MessageValue); if(ans.index>0) {MessageValue = "{code:" + ans[0].substring (6,ans[0].length-1)+""};}</pre>
	<code></TokenProcessor></code>
	<code></Authorization></code>

Foursquare OAuth Example

Foursquare uses Query as the access token type. However, it uses oauth_token as the access token name instead of using access_token in the URL, as defined in RFC 6749, so you need to use QueryTokenName for the service.

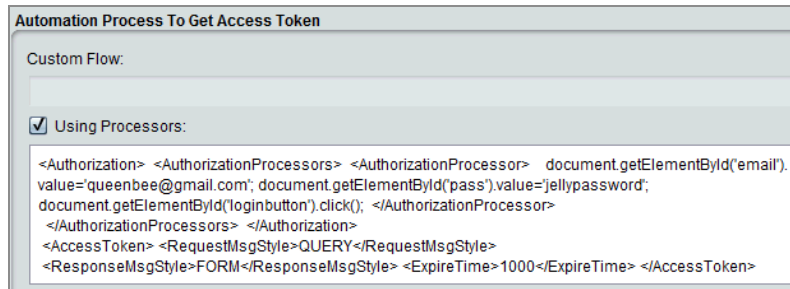
OAuth Tab Field	Sample Values
Authorization URI	https://foursquare.com/oauth2/authenticate
AccessToken URI	https://foursquare.com/oauth2/access_token
Text field below the Using Processors check box	<pre> <Authorization> <AuthorizationProcessors> <AuthorizationProcessor> document.getElementById ('username').value='test@gmail.com'; document.getElementById ('password').value='xxxxxx'; document.getElementById ('loginToFoursquare').submit(); </AuthorizationProcessor> </AuthorizationProcessors> </Authorization> <QueryTokenName>oauth_token</QueryTokenName> </pre>

TDV OAuth Tab XML Processors Field Reference

On the OAuth tab for your WSDL, SOAP, and REST data sources, you can define custom processors. The custom processors use XML to provide authorization and access token

customization. TDV provides a collection of processors and XML elements that you can use to accommodate nonconforming requests and responses. This topic provides examples that use these processors to illustrate how they obtain access tokens from several well-known third parties.

This topic is a reference of the XML element syntax that is valid for entry in this field.



Automation Process To Get Access Token

Custom Flow:

☒ Using Processors:

```
<Authorization> <AuthorizationProcessors> <AuthorizationProcessor> document.getElementById('email').
value='queenbee@gmail.com'; document.getElementById('pass').value='jellypassword';
document.getElementById('loginbutton').click(); </AuthorizationProcessor>
</AuthorizationProcessors> </Authorization>
<AccessToken> <RequestMsgStyle>QUERY</RequestMsgStyle>
<ResponseMsgStyle>FORM</ResponseMsgStyle> <ExpireTime>1000</ExpireTime> </AccessToken>
```

The OAuth authorization framework enables a third-party application to obtain secure, temporary access to an HTTP service. However, because most interactions between client and resource owner do not conform to RFC 6749, it may be necessary for you to modify requests and responses to make them conform to the specification.

For example, you might have code similar to the following in the field:

```
<Authorization>
```

```
<AuthorizationProcessors>
```

```
<AuthorizationProcessor>
```

```
document.getElementById('email').value='test@gmail.com';
document.getElementById('pass').value='xxxxxx';
```

```
document.getElementById('loginbutton').click();
```

```
</AuthorizationProcessor>
```

```
</AuthorizationProcessors>
```

```
</Authorization>
```

```
<AccessToken>
```

```
<RequestMsgStyle>QUERY</RequestMsgStyle>
```

```
<ResponseMsgStyle>FORM</ResponseMsgStyle>
```

```
<ExpireTime>1000</ExpireTime>
```

```
</AccessToken>
```

```
<TokenProcessor>
```

```
VAR accesstoken;
```

```
VAR expires;
```

```
...//Get access token and expire-time value from MessageValue
```

```
MessageValue = "{access_token:" + accesstoken + ", expires_in:" +  
expires+ "}";
```

```
</TokenProcessor>
```

- [Authorization Element Reference](#)
- [AccessToken XML Element Reference](#)
- [RefreshToken Element](#)
- [QueryTokenName Element](#)

Authorization Element Reference

The authorization element lets you customize the authorization segment of the OAuth flow.

XML Schema of the Element

```
<xs:element name="Authorization" minOccurs="0">
```

```
<xs:complexType>
```

```
<xs:sequence>
```

```
<xs:element ref="RequestMsgStyle" minOccurs="0"/>
```

```
<xs:element ref="ResponseMsgStyle" minOccurs="0"/>
```

```
<xs:element ref="AuthorizationProcessors" minOccurs="0"/>
```

```
<xs:element ref="TokenProcessor" minOccurs="0"/>
```

```
</xs:sequence>
```

```
</xs:complexType>
```

```
</xs:element>
```

Sequence Element	Description of value
RequestMsgStyle	According to RFC 6749, the default value is GET for Authorization Code Grant, Implicit Grant, and Customized Flow.

Sequence Element	Description of value
	<pre><xs:element name="RequestMsgStyle" minOccurs="0"></pre>
	<pre><xs:simpleType></pre>
	<pre><xs:restriction base="xs:string"></pre>
	<pre><xs:enumeration value="FORM"/></pre>
	<pre><xs:enumeration value="QUERY"/></pre>
	<pre><xs:enumeration value="QUERYPOST"/></pre>
	<pre></xs:restriction></pre>
	<pre></xs:simpleType></pre>
	<pre></xs:element></pre>
	<ul style="list-style-type: none"> • QUERY—Append all request parameters of OAuth to the URL as a query string, using the HTTP GET method. • QUERYPOST—Add all request parameters of OAuth to URL as query string, using the HTTP POST method. • FORM—Add all request parameters of OAuth to the request body, using a Content-Type of application/x-www-form-urlencoded.
ResponseMsgStyle	According to RFC 6749, the default value is GET for Authorization Code Grant, Implicit Grant, and Customized

Sequence Element	Description of value
	<p>Flow.</p> <pre><xs:element name="ResponseMsgStyle" minOccurs="0"></pre> <pre><xs:simpleType></pre> <pre><xs:restriction base="xs:string"></pre> <pre><xs:enumeration value="FORM"/></pre> <pre><xs:enumeration value="QUERY"/></pre> <pre><xs:enumeration value="JSON"/></pre> <pre><xs:enumeration value="RAWBODY"/></pre> <pre></xs:restriction></pre> <pre></xs:simpleType></pre> <pre></xs:element></pre> <ul style="list-style-type: none"> • QUERY—All response parameters of OAuth are returned as a query string appended to a redirect URL. • FORM—All response parameters of OAuth are returned with the entity, and Content-Type is application/x-www-form-urlencoded. • JSON—All response parameters of OAuth are returned

Sequence Element	Description of value
	<p>with the entity, and Content-Type is application/json.</p> <ul style="list-style-type: none">• RAWBODY—All response parameters of OAuth are returned with the entity, but the format is not clearly defined. In this case, use tokenProcessor(Javascript) to retrieve all parameters.
AuthorizationProcessors	<p>Simulates a browser (user agent), automatically performing whatever authorization process is required to log in. Each AuthorizationProcessor within AuthorizationProcessors should be mapped to one pop-up page of the browser.</p> <div><pre><xs:element name="AuthorizationProcessors" minOccurs="0"></pre></div> <div><pre><xs:complexType></pre></div> <div><pre><xs:sequence></pre></div> <div><pre><xs:element maxOccurs="unbounded" ref="AuthorizationProcessor"/></pre></div> <div><pre></xs:sequence></pre></div> <div><pre></xs:complexType></pre></div> <div><pre></xs:element></pre></div> <div><pre><xs:element name="AuthorizationProcessor"></pre></div> <div><pre><xs:simpleType></pre></div>

Sequence Element	Description of value
	<pre><xs:restriction base="xs:string"/></pre>
	<pre></xs:simpleType></pre>
	<pre></xs:element></pre>
TokenProcessor	<p>Get tokens or other parameters if the response is not in the specified format.</p> <h3>XML Schema of the Element</h3> <pre><xs:element name="TokenProcessor" minOccurs="0"></pre> <pre><xs:simpleType></pre> <pre><xs:restriction base="xs:string"/></pre> <pre></xs:simpleType></pre> <pre></xs:element></pre>

AccessToken XML Element Reference

The AccessToken element lets you customize the acquisition of an access token in the OAuth flow. A way to get expire time is to use TokenProcessor, which can handle the input data and return standard JSON data. A MessageValue can be used to retrieve from the response body, because the valid response is in FORM format. By retrieving access token and expire time from MessageValue, the token processor can return standard parameters that conforms to RFC 6749 and JSON format.

XML Schema of the Element

```
<xs:element name="AccessToken" minOccurs="0">

  <xs:complexType>

    <xs:sequence>

      <xs:element ref="RequestMsgStyle" minOccurs="0"/>

      <xs:element ref="ResponseMsgStyle" minOccurs="0"/>

      <xs:element ref="ExpireTime" minOccurs="0"/>

      <xs:element ref="TokenProcessor" minOccurs="0"/>

    </xs:sequence>

  </xs:complexType>

</xs:element>
```

Sequence Element	Description of Values
RequestMsgStyle	According to RFC 6749, the default value is FORM for Authorization Code Grant, Client Credentials Grant, Resource Owner Password Credentials Grant, and Customized Flow. <div><xs:element name="RequestMsgStyle" minOccurs="0"></div> <div><xs:simpleType></div>

Sequence Element	Description of Values
	<pre><xs:restriction base="xs:string"></pre>
	<pre><xs:enumeration value="FORM"/></pre>
	<pre><xs:enumeration value="QUERY"/></pre>
	<pre><xs:enumeration value="QUERYPOST"/></pre>
	<pre></xs:restriction></pre>
	<pre></xs:simpleType></pre>
	<pre></xs:element></pre> <ul style="list-style-type: none"> • QUERY—Append all request parameters of OAuth to the URL as a query string, using the HTTP GET method. • QUERYPOST—Add all request parameters of OAuth to URL as query string, using the HTTP POST method. • FORM—Add all request parameters of OAuth to the request body, using a Content-Type of application/x-www-form-urlencoded.
ResponseMsgStyle	<p>According to RFC 6749, the default value is JSON for Authorization Code Grant, Client Credentials Grant, Resource Owner Password Credentials Grant, and Customized Flow.</p> <pre><xs:element name="ResponseMsgStyle" minOccurs="0"></pre> <pre><xs:simpleType></pre>

Sequence Element	Description of Values
	<code><xs:restriction base="xs:string"></code>
	<code><xs:enumeration value="FORM"/></code>
	<code><xs:enumeration value="QUERY"/></code>
	<code><xs:enumeration value="JSON"/></code>
	<code><xs:enumeration value="RAWBODY"/></code>
	<code></xs:restriction></code>
	<code></xs:simpleType></code>
	<code></xs:element></code>
	<ul style="list-style-type: none"> • QUERY—All response parameters of OAuth are returned as a query string appended to a redirect URL. • FORM—All response parameters of OAuth are returned with the entity, and Content-Type is application/x-www-form-urlencoded. • JSON—All response parameters of OAuth are returned with the entity, and Content-Type is application/json. • RAWBODY—All response parameters of OAuth are returned with the entity, but the format is not clearly defined. In this case, use tokenProcessor(JavaScript) to retrieve all parameters.
ExpireTime	Sets an expiration time for the access token, overriding the default time of 5 seconds.

Sequence Element	Description of Values
------------------	-----------------------

XML Schema of the Element

```
<xs:element name="ExpireTime" minOccurs="0">
```

```
<xs:simpleType>
```

```
<xs:restriction base="xs:integer"/>
```

```
</xs:simpleType>
```

```
</xs:element>
```

TokenProcessor

Gets tokens or other parameters if the authorization response is not in the specified format.

XML Schema of the Element

```
<xs:element name="TokenProcessor" minOccurs="0">
```

```
<xs:simpleType>
```

```
<xs:restriction base="xs:string"/>
```

```
</xs:simpleType>
```

```
</xs:element>
```

RefreshToken Element

The RefreshToken element lets you customize the way the access token is refreshed in the OAuth flow.

XML Schema of the Element

```
<xs:element name="RefreshToken" minOccurs="0">
```

```
<xs:complexType>
```

```
<xs:sequence>
```

```
<xs:element ref="RequestMsgStyle" minOccurs="0"/>
```

```
<xs:element ref="ResponseMsgStyle" minOccurs="0"/>
```

```
<xs:element ref="ExpireTime" minOccurs="0"/>
```

```
<xs:element ref="TokenProcessor" minOccurs="0"/>
```

```
</xs:sequence>
```

```
</xs:complexType>
```

```
</xs:element>
```

Sequence Element	Description of Values
RequestMsgStyle	According to RFC 6749, the default value is FORM for Authorization Code Grant, Resource Owner Password Credentials Grant, and Customized Flow.

Sequence Element	Description of Values
	<pre><xs:element name="RequestMsgStyle" minOccurs="0"></pre>
	<pre><xs:simpleType></pre>
	<pre><xs:restriction base="xs:string"></pre>
	<pre><xs:enumeration value="FORM"/></pre>
	<pre><xs:enumeration value="QUERY"/></pre>
	<pre><xs:enumeration value="QUERYPOST"/></pre>
	<pre></xs:restriction></pre>
	<pre></xs:simpleType></pre>
	<pre></xs:element></pre>
	<ul style="list-style-type: none"> • QUERY—Append all request parameters of OAuth to the URL as a query string, using the HTTP GET method. • QUERYPOST—Add all request parameters of OAuth to URL as query string, using the HTTP POST method. • FORM—Add all request parameters of OAuth to the request body, using a Content-Type of application/x-www-form-urlencoded.
ResponseMsgStyle	According to RFC 6749, the default value is JSON for Authorization Code Grant, Resource Owner Password Credentials Grant, and Customized Flow.

Sequence Element	Description of Values
	<code><xs:element name="ResponseMsgStyle" minOccurs="0"></code>
	<code><xs:simpleType></code>
	<code><xs:restriction base="xs:string"></code>
	<code><xs:enumeration value="FORM"/></code>
	<code><xs:enumeration value="QUERY"/></code>
	<code><xs:enumeration value="JSON"/></code>
	<code><xs:enumeration value="RAWBODY"/></code>
	<code></xs:restriction></code>
	<code></xs:simpleType></code>
	<code></xs:element></code>
	<ul style="list-style-type: none"> • QUERY—All response parameters of OAuth are returned as a query string appended to a redirect URL. • FORM—All response parameters of OAuth are returned with the entity, and Content-Type is application/x-www-form-urlencoded. • JSON—All response parameters of OAuth are returned with the entity, and Content-Type is application/json. • RAWBODY—All response parameters of OAuth are returned with the entity, but the format is not clearly defined. In this

Sequence Element	Description of Values
	case, use tokenProcessor(JavaScript) to retrieve all parameters.
ExpireTime	Sets an expiration time for the access token, overriding the default time of 5 seconds.

XML Schema of the Element

```
<xs:element name="ExpireTime" minOccurs="0">
```

```
<xs:simpleType>
```

```
<xs:restriction base="xs:integer"/>
```

```
</xs:simpleType>
```

```
</xs:element>
```

TokenProcessor	Gets tokens or other parameters if the authorization response is not in the specified format.
----------------	---

XML Schema of the Element

```
<xs:element name="TokenProcessor" minOccurs="0">
```

```
<xs:simpleType>
```

```
<xs:restriction base="xs:string"/>
```

```
</xs:simpleType>
```

Sequence Element	Description of Values
	<code></xs:element></code>

QueryTokenName Element

If the access token type is Query, this element specifies the name in the query string if the name is different from `access_token`.

XML Schema of the Element

```
<xs:element name="QueryTokenName" minOccurs="0">
```

```
<xs:simpleType>
```

```
<xs:restriction base="xs:string"/>
```

```
</xs:simpleType>
```

```
</xs:element>
```

Partial Example of Using OAuth Customized Flow for WSDL, SOAP, and REST

The CustomFlow interface is made available to support customized options or processes so that they can obtain an access token and other parameters.

In OAuthProfileCallback, all required fields are provided in the UI. In each request:

- Construct the OAuthProfileCallback.Request with method info, request URL, request body and headers.

- Handle OAuthProfileCallback.Response to get results.

You might need to call `handleAuthResponse` several times while interacting with the service side. Because this is a simulation of the browser, it is your responsibility to extract the required information from `OAuthProfileCallback.Response`.

```
package com.compositesw.extension.security.oauth;
```

```
import java.util.Map;
```

```
/**Custom flow used for Extension Grants of OAuth2.0:  
http://tools.ietf.org/html/rfc6749#section-4.5.
```

```
* Any OAuth 2.0 flow with customized request or response that does not  
conform to RFC 6749 can be
```

```
* customized.
```

```
* With CustomFlow, a flow step is ignored if the request is NULL, or if  
no request info is defined
```

```
* in OAuthProfileCallback.
```

```
*/
```

```

public interface CustomFlow {

    /**

        * Build authorization request. If request info is NULL, the
        authorization step is ignored. */

    public void buildAuthRequest(OAuthProfileCallback callback) throws
    Exception;

    /**

        * Handle authorization response. */

    public void handleAuthResponse(OAuthProfileCallback callback) throws
    Exception;

    /**

        * Build access token request. If request info is NULL, the
        authorization step is ignored.

        * The flow fails if both authorization request and access token
        request info are NULL.

        */

    public void buildAccessTokenRequest(OAuthProfileCallback callback)
    throws Exception;

    /**

        * Handle access token response. */

```

```
public void handleAccessTokenResponse(OAuthProfileCallback callback)
throws Exception;
```

```
/**
```

```
    * Build refresh token request. If request info is NULL, the
    authorization step is ignored.
```

```
*/
```

```
public void buildRefreshTokenRequest(OAuthProfileCallback callback)
throws Exception;
```

```
/**
```

```
    * Handle refresh token response. */
```

```
public void handleRefreshTokenResponse(OAuthProfileCallback callback)
throws Exception;
```

```
/**
```

```
    * All OAuth elements (access_token, refresh_token, expires_in,
    token_type, scope, etc.)
```

```
    * extracted from response can be found in the value map returned by
    getOAuthElements(). */
```

```
public Map<String, Object> getOAuthElements();
```

```
/**
```

```
    * Get access token. */
```

```
public String getAccessToken();
```

```
/**
```

```
 * Get refresh token. */
```

```
public String getRefreshToken();
```

```
}
```


TIBCO Product Documentation and Support Services

For information about this product, you can read the documentation, contact TIBCO Support, and join the TIBCO Community.

How to Access TIBCO Documentation

Documentation for TIBCO products is available on the [TIBCO Product Documentation](#) website, mainly in HTML and PDF formats.

The [TIBCO Product Documentation](#) website is updated frequently and is more current than any other documentation included with the product.

Product-Specific Documentation

The following documentation for this product is available on the [TIBCO® Data Virtualization](#) page.

- **Users**
 - TDV Getting Started Guide
 - TDV User Guide
 - TDV Web UI User Guide
 - TDV Client Interfaces Guide
 - TDV Tutorial Guide
 - TDV Northbay Example
- **Administration**
 - TDV Installation and Upgrade Guide
 - TDV Administration Guide
 - TDV Active Cluster Guide
 - TDV Security Features Guide
- **Data Sources**

TDV Adapter Guides

TDV Data Source Toolkit Guide (Formerly Extensibility Guide)

- **References**

TDV Reference Guide

TDV Application Programming Interface Guide

- **Other**

TDV Business Directory Guide

TDV Discovery Guide

- *TIBCO TDV and Business Directory Release Notes* Read the release notes for a list of new and changed features. This document also contains lists of known issues and closed issues for this release.

How to Contact TIBCO Support

Get an overview of [TIBCO Support](#). You can contact TIBCO Support in the following ways:

- For accessing the Support Knowledge Base and getting personalized content about products you are interested in, visit the [TIBCO Support](#) website.
- For creating a Support case, you must have a valid maintenance or support contract with TIBCO. You also need a user name and password to log in to [TIBCO Support](#) website. If you do not have a user name, you can request one by clicking **Register** on the website.

Release Version Support

TDV 8.5 is designated as a Long Term Support (LTS) version. Some release versions of TIBCO® Data Virtualization products are selected to be long-term support (LTS) versions. Defect corrections will typically be delivered in a new release version and as hotfixes or service packs to one or more LTS versions. See also

https://docs.tibco.com/pub/tdv/general/LTS/tdv_LTS_releases.htm.

How to Join TIBCO Community

TIBCO Community is the official channel for TIBCO customers, partners, and employee subject matter experts to share and access their collective experience. TIBCO Community offers access to Q&A forums, product wikis, and best practices. It also offers access to extensions, adapters, solution accelerators, and tools that extend and enable customers to gain full value from TIBCO products. In addition, users can submit and vote on feature requests from within the [TIBCO Ideas Portal](#). For a free registration, visit [TIBCO Community](#).

Legal and Third-Party Notices

SOME TIBCO SOFTWARE EMBEDS OR BUNDLES OTHER TIBCO SOFTWARE. USE OF SUCH EMBEDDED OR BUNDLED TIBCO SOFTWARE IS SOLELY TO ENABLE THE FUNCTIONALITY (OR PROVIDE LIMITED ADD-ON FUNCTIONALITY) OF THE LICENSED TIBCO SOFTWARE. THE EMBEDDED OR BUNDLED SOFTWARE IS NOT LICENSED TO BE USED OR ACCESSED BY ANY OTHER TIBCO SOFTWARE OR FOR ANY OTHER PURPOSE.

USE OF TIBCO SOFTWARE AND THIS DOCUMENT IS SUBJECT TO THE TERMS AND CONDITIONS OF A LICENSE AGREEMENT FOUND IN EITHER A SEPARATELY EXECUTED SOFTWARE LICENSE AGREEMENT, OR, IF THERE IS NO SUCH SEPARATE AGREEMENT, THE CLICKWRAP END USER LICENSE AGREEMENT WHICH IS DISPLAYED DURING DOWNLOAD OR INSTALLATION OF THE SOFTWARE (AND WHICH IS DUPLICATED IN THE LICENSE FILE) OR IF THERE IS NO SUCH SOFTWARE LICENSE AGREEMENT OR CLICKWRAP END USER LICENSE AGREEMENT, THE LICENSE(S) LOCATED IN THE “LICENSE” FILE(S) OF THE SOFTWARE. USE OF THIS DOCUMENT IS SUBJECT TO THOSE TERMS AND CONDITIONS, AND YOUR USE HEREOF SHALL CONSTITUTE ACCEPTANCE OF AND AN AGREEMENT TO BE BOUND BY THE SAME.

This document is subject to U.S. and international copyright laws and treaties. No part of this document may be reproduced in any form without the written authorization of TIBCO Software Inc.

TIBCO, TIBCO logo, TIBCO O logo, ActiveSpaces, Enterprise Messaging Service, Spotfire, TERR, S-PLUS, and S+ are either registered trademarks or trademarks of TIBCO Software Inc. in the United States and/or other countries.

Java and all Java based trademarks and logos are trademarks or registered trademarks of Oracle Corporation and/or its affiliates.

All other product and company names and marks mentioned in this document are the property of their respective owners and are mentioned for identification purposes only.

This software may be available on multiple operating systems. However, not all operating system platforms for a specific software version are released at the same time. See the

readme file for the availability of this software version on a specific operating system platform.

THIS DOCUMENT IS PROVIDED “AS IS” WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, OR NON-INFRINGEMENT.

THIS DOCUMENT COULD INCLUDE TECHNICAL INACCURACIES OR TYPOGRAPHICAL ERRORS. CHANGES ARE PERIODICALLY ADDED TO THE INFORMATION HEREIN; THESE CHANGES WILL BE INCORPORATED IN NEW EDITIONS OF THIS DOCUMENT. TIBCO SOFTWARE INC. MAY MAKE IMPROVEMENTS AND/OR CHANGES IN THE PRODUCT(S) AND/OR THE PROGRAM(S) DESCRIBED IN THIS DOCUMENT AT ANY TIME.

THE CONTENTS OF THIS DOCUMENT MAY BE MODIFIED AND/OR QUALIFIED, DIRECTLY OR INDIRECTLY, BY OTHER DOCUMENTATION WHICH ACCOMPANIES THIS SOFTWARE, INCLUDING BUT NOT LIMITED TO ANY RELEASE NOTES AND "READ ME" FILES.

This and other products of TIBCO Software Inc. may be covered by registered patents. Please refer to TIBCO's Virtual Patent Marking document (<https://www.tibco.com/patents>) for details.

Copyright © 2002-2023 Cloud Software Group, Inc All Rights Reserved.