

Text Protocol Driver



Stefaan Van Cauwenberge

Version 1.0



Table of Contents

1. Introduction.....	4
1.1. About this driver	4
2. Components	4
2.1. Architecture Overview	4
2.1.1. High level.....	4
2.1.2. Shim Subscriber	5
2.1.3. Shim Publisher	6
2.1.3.1. Poller	6
2.1.3.2. Listener.....	7
2.2. General concepts.....	7
2.2.1. Context parameters.....	7
2.2.2. Velocity formatting.....	8
2.2.3. Transaction object	9
2.2.3.1. Common transaction features	9
2.2.3.1.1. Operation data	9
2.2.3.1.2. Tracing sensitive data	9
2.2.3.2. Subscriber channel.....	10
2.2.3.3. Publisher channel.....	10
2.3. Subscriber specific concepts	10
2.3.1. ICommand	10
2.3.2. Chaining.....	11
2.3.3. Initialization	12
2.3.4. Token replacement	12
2.4. Publisher specific Concepts	13
2.4.1. Polling.....	13
2.4.1.1. Introduction	13
2.4.1.2. Operation data.....	13
2.4.1.3. Processing.....	13
2.4.2. Listening	14
3. Strategies (plugins).....	14
3.1. Subscriber.....	14
3.1.1. Command formatting	14
3.1.2. Command Execution	14
3.1.3. Response Parsing	15
3.1.4. Response Formatting	16
3.2. Publisher.....	16
3.2.1. Request Parsing.....	16
3.2.2. Request Formatting	16
3.2.3. Response Formatting	16
4. Shim configuration.....	17
4.1. Subscriber.....	17
4.1.1. Main.....	17
4.1.2. Command formatting	18
4.1.2.1. VelocitySubRequestFormatStrategy	18
4.1.2.2. JsonSubRequestFormatStrategy	18
4.1.3. Command Execute	18
4.1.3.1. Http	18
4.1.3.2. Script	21
4.1.3.3. File	22

4.1.4. Response Parsing	22
4.1.4.1. JSON	22
4.1.4.2. JDOM.....	22
4.1.4.3. W3C DOM	23
4.1.4.4. None	23
4.1.5. Response Formatting	23
4.1.5.1. Velocity.....	23
4.2. Publisher.....	23
4.2.1. Main.....	23
4.2.2. Request parsing	24
4.2.2.1. JSON	24
4.2.2.2. JDOM.....	24
4.2.2.3. W3C DOM	24
4.2.2.4. None	24
4.2.3. Submit Judge.....	25
4.2.3.1. Javascript	25
4.2.4. Request Formatting	25
4.2.4.1. Velocity.....	25
4.2.5. Response Formatting	25
4.2.5.1. Velocity.....	25
4.2.6. Poller.....	25
4.2.7. Listener	25
4.2.7.1. Http	25
5. Installation	26
5.1. Non remote loader	26
5.2. Remote loader.....	26
6. Configuration	27
6.1. Velocity Templating	27
6.2. Context Parameters.....	27
7. Example Configurations.....	28
7.1. SLES user provisioning (scripting)	28
7.2. Redhat IPA (HTTP)	28
7.3. Simple scripts for request URL and headersGoogle Apps (HTTP).....	28
8. Third party libraries and licenses	29

1

2 Introduction

2.1 About this driver

This shims is a **protocol level shim**, focusing on text based protocols. Text base protocols are those protocols that are mainly used as a means to transfer the (text) content. As a result, the actual number of methods in the API's are typically limited.

Examples:

- http: the content is for example the actual JSON, SOAP or XML message. Http is the means of transporting the content, with a limited number of API calls (PUT, GET, POST,...).

- Scripting: the content is the actual on the fly generated script. The protocol is the execution of the script.

This driver provides an extremely flexible but simple method to integrate with these text base protocols.

Flexibility and simplicity are provided via:

- a pluggable architecture in every important step of the process
- javascript based configuration
- velocity templates for formatting both the XDS and application commands
- example configuration for eg Google Apps (bidirectional) and Linux account creation.

3 Components

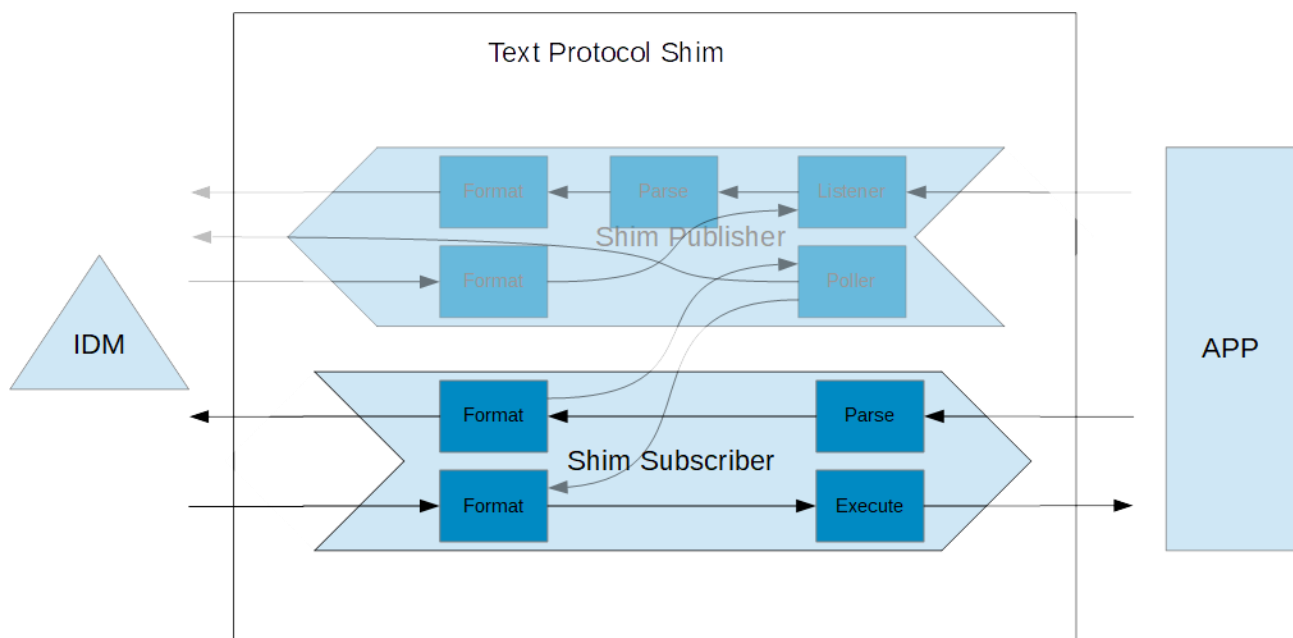
3.1 Architecture Overview

3.1.1 High level

The shim has the following architecture:

3.1.2 Shim Subscriber

The subscriber channel uses the following 4 components:



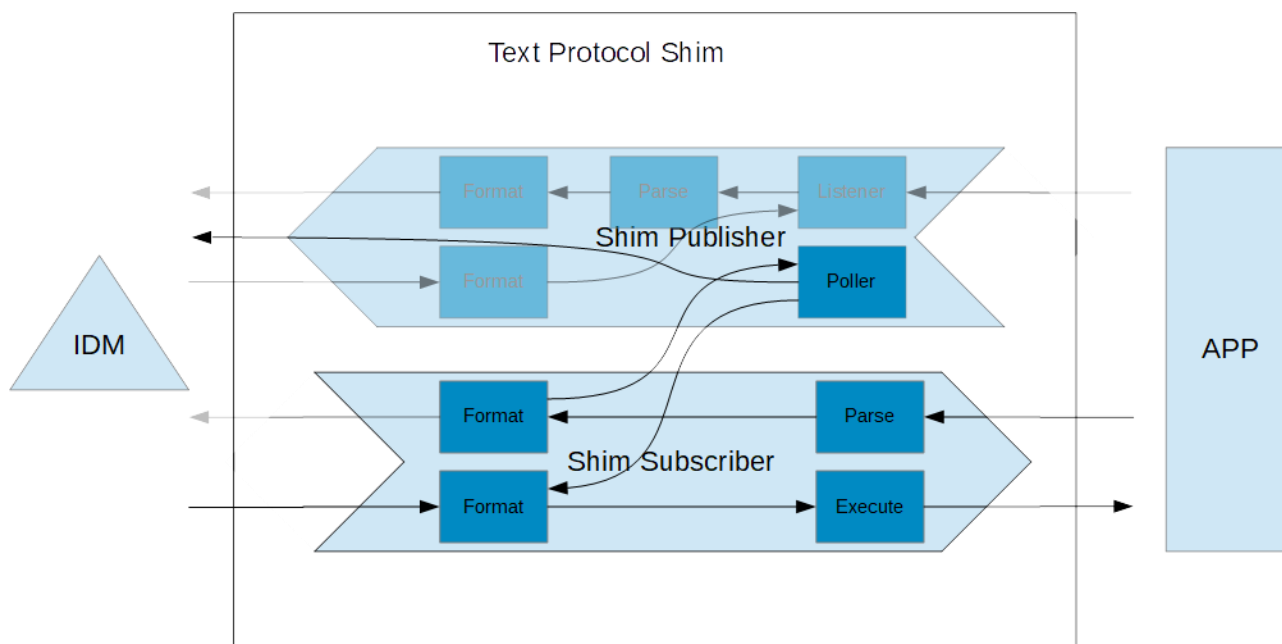
1. Request Formatter: this components formats the received XDS command into something the application expects: JSON, XML, SOAP, script,...
2. Executioner: the content is "executed". This execution can be literal (eg a script) or logical (eg: http put).

3. Response Parser: the response from the application is parsed into a usable java object.
4. Response Formatter: the response is formatted to something that the IDM engine can work with (minimal requirement: XML, preferably XSD).

3.1.3 Shim Publisher

The publisher has two major options: polling or listening. Both can be used at the same time if wanted.

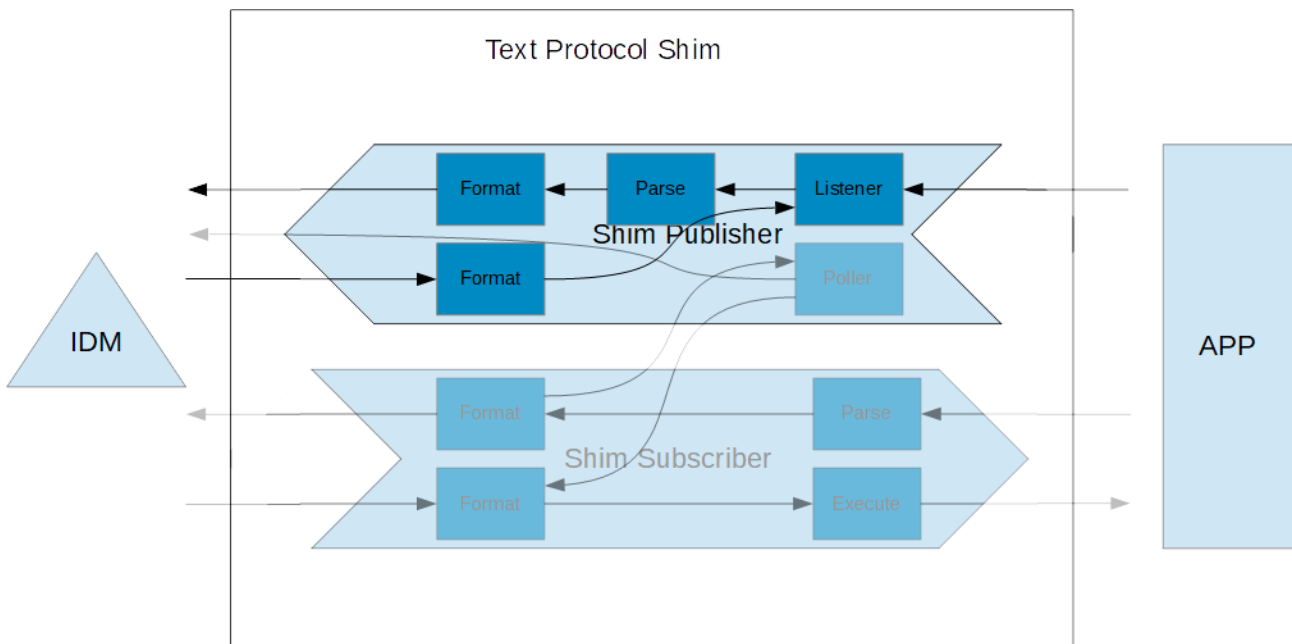
3.1.3.1 Poller



The poller uses the subscriber channel of the shim. It is an active component that interrogates the application. It adds one component.

1. Poller: the poller generates the queries to the application, and based on the formatted responses, calculates a delta and generates the needed events on the publisher channel.

3.1.3.2 Listener



The listeners is a passive publisher channel where the connected application reaches out to the shim. It has the following components:

1. Listener: this listens for incoming requests, and returns, after processing, the result back
2. Parser: the parser parses the incoming request into a java object for easy handling later on.
3. Request formatter: the formatter formats the parsed request into something that the IDM engine understands (minimal requirements: XML, preferably XSD).
4. Response formatter: the response formatter formats whatever is returned by the IDM engine to a format suitable for the connected application.

3.2 General concepts

3.2.1 Context parameters

The following context parameters are common in various places of the strategies:

- where javascript is used to calculate a value.
- during actual processing of the strategy.

Parameter	Description	Available in
transaction	The current transaction. Object of type <code>ISubTransaction</code> or <code>IPubTransaction</code> .	all
driver	A wrapper of the driver configuration (connect string, username and password, filter), driver persistent data and results of the last initialization cycle. Object of type <code>IdriverAbstraction</code> .	all



Parameter	Description	Available in
trace	Trace object that enables you to write data to the IDM trace file. Object of type <code>com.novell.nds.dirxml.driver.Trace</code>	all
stderr	Allows the code in the script engine to redirect stderr for capturing error conditions. Object of type <code>java.io.PrintStream</code>	ScriptExecuteStrategy
stdout	Allows the code in the script engine to redirect stdout for capturing the script response. Object of type <code>java.io.PrintStream</code>	ScriptExecuteStrategy

3.2.2 Velocity formatting

Velocity is the template engine that is used to format request & responses on both the subscriber and publisher. It contains some simple coding constructs (<http://velocity.apache.org/engine/devel/user-guide.html>) to generate text.

The engine is preconfigured with the following tools (<http://velocity.apache.org/tools/2.0/>):

- `org.apache.velocity.tools.generic.ClassTool`
- `org.apache.velocity.tools.generic.ComparisonDateTool`
- `org.apache.velocity.tools.generic.ConversionTool`
- `org.apache.velocity.tools.generic.DisplayTool`
- `info.vancauwenberge.idm.driver.txtprotocol.velocity.EscapeTool`
- `org.apache.velocity.tools.generic.FieldTool`
- `org.apache.velocity.tools.generic.MathTool`
- `org.apache.velocity.tools.generic.NumberTool`
- `org.apache.velocity.tools.generic.ResourceTool`
- `org.apache.velocity.tools.generic.SortTool`
- `org.apache.velocity.tools.generic.XmlTool`
- `info.vancauwenberge.idm.driver.txtprotocol.velocity.OauthTool`
- `info.vancauwenberge.idm.driver.txtprotocol.velocity.DnTool`
- `info.vancauwenberge.idm.driver.txtprotocol.velocity.StringTool`

You can configure the tools by updating the file `info\vancauwenberge\idm\driver\txtprotocol\velocity\configuration.xml` in the shim's jar.

The following extensions have been added to Velocity:

- `EscapeTool` has been extended to add support for escaping Json and bash scripts.
- `OauthTool` has been added to handle cryptographic and encoding use cases.
- `DnTool` has been added to extract elements (values and types) from a dn. It supports ldap and (q)slash formats by auto-detection.

- `StringTool`: a wrapper around Apache's `org.apache.commons.lang.StringUtils`

3.2.3 Transaction object

The transaction object is updated as it travels through the shim.

3.2.3.1 Common transaction features

3.2.3.1.1 Operation data

Each transaction can store operation data. This data is kept for the duration of the transaction.

Operation data is a key-value pair where the key is a string, and the value can be any object. You can access the operation data using the following methods on the transaction object:

method	Description
<code>setOperationData(String name, Object value)</code>	Set operations data with the given name. This will overwrite any previously stored operation data with this name.
<code>getOperationData(String name)</code>	Retrieve operation data with the given name.
<code>containsOperationData(String name)</code>	Tests if operation data with the given name is known.

During a polling cycle, the poller pre-sets some operation data. See chapter 2.4.1: Polling.

3.2.3.1.2 Tracing sensitive data

Depending on the trace level, the shim traces the formatted requests and responses. Since this can contain sensitive data, the shim cannot blindly dump all data in the trace. It masks data using the following rules:

- The values of the known sensitive attributes as defined by the filter/class definition.
- The values of the attributes that are marked as sensitive in the XSD command.
- Data that is marked as sensitive by the shim configurator via the transactions `addSensitiveData()`. This allows data that is converted or transformed in the formatting strategy, or that is returned from the application, to be masked as well.

The value to mask will become '***' in the trace.

Query

3.2.3.2 Subscriber channel

The following tables tries to give an overview of the various stages of this object on the subscriber channel. After each stage, more information is stored in the transaction object.

Stage	Result retrievable by
Receive an XSD command	<code>getXDSCCommand()</code>

Stage	Result retrievable by
Format request	<code>getFormattedRequest()</code>
Execute request	<code>getRawResponseString()</code> <code>getRawResponseObject()</code> <code>getRawMetaData()</code> <code>shouldInitialize()</code> <code>getExecuteException()</code>
Parse response	<code>getParsedResponse()</code>
Format response	Returned as a response to the subscriber channel.
Chained transaction	<code>getChainDepth()</code> <code>getParentTransaction()</code>

During every step, you have access to attributes stored in the vault via the queryhandler (`getQueryHandler()`). It allows retrieving attributes of the object in the current transaction via `getAttribute()` (even if the current command did not contain this attribute) or of any other object in the identity vault (`getSourceAttribute()`).

3.2.3.3 Publisher channel

Stage	Result retrievable by
Receive an application request	<code>getRawStringRequest()</code> <code>getRawObjectRequest()</code> <code>getRawMetaData()</code>
Parse the request	<code>getParsedRequest()</code>
Format the request to XSD	<code>getXSDFormattedRequest()</code>
Submit the XSD on the publisher channel	<code>isSubmitted()</code> <code>getResponseList()</code>
Format response	Returned as a response to the application.

3.3 Subscriber specific concepts

3.3.1 ICommand

The transaction contains a reference to the command received by the shim. This command allows simplified access to the various attributes in the command.

Command	attributes	getOperation
add	All attribute nodes from the add event. If a password element is present, an attribute 'password' will be added, overruling any password attribute that was present.	ADD
delete	Empty	DELETE

Command	attributes	getOperation
modify	The attribute nodes.	MODIFY
modify-password	A 'password' attribute.	MODIFYPASSWORD
move	The following attributes: <ul style="list-style-type: none"> · src-dn · qualified-src-dn · parent-src-dn · parent-qualified-src-dn · parent-dest-dn 	MOVE
query	The matching attributes (search-attr)	QUERY
query-ex	The matching attributes (search-attr), if any.	QUERY
rename	The following attributes: <ul style="list-style-type: none"> · src-dn · qualified-src-dn · new-name 	RENAME
init	empty	INIT
querypoll		QUERY

3.3.2 Chaining

The shim allows chaining of commands on the subscriber channel.

This means that one XSD command can result in multiple iterations of format-execute-parse-format cycles. This can be useful if eg the target application does not allow to set some attributes upon an 'add' event. One could either write policies to react on an add-association response to execute a modify, or one could configure the shim to execute a 'modify' after the initial 'add'.

The second scenario is done by using chaining. An ECMAScript expression determines whether or not a given operation should be chained. The depth of chaining is limited by the memory.

This can be presented as follows:

On each iteration, the previous transaction is stored as a parent transaction object in the current transaction. The `getChainDepth()` method returns how deep in the chain you are. All transactions in the chain share a single XSDCommand (the initial command that was received on the subscriber).

3.3.3 Initialization

The various execution strategies detect (lazy) if they need to initialize. This can be configured per strategy. It is also possible to configure a initialization at start of the shim.

Initialization injects a transaction of type `IInitTransaction` into the subscriber channel of the shim. This `IInitTransaction` contains a pseudo init XSD command where the `getOperation()` method returns the INIT operation. The resulting response of this transaction is not returned to the IDM engine, but stored in the transaction object itself. This can then be retrieved via the 'driver' context parameter using `driver.getLastInitOperation()` whenever needed (eg if it contains a token that must be used in every subsequent execution of any other transaction). The initialization transaction can also be chained if needed (see chapter 2.3.2: Chaining).

3.3.4 Token replacement

The subscribe side supports the token `{ $association }` inside the commands received. This token will be replaced with the value of the last known `<add-association>` value returned in the current batch. If the current batch does not contain an add that returned an `<add- association>`, the token will be replaced with an empty string.

This allows one to send the following (as one batch) to the shim:

```
<add class-name="user" dest-dn="aDn">
  <add-attr attr-name="uid">
    <value>testHtt</value>
  </add-attr>
</add>
<modify class-name="group">
  <association>ftp</association>
  <modify-attr attr-name="user">
    <add-value>
      <value association-ref="{ $association }" type="dn">aDn</value>
    </add-value>
  </modify-attr>
</modify>
```

If sent in one batch, and the add operation returns an add-association element, then the `{ $association }` will be replaced in the modify operation, before any formatting of the modify operation is performed, with the value of the add-association. This does not do any further validation (eg: no validation is done to make sure that that the object class matches or that the DN's match).

If you need this token as a literal, use `{{ $association }}` to escape it.

3.4 Publisher specific Concepts

3.4.1 Polling

3.4.1.1 Introduction

The publisher can be configured to poll the application. This is, configuration wise, the simplest form of enabling the publisher channel, but has, resource wise, the biggest impact on the system. Resources are CPU, memory and disk space.

3.4.1.2 Operation data

Polling injects query events into the subscriber shim. The query will have the following operation data set:

- `_previousPollTime_`: String, the time that the previous polling for this class started (in milliseconds).
- `_previousPollEndTime_`: String, the time that the previous polling for this class was completed (in milliseconds).
- `_thisPollTime_`: String, the time that this polling for this class started (in milliseconds). On a successful poll, the next polling cycle will have this value as `_previousPollTime_`
- `_isPoll_`: Boolean, set to true.

These can be used to eg request the changes in the connected system since the last polling cycle.

3.4.1.3 Processing

Each parsed and formatted result is interpreted as follows:

- instance: used to create a delta between two polling cycles. The delta is using the association to identify objects:
 - For objects not found during the last polling cycle, add events will be generated.
 - For objects no longer found in this polling cycle, delete events will be generated.
 - For objects that are changed, modify events will be generated.
- query and query-ex: are submitted back via the subscriber channel. Any operation data from the originating query is copied over.
- others: all other elements (add, modify, delete, move,etc) are send as publisher events as is.

Note: polling does not exclude listening.

3.4.2 Listening

The publisher can be configured to listen for incoming requests. Listening uses the publisher specific configuration for parsing and formatting.

Listening is, configuration wise, more complex (you need to configure parsers and formatters), but has in a typical set-up, less impact on the system.

Note: listening does not exclude polling.

4 Strategies (plugins)

4.1 Subscriber

4.1.1 Command formatting

Strategy	Description
Velocity	Uses Velocity for reformatting the XDS command. See chapter 2.2.2 (Velocity formatting) for more details.
Json	A configurable JSON formatter is included. This allows for basic JSON conversion. If

Strategy	Description
	more advanced conversion is required, you should use one of the other formatters above.

4.1.2 Command Execution

Strategy	Description									
HTTP	<p>Uses the http protocol to 'execute' the request.</p> <p>The subscriber REST execution strategy sets the following meta data, retrieveable via <code>getRawMetaData()</code>:</p> <table border="1"> <thead> <tr> <th>Key</th> <th>Class</th> <th>Description</th> </tr> </thead> <tbody> <tr> <td>status</td> <td>Integer</td> <td>The status code of the HTTP request.</td> </tr> <tr> <td>headers</td> <td>Map<String, String></td> <td>All response headers returned.</td> </tr> </tbody> </table>	Key	Class	Description	status	Integer	The status code of the HTTP request.	headers	Map<String, String>	All response headers returned.
Key	Class	Description								
status	Integer	The status code of the HTTP request.								
headers	Map<String, String>	All response headers returned.								
Script	<p>Uses java's scriptengine framework to execute scripts on demand. By default, java has the javascript scriptengine, but others should work as well.</p> <p>The subscriber Script execution strategy sets the following meta data, retrieveable via <code>getRawMetaData()</code>:</p> <table border="1"> <thead> <tr> <th>KeyS</th> <th>Class</th> <th>Description</th> </tr> </thead> <tbody> <tr> <td>stderr</td> <td>String</td> <td>String containing all text written to stderr during execution.</td> </tr> <tr> <td>stdout</td> <td>String</td> <td>String containing all text written to stdout during execution.</td> </tr> </tbody> </table>	KeyS	Class	Description	stderr	String	String containing all text written to stderr during execution.	stdout	String	String containing all text written to stdout during execution.
KeyS	Class	Description								
stderr	String	String containing all text written to stderr during execution.								
stdout	String	String containing all text written to stdout during execution.								
File	<p>Uses plain files to 'execute' the plugin:</p> <ul style="list-style-type: none"> · It dumps the command to a file and · reads the response from another file. <p>Useful for debugging or developing purposes (eg: if the SOAP or REST end-point is not yet available when you start developing your driver).</p> <p>On the file written, user defined attributes are set (if the OS & file system support this). On the file read, user defined attributes of the file are put in the metadata map.</p> <p>Note: user defined file attributes in Linux are set using</p> <pre>setfattr -n user.comment -v "home sweet home" /path/toFile</pre> <p>On windows, user defined attributes are stored as Alternate Data Streams (ADS). You can list the ADS using powershell:</p> <pre>Get-Item -Path C:\path\to\file -stream *</pre> <p>View the content of a specific stream ('method' in this example):</p>									

Strategy	Description
	Get-Content -Path C:\temp\stdOut.txt -stream method

4.1.3 Response Parsing

Strategy	Description
JSON	Parses the response to a JSON object.
JDOM	Parses the response to a JDOM2 document. Depending on te situation, JDOM2 can be more friendly to use than the standard W3C document.
W3C DOM	Parses the response to a standard W3C DOM document.
None	Does not parse anything. This will cause the corresponding context parameter to be null all the time.

4.1.4 Response Formatting

Strategy	Description
Velocity	Uses Velocity for reformatting the XDS command. See chapter 2.2.2 (Velocity formatting) for more details.

4.2 Publisher

When the publisher uses the polling mechanism, the subscriber channel is used. See the subscriber chapter above.

When the publisher is a passive listener component, the publisher channel of the shim uses it's own strategies.

Every incoming request is processed in the following sequence:

1. Request parsing
2. Submit judge
3. (optional) Request formatting
4. (optional) Submit on the publisher channel
5. Response formatting

The optional items (3 & 4) are only executed if the submit-to-shim judge returned true.

4.2.1 Request Parsing

Strategy	Description
JSON	Parses the request to a JSON object.
JDOM	Parses the request to a JDOM2 document. Depending on the situation, JDOM2 can be more friendly to use than the standard W3C document.
W3C DOM	Parses the request to a standard W3C DOM document.
None	Does not parse anything. This will cause the corresponding context parameter to be null all the time.

4.2.2 Request Formatting

Strategy	Description
Velocity	Uses Velocity for reformatting the request to an XDS command. See chapter 2.2.2 (Velocity formatting) for more details.

4.2.3 Response Formatting

Strategy	Description
Velocity	Uses Velocity for reformatting the response to the application's requested format. See chapter 2.2.2 (Velocity formatting) for more details.

5 Shim configuration

5.1 Subscriber

5.1.1 Main

Param	Name	Description
<code>sub.core.chain.judge.js</code>	Chain script	Javascript that returns true or false depending on whether the current transaction should be executed again (chained).
<code>sub.core.ignoreParserException</code>	Ignore response parser exceptions	Should exceptions thrown by the response parser be ignored (and the shim moves on to the next step: response formatting), or should this generate an error. When ignored, the transaction's <code>getParsedResponse()</code> will return null when an exception happened.
<code>sub.core.initOnboot</code>	Initialize on boot	This will not postpone the initialization until it is required (lazy initialization),

Param	Name	Description
		but will force an initialization upon starting of the driver. This does not exclude lazy initialization later on.
<code>sub.core.prettyprintResult</code>	Pretty print response	Should the shim pretty print the responses. This helps (readability) while developing a driver, but in rare occasions, it might impact your results as well (eg: an attribute value consisting only of white-space data).
<code>sub.core.querybackModifyCommand</code>	Query back for missing attributes	Should the shim perform a query back for missing attributes on a modify command
<code>sub.core.queryexSupported</code>	Is query-ex (paged search) supported	Is query-ex (paged search) supported by the target application and the implemented formatting logic.
<code>sub.command.strategy.execute</code>	Command execution	How will the command be executed.
<code>sub.response.strategy.format</code>	Response Formatter	How will the response be formatted. This formatter must format the response to one or more XDS command element(eg: add, add-association, status,...).
<code>sub.command.strategy.format</code>	Subscriber request formatting	How will the requests be formatted to the application's expected format.
<code>sub.response.strategy.parser</code>	Response Parser	Optional parsing of the response. The parsed response can be used in the response formatter later on.

5.1.2 Command formatting

5.1.2.1 VelocitySubRequestFormatStrategy

Param	Name	Description
<code>sub.outputformat.velocity.template</code>	Velocity Template	Contains the reference to the main velocity template.

5.1.2.2 JsonSubRequestFormatStrategy

Param	Name	Description
<code>sub.outputformat.json.arraysForClass.script.js</code>	Classes that must be a JSON array	By default, the root node in the generated JSon is not an array but an object. The classes in the returned array (javascript) will be formatted as an array. Example value: <code>dummy=["user", "group"]</code>
<code>sub.outputformat.json.arraysForAttributes.script.js</code>	Map of class-attribute list that must be a	By default, single attribute values in the

Param	Name	Description
<code>s</code>	JSON array	<p>generated JSon is not of type array but object. The map (class:attribute list) return by this ecmascript expression list the attributes for a given class that must be formatted as array values.</p> <p>Example value: <code>dummy={ "user": ["phone", "address"], "group"=["member"]};</code></p>

5.1.3 Command Execute

5.1.3.1 Http

Param	Name	Description
<code>sub.execute.rest.reinit.statusregex</code>	(Re)initialization status regex	<p>Regex that, when matched on the returned http status code, will trigger a (re)-initialization command. Initialization (authentication) is done lazy. Whenever the status code of the current operation matches the regex given, an init command will be executed, followed with a re-execution of the original command.</p> <p>Example value: <code>403 401</code></p>
<code>sub.execute.rest.reinit.rawresponseregex</code>	(Re)initialization raw response regex	<p>Regex that, when matched on the raw response, will trigger a (re)-initialization command. Initialization (authentication) is done lazy. Whenever the status code of the current operation matches the regex given, an init command will be executed, followed with a re-execution of the original command.</p> <p>Example value: <code>.*Access denied.*</code></p>

Param	Name	Description
<code>sub.execute.rest.exception.handle</code>	Http exception strategy	<p>What to do when the http operation itself returns an exception.</p> <ul style="list-style-type: none"> · JSONRPC20: create a Json error response as raw response string · SOAP11: create a SOAP1.1 fault response as raw response string · SOAP12: create a SOAP1.2 fault response as raw response string · THROW: throw the exception. · NDSSTATUS: generate a XDS error status element <p>Throwing the exception causes the transaction to stop. All others will continue the transaction (parse & format), allowing you to handle in the shim what should happen.</p>
<code>sub.execute.rest.keystore.file</code>	Keystore file	Optional keystore file. If using SSL, you can specify an optional driver specific keystore file. If not specified, the java default keystore will be used.
<code>sub.execute.rest.header.script.js</code>	Custom headers script	<p>Map (associative array) containing custom headers that must be added to the http request (eg: SOAPAction for SOAP calls or some authentication token). Example scripts:</p> <pre> * a={"SOAPAction":driver.getConnection()+"/"} * function getHeaders(command){ var a = {}; var xdsOperation = command.getOperation().toString(); if (xdsOperation=="modify"){ a["SOAPAction"] = "updateAction"+command.getOperationClass(); }else{ a["SOAPAction"] = command.getOperationClass(); } return a; } getHeaders(operation.getXDSCommand()); </pre>

Param	Name	Description
<code>sub.execute.rest.method.script.js</code>	HTTP Method script	<p>HTTP Method script returning GET,PUT, POST or GET for the current command. Note: SOAP uses POST for all operation. Example scripts:</p> <pre>* "POST" * function getMethod(command) { var xdsOperation = command.getOperation().toString(); if (xdsOperation=="query"){ return "GET"; }else{ return "POST"; } } getMethod(operation.getXDSCommand());</pre>

Param	Name	Description
<code>sub.execute.rest.url.script.js</code>	URL calculation script	<p>Script to calculate the URL.</p> <p>Example scripts:</p> <pre> * driver.getConnectionString() * function getConnectionString(connectString, command) { var className = command.getOperationClass(); var operation = command.getOperation().toString(); var association = command.getAssociation(); //Query without association value //=> The query needs to translate to URL parameters if (operation=="query" && !association){ var result = connectString+"/"+className+"?"; var attributeMap = command.getAttributes(); var keys = attributeMap.keySet(); //No matching attributes: read without any filter if (keys.size()==0){ return connectString+"/"+className; } for (var iterator = keys.iterator(); iterator.hasNext();) { var aKey = iterator.next(); result = result+aKey+"="+attributeMap.get(aKey)+"&"; } return result; } //Query without association value //=>Direct read of an instance based on association if (operation=="query"){ return connectString+"/"+className+"/"+associati on; } //Default: the url contains the className from the command. We can do POST, DELETE and GET operations return connectString+"/"+className; } getConnectionString(driver.getConnectionString(),operation.getXDSCCommand()); </pre>

5.1.3.2 Script

Param	Name	Description
<code>sub.execute.scriptengine.stderr.regexp.error</code>	Error regexp for stderr	Regexp that, when matched on stderr, will throw an exception during execution.
<code>sub.execute.scriptengine.stderr.regexp.reinit</code>	Re-initialize regexp for stderr	Regexp that, when matched on stderr, will trigger a (re)-initialization command. Initialization (authentication) is done lazy. Whenever stderr of the current

Param	Name	Description
		execution matches the regexp given, an init command will be executed, followed with a re-execution of the original command.
<code>sub.execute.scriptengine.stdout.regexp.error</code>	Error regexp for stdout	Regex that, when matched on stdout, will throw an exception during execution.
<code>sub.execute.scriptengine.stdout.regexp.reinit</code>	Re-initialize regexp for stdout	Regex that, when matched on stdout, will trigger a (re)-initialization command. Initialization (authentication) is done lazy. Whenever stdout of the current execution matches the regexp given, an init command will be executed, followed with a re-execution of the original command.
<code>sub.execute.scriptengine.shortname</code>	Engine name	The name of the engine to use. By default, java includes 'js' (ecmascript) script engine. See http://en.wikipedia.org/wiki/List_of_JVM_languages for more java script engines and their respective documentation on how to install them.

5.1.3.3 File

Param	Name	Description
TODO		

5.1.4 Response Parsing

5.1.4.1 JSON

No configuration parameters for JSON parsing.

5.1.4.2 JDOM

Param	Name	Description
<code>sub.parse.jdom.ignore_boundary_whitespace</code>	Ignore boundary whitespace	Should the DOM parser ignore boundary whitespace or not.
<code>sub.parse.jdom.validating</code>	Validating parser	Should the parser be a validating DOM parser or not.
<code>sub.parse.jdom.ignore_element_content_whitespace</code>	Ignore element content whitespace	Should the DOM parser ignore element content whitespace or not.

5.1.4.3 W3C DOM

Param	Name	Description
<code>sub.parsing.dom.validating</code>	Validating parser	Should the DOM parser be a validating DOM parser or not.
<code>sub.parsing.dom.ns_aware</code>	Namespace-aware parser	Should the DOM parser be namespace aware or not.

5.1.4.4 None

No configuration parameters when no parsing is done.

5.1.5 Response Formatting

5.1.5.1 Velocity

Param	Name	Description
<code>sub.format.response.velocity.template</code>	Template reference	Velocity template.

5.2 Publisher

5.2.1 Main

Param	Name	Description
<code>pub.heartbeat.interval</code>	Heartbeat interval in minutes	Specify the heartbeat interval in minutes. Set this value to 0 to turn off the heartbeat.
<code>pub.listening.class</code>	Publisher Listener	Listening class for receiving publisher events. This allows an external partner to trigger the publisher channel. Listening can be combined with polling if needed.
<code>pub.request.parser.class</code>	Publisher request parser	Parser class used to parse what is received on the publisher channel.
<code>pub.polling.class</code>	Publisher Polling	Polling class for generating publisher events using the subscriber channel of the shim. This allows active polling of the connected system. Polling can be combined with listening if needed.
<code>pub.request.formatter.class</code>	Publisher Request formatter	Formatter for transforming the publisher request to XSD command documents.
<code>pub.response.formatter.class</code>	Publisher Response formatter	Formatter for transforming the result of the request from XSD to the application expected format.
<code>pub.submitjudge.class</code>	Submit Judge	Judge that determines if a given publisher request should be submitted to the engine

Param	Name	Description
		or not. When submitting is not needed, the shim skips the request formatting step, does not submit anything on the publisher thread to the engine, and executes the response formatting step to return the result.

5.2.2 Request parsing

5.2.2.1 JSON

No configuration parameters for JSON parsing.

5.2.2.2 JDOM

Param	Name	Description
<code>pub.parse.jdom.ignore_boundary_whitespace</code>	Ignore boundary whitespace	Should the DOM parser ignore boundary whitespace or not.
<code>pub.parse.jdom.validating</code>	Validating parser	Should the parser be a validating DOM parser or not.
<code>pub.parse.jdom.ignore_element_content_whitespace</code>	Ignore element content whitespace	Should the DOM parser ignore element content whitespace or not.

5.2.2.3 W3C DOM

Param	Name	Description
<code>pub.parsing.dom.validating</code>	Validating parser	Should the DOM parser be a validating DOM parser or not.
<code>sub.parsing.dom.ns_aware</code>	Namespace-aware parser	Should the DOM parser be namespace aware or not.

5.2.2.4 None

No configuration parameters when no parsing is done.

5.2.3 Submit Judge

5.2.3.1 Javascript

Param	Name	Description
<code>pub.submitjudge.js.submitjudge.js</code>	Submit script	Javascript, returning 'true' when the current transaction should be submitted to the engine, or 'false' when no.

5.2.4 Request Formatting

5.2.4.1 Velocity

Param	Name	Description
<code>pub.format.request.velocity.template</code>	Velocity Template	Contains the reference to the main velocity template.

5.2.5 Response Formatting

5.2.5.1 Velocity

Param	Name	Description
<code>pub.format.response.velocity.template</code>	Velocity Template	Contains the reference to the main velocity template.

5.2.6 Poller

Param	Name	Description
<code>pub.polling.custom.classlist</code>	Polling classes and attributes	Polling classes and attributes in the applications namespace. Each line can contain a class or a class and attributes in the format <class>[:attribute1,attribute2,attribute3]. Example: user:firstName,lastName
<code>pub.polling.interval</code>	Polling interval	Polling interval (in seconds).
<code>pub.polling.prettyprint</code>	Pretty print	
<code>pub.polling.type</code>	Polling classes	What classes should be polled. User defined allows entry of a list of classes and it's attributes, "filter" uses the driver filter and attributes.

5.2.7 Listener

5.2.7.1 Http

Param	Name	Description
<code>pub.listener.http.auth</code>	Publisher authentication	Authentication to use on the publisher. Values: <ul style="list-style-type: none"> · None · Basic · OSP OAuth2
<code>pub.listener.http.basicauth.password</code>	Password	For Basic authentication, password of authorized user.
<code>pub.listener.http.basicauth.user</code>	Username	For Basic authentication, username of authorized user.

Param	Name	Description
<code>pub.listener.http.bearerauth.issuer</code>	Bearer issuer	For OSP OAuth2: the osp oauth URL (the issuer to trust and consult). Eg: <code>https://osp.mycompany.com:8543/osp/admin/auth/oauth2</code>
<code>pub.listener.http.bearerauth.subjects</code>	Bearer subject to grant access	The bearer subject that will be granted access. This has the format of "oauth2- <code><clientID></code> ". Example: "oauth2-System1"
<code>pub.listener.http.bearerauth.keystore</code>	OAuth1 keystore file	Keystore file where OSP's public key is stored. Leave empty to use the JVM's default keystore. Note: the shim contains a Keytool helper.
<code>pub.listener.http.script.contenttype.js</code>	Result content type	Content type script. The result will be used to set the content type
<code>pub.listener.http.host</code>	Listening host	Host to listen on. Use 127.0.0.1 to listen to all IP addresses
<code>pub.listener.http.supported.methods</code>	Supported HTTP methods	Comma separated list of supported HTTP methods. 405 will be returned immediately on non supported methods without parsing, formatting etc. Blank means that all methods are supported.
<code>pub.listener.http.keystore.password</code>	Keystore password	Password of the keystore.
<code>pub.listener.http.keystore.path</code>	Keystore path	Path to the keystore file.
<code>pub.listener.http.port</code>	Listening port	Listening port.
<code>pub.listener.http.serverkey.alias</code>	Key Alias	Alias in the keystore of the server key to use.
<code>pub.listener.http.serverkey.password</code>	Key Password	Password of the above key.
<code>pub.listener.http.mutual</code>	Mutual SSL certificates	Is client authentication required
<code>pub.listener.http.ssl</code>	Use SSL	Use SSL protocol.

6 Installation

6.1 Remote loader vs non-remote loader

6.1.1 Non remote loader

This driver is, due to version conflicts in various jar's, only supported in a remote loader configuration.

6.1.2 Remote loader

The shim has been tested in the java (dirxml_jremote on Linux), Linux native (rdxml) and Windows remote loaders. In order to be able to use more recent version of various open source jars, and in order to be independent of remote loader upgrades or version, this shim includes it's dedicated jars in a separate lib folder. The configuration for this is straight forward:

1. Install the remote loader per documentation. Setting the remote loader password and driver password as required
2. Configure a remote loader instance, using the following class:
`info.vancauwenberge.idm.driver.txtprotocol.shim.driver.TxtProtocolShim`
3. Copy the Text Protocol shim jar (`HttpDriver_<version>.jar`) to the remote loaders classpath on the servers.

Eg: `/opt/novell/eDirectory/lib/dirxml/classes/`
4. Copy the jar dependencies from the lib folder to a dedicated path on the servers.

Eg: `/opt/txtpl/lib`
5. In the shim configuration, set the lib path (Driver Configuration → Driver Parameters → Driver Options) the the path where you copied the jar dependencies.

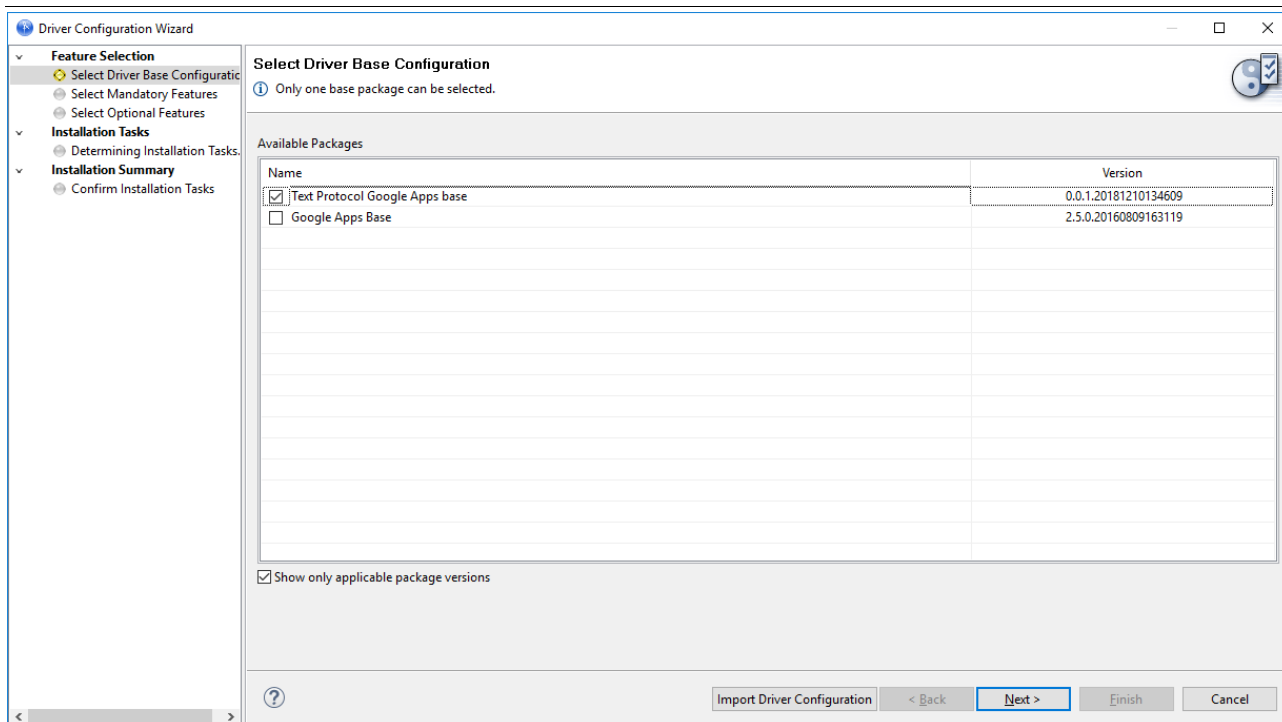
Eg: `/opt/txtpl/lib`
6. Configure and start the driver and remote loader as normal

6.2 Designer

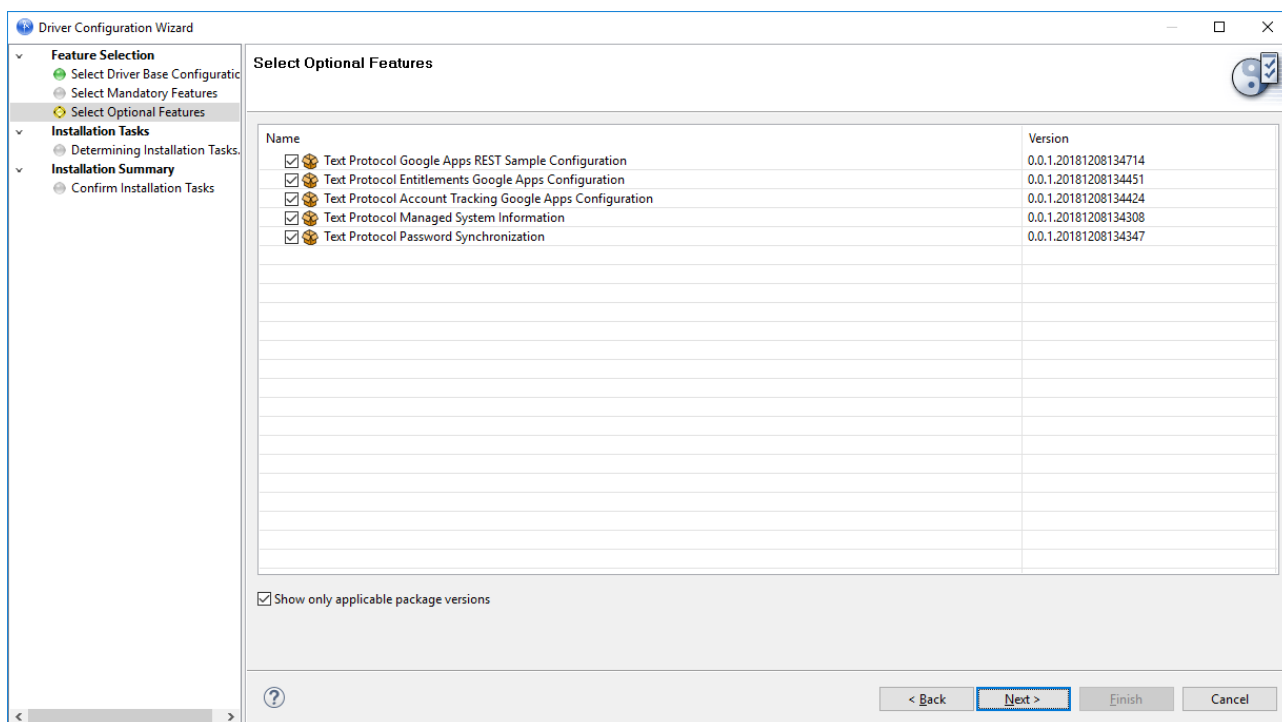
In designer, import the Text Protocol IDM packages, and drag-and-drop the required icon (REST, SOAP, Scripting, ...). Select the Text Protocol Base package and the desired options.

Example for Google apps:

1. Drag and drop the Google Apps from the palette onto your workspace.
2. Select “Text protocol Google Apps base” and click “Next”



3. Select all options required (by default, all options are selected) and click “Next”.



4. Provide the required information to connect to Google Apps. See chapter 7.3: Google Apps (HTTP)

7 Configuration

7.1 Velocity Templating

See the Velocity user guide for what Velocity is and how to use it. Velocity tools is configured by



default. The configuration can be changed by modifying the configuration.xml file located in the shim jar (info/vancauwenberge/idm/httpdriver/velocity/configuration.xml).

7.2 Context Parameters

Each javascript driver configuration parameter or velocity template has the following context variables/parameters:

Parameter	Description	Class	Example
driver	Enables access to driver settings: <ul style="list-style-type: none"> the driver's connection configuration (connectstring, username and password) allows storage and retrieval of state data the driver's configuration parameters the last know 'init' operation 	IDriverAbstracti on	driver.getConnectionStr ing()
trace	Contains a trace object for debugging purposes	com.novell.nds.d irxml.driver.Trac e	trace.trace("message ,0)
transacti on	The current transaction. This can either be an ISubTransaction or an IPubTransaction.	ISubTransaction IPubTransaction	transaction.getXDS Command().getAttri bute("CN")

7.2.1 OSP OAuth

When you want the publisher channel to be protected by OSP's OAuth, you need to configure OSP to allow additional clients. For this, edit the file ism-configuration.properties on your OSP server, adding, for each client, the following lines, where clientID is the 'username':

```
com.netiq.<clientID>.clientID = <clientID>
com.netiq.<clientID>.clientPass = <client password/secret>
com.netiq.<clientID>.response-types = client_credentials
```

8 Example Configurations

8.1 SLES user provisioning (scripting)

The SLES user provisioning configuration provisions users and groupmemberships to a SLES server (subscriber only). It uses the Script execution strategy to execute the needed Linux commands on the Linux server.

The script execution strategy uses whatever was generated by the template engine, as a script. In this example: a javascript. Due to this, you do not have to fear that upgrades will alter your server side scripts.

This javascript executes the following Linux commands:

- useradd
- userdel

- usermod
- id
- getent

8.2 Redhat IPA (HTTP)

8.2.1 Introduction

IPA has an online demo site (<https://www.freeipa.org/page/Demo>). This example configuration is configured to provision that site. It is a simple example that can be used for basic understanding of the shim's possibilities.

You can browse the IPA API at <https://ipa.demo1.freeipa.org/ipa/ui/#/p/apibrowser/type=command>

This configuration uses the following features:

- Authentication is form based (HTTP Post).
- Simple scripts for request URL and headers

8.2.2 Driver Configuration

To install the demo driver against the IPA demo site, do the following steps.

1. Download the SSL certificate from TODO and put it in a keystore on the remote loader server (eg /root/IPA.keystore)
2. Follow the steps in chapter 5.1.2 (Remote loader) to install/configure the remote loader.
3. In Designer, drag and drop the REST Server (Tool category) icon on the workspace.
4. Select the package “Text Protocol IPA base”
5. Select the optional features (all by default)
6. Confirm installation of the package dependencies.
7. Give the driver a name (default: HTTP IPA Rest Driver)
8. Configure the shim:

Shim parameter	Value (Example)	Description
Authentication ID	admin	An account with enough rights in the IPA system
Connection information	https://ipa.demo1.freeipa.org/ipa/session/	The URL of the demo application. Leave default.
Password	Secret123	The password of the above account. The demo system uses 'Secret123'

Shim parameter	Value (Example)	Description
Server lib path	/opt/txtpl/lib/	Path where you copied the java jar dependencies (step 2 above).
Keystore file	/root/IPA.keystore	Path to the keystore file created in step 1 above.

9. Specify the remote loader details
10. (Optional) Specify Managed System details
11. Click Finish and deploy the driver.

8.2.3 Driver Functions

This example IPA configuration synchronizes users and groupmemberships between IPA and the IDVault based on entitlements. Users are synchronized unidirectional (from the IDVault to IPA).

It synchronizes the following attributes:

IDVault	IPA	Matching	Comment
CN	uid	primary	
Full Name	displayname		
Internet EMail Address	mail		
Given Name	givenname		
Login Disabled	N/A		Used to disable/enable the account.
Surname	sn		
Telephone Number	telephonenumber		
nspmDistributionPassw ord	password		

8.3 Google Apps (HTTP)

8.3.1 Introduction

The Google Apps configuration implements a bidirectional sync between Google Apps and the IDVault.

The Google Apps configuration demonstrates the following features:

- HTTP execution strategy
- QueryHandler: Google apps supports delta updates (PATCH), but some attributes are objects or arrays of objects. These must always be submitted as a whole. Examples are the name

object (consisting of givenName and familyName) and phones array of objects (where each object has a type and a value). This is handled by the naming convention (either <attribute> or <object>.<attribute> or <array>.<type>.<attribute>).


In order to get the missing attributes (eg the surname when the modify event only contains the given name) from an object or an array of object, the template uses `transaction.getQueryHandler().getAttribute(String name)`. This either returns the attribute present in the current command, or queries back to the IDV to get the attribute.

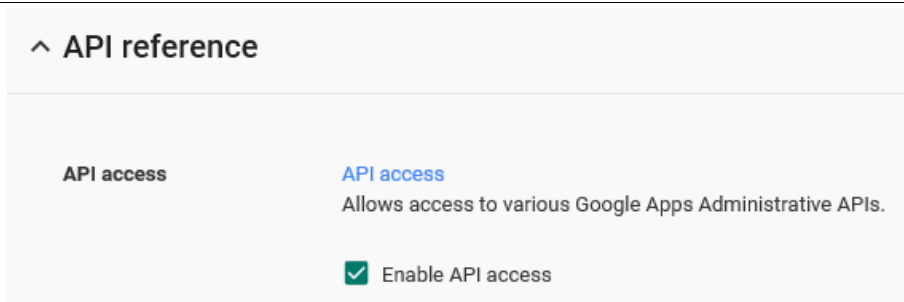
- Google Apps also demonstrates the use of OAuth to authenticate.
- Shim GCV's are used for the additional OAuth parameters needed:
 - `oauthImpersonateUserId`: userid for the OAuth claim.
 - `oauthScopes`: space separated list of scopes of the OAuth claim.
<https://www.googleapis.com/auth/admin.directory.group>
<https://www.googleapis.com/auth/admin.directory.user>
 - `oauthPrivateKeyId`: private key as generated on the account. This value is used to populate the 'kid' value in the JWT.
 - `oauthURL`: <https://accounts.google.com/o/oauth2/token>
 - `domain`: the domain to search in. Typically “mycompany.com”
- OAuth Velocity tool for base64 encoding and signing.
- Complex URL and header scripting
- Support for paged search. Google Apps requires that every request for a new page contains all query parameters. Since the IDM engine only provides these argument in the initial search, this configuration encodes these arguments and the Google Apps search token into a new search token. This is then, for all subsequent pages, used to pass on the needed info to Google Apps.

8.3.2 Driver configuration

8.3.2.1 Google Apps configuration

8.3.2.1.1 Enable API access

1. Open your browser to: <https://admin.google.com/<yourdomain>/AdminHome> where <yourdomain> is your domain name (eg “yourdomain.com”)
2. Open the Security dashboard by clicking the icon (by default on the bottom): 
3. Select “API reference”
4. Enable “API Access”

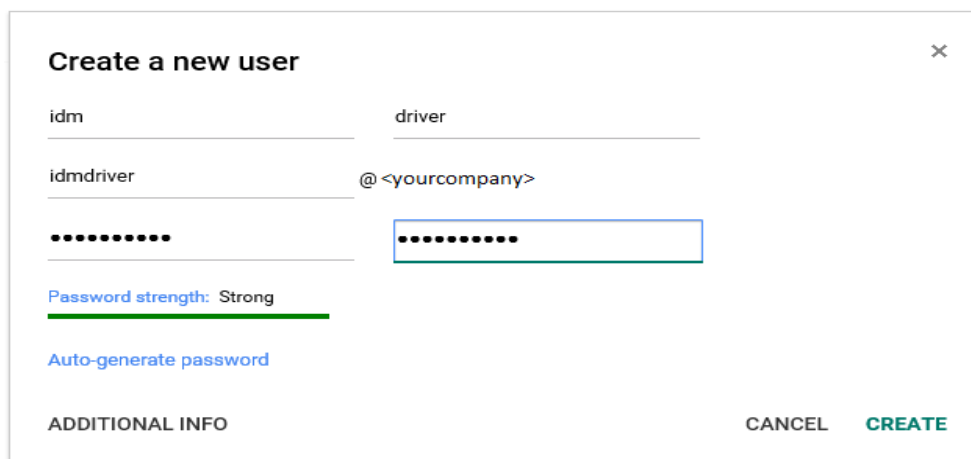


5.

8.3.2.1.2 Create a Google user account

This user account will be used by the driver as the user to impersonate as. The service account later on, will impersonate this user when doing actions in Google Apps.

1. Open your browser to: <https://admin.google.com/<yourdomain>/AdminHome> where <yourdomain> is your domain name (eg “yourdomain.com”)
2. Goto Users to create a new user. Enter the name, email and password.

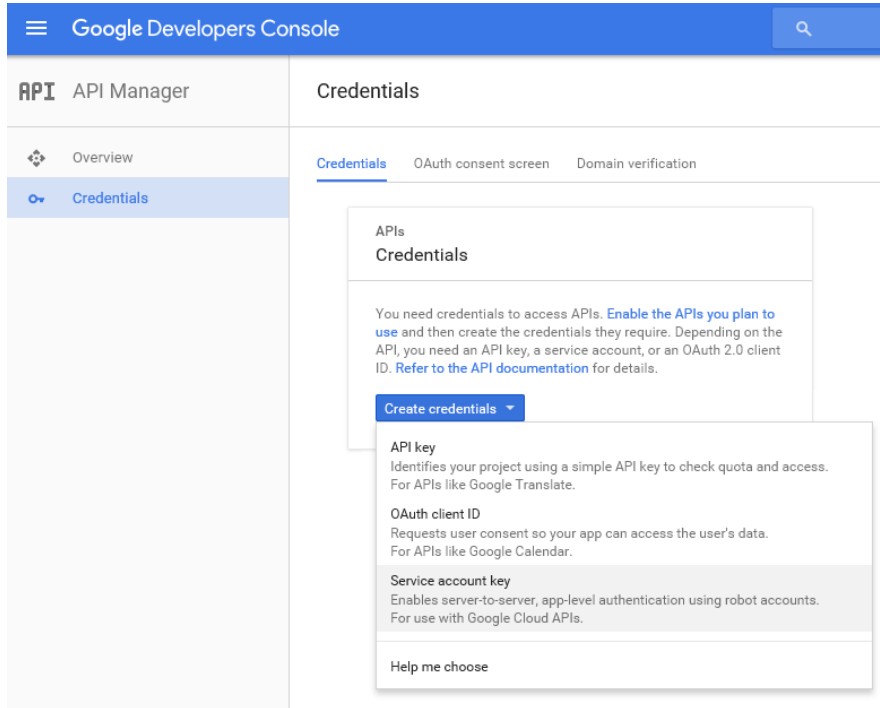


3.

4. Edit this user, and add the roles “Services Admin” and “User Management Admin”.
5. Log out and log in with the above account to accept the Google term of service.

8.3.2.1.3 Generate a service account key

1. Open the Google Developer console at: <https://console.developers.google.com/apis/credentials>, creating a new project if non was created before.
2. Create a new service account key.



- 3.
4. Enter a name, and specify the format “JSON”.
- 5.
6. Save the Json file in safe location for later use. Whenever you need to (re)configure the shim, you need the data in this file.

8.3.2 Shim Installation

1. Follow the steps in chapter 5.1.2 (Remote loader) to install/configure the remote loader.
2. In Designer, drag and drop the Google Apps (Enterprise category) icon on the workspace.
3. Select the package “Text Protocol Google Apps base”
4. Select the optional features (all by default)
5. Confirm installation of the package dependencies.
6. Give the driver a name (default: Google Apps)
7. Configure the shim as follows, using the data from the Json file generated above:

Shim parameter	Json value	Description
Authentication ID	Field “client_email”	service account id. Eg: idmdriver@compose-agu-111.iam.gserviceaccount.com.



Shim parameter	Json value	Description
Password	Field "private_key"	Copy paste the private key. This is everything between " <i>-----BEGIN PRIVATE KEY-----\n</i> " and " <i>\n-----END PRIVATE KEY-----\n</i> ", and removing all '\n' character sequences. This is a long string of more than 1000 characters long! Eg: assuming the value is " <i>-----BEGIN PRIVATE KEY-----\nThis\nIs\nThe\nValue\n-----END PRIVATE KEY-----\n</i> ", then the password is: " <i>ThisIsTheValue</i> " (without the quotes).
Server lib path	N/A	Path where you copied the java jar dependencies (step 1 above).
oauthImpersonateUserId	N/A	The user account created above. Eg: admin@yourdomain.com
oauthScopes	N/A	This declares the rights that will be asked authorization for by the shim. Leave the default.
oauthPrivateKeyId	Field "private_key_id"	Private key. A hex looking string of about 40 characters.
domain	N/A	Your Google Apps domain
oauthURL	Field "token_uri"	OAuth endpoint of Google Apps. Should be https://accounts.google.com/o/oauth2/token

8. Specify the remote loader details
9. (Optional) Specify Managed System details
10. Click Finish and deploy the driver.

8.3.3 Driver Functions

This example Google Apps configuration synchronizes users and groupmemberships between Google Apps and the IDVault based on entitlements. Users are synchronized bidirectional (except for the password).

It synchronizes the following attributes:

IDVault	Google Apps	Matching	Comment
assistantPhone	phones.assistant.value		
CN	externalIds[] of type login_id	primary	
co	addresses[].country of type work		
company	organizations[].name of type work		
costCenter	organizations[].costcenter of type work		
Facsimile Telephone Number	phones[] of type work_fax		
Full Name	name.fullName		Generated By Google apps. Only on publisher channel
Given Name	name.givenName		
homeCity	addresses[].locality of type home		
homeFax	phones[] of type home_fax		
homePhone	phones[] of type home		
homePostalAddress	addresses[].streetAddress of type home		
homeState	addresses[].region of type home		
homeZipCode	addresses[].poBox of type home		
Internet EMail Address	primaryEmail	secondary	
L	organizations[].location of type work		
Language	languages		The language should be one in the list at https://developers.google.com/admin-sdk/directory/v1/languages
Login Disabled	suspended		
mobile	phones[] of type mobile		
nspmDistributionPassword	password		Only sync from IDVault to Google Apps.
otherPhoneNumber	phones[] of type other		
OU	organizations[].department of type work		
pager	phones[] of type pager		
Physical Delivery	addresses[].locality of type		

IDVault	Google Apps	Matching	Comment
Office Name	work		
Postal Code	addresses[].postalCode of type work		
Postal Office Box	addresses[].poBox of type work		
S	addresses[].region of type work		
SA	addresses[].streetAddress of type work		
Surname	name.familyName		
Telephone Number	phones[] of type work		
telexNumber	phones[] of type telex		
Title	organizations[].title of type work		

9 Keytool helper

In order to simply create or extend any keystore with trusted certificates, the shim provides a keytool helper. This keytool helper can validate a keystore against a host, or can import certificates into a keystore from a host when needed (host not yet trusted by the keystore).

```
java -cp <path to shim> info.vancauwenberge.idm.driver.txtprotocol.util.KeyTool
<command> <-options>
```

Where command and options are:

Command	Description	Options
UpdateKeystore	Update or create a keystore with one or more certificates from the given host/port	Required: <ul style="list-style-type: none"> · -host: the host · -keystore: path to the keystore file Optional: <ul style="list-style-type: none"> · -passphrase. Defaults to 'changeit' · -port. Defaults to '443'
TestKeystore	Test the certificates in a given keystore against the given host/port	Required: <ul style="list-style-type: none"> · -host: the host · -keystore: path to the keystore file Optional: <ul style="list-style-type: none"> · -passphrase. Defaults to 'changeit' · -port. Defaults to '443'

Command	Description	Options
AutoConfigure	Import the SSL chain of the given host/port if not yet present in the given keystore.	Required: <ul style="list-style-type: none"> · -host: the host · -keystore: path to the keystore file Optional: <ul style="list-style-type: none"> · -passphrase. Defaults to 'changeit' · -port. Defaults to '443'

Example usage:

```
java -cp <path to shim> info.vancauwenberge.idm.driver.txtprotocol.util.KeyTool
updatekeystore -host 172.217.19.206 -keystore c:\Temp\testKeysoter.jks -port 443
-passphrase test
```

10 Third party libraries and licenses

Except the obvious dependency on the IDM framework, the Text Protocol Driver depends on the following third party libraries:

Package	License	URL
anakia-1.0.jar	Apache Software License, Version 2.0	https://velocity.apache.org/anakia/1.0/
antlr-2.7.7.jar	public domain	http://www.antlr.org/
commons-beanutils-1.7.0.jar	Apache Software License, Version 2.0	https://commons.apache.org/proper/commons-beanutils/
commons-codec-1.9.jar	Apache Software License, Version 2.0	https://commons.apache.org/proper/commons-codec/
commons-collections-3.2.1.jar	Apache Software License, Version 2.0	https://commons.apache.org/proper/commons-collections/
commons-digester-1.8.jar	Apache Software License, Version 2.0	https://commons.apache.org/proper/commons-digester/
commons-lang-2.6.jar	Apache Software License, Version 2.0	https://commons.apache.org/proper/commons-lang/
commons-logging-1.1.3.jar	Apache Software License, Version 2.0	https://commons.apache.org/proper/commons-logging/
dom4j-1.6.1.jar	BSD style license	http://dom4j.sourceforge.net
eclipse-collections-7.1.0.jar	Eclipse Public License - v 1.0	https://projects.eclipse.org/projects/technology.collections
eclipse-collections-api-7.1.0.jar	Eclipse Public License - v 1.0	https://projects.eclipse.org/projects/technology.collections
eclipse-collections-forkjoin-7.1.0.jar	Eclipse Public License - v 1.0	https://projects.eclipse.org/projects/technology.collections

Package	License	URL
elsa-3.0.0-M5.jar	Apache Software License, Version 2.0	https://github.com/jankotek/elsa/blob/master/LICENSE.txt
guava-19.0.jar	Apache Software License, Version 2.0	https://github.com/google/guava
httpclient-4.3.3.jar	Apache Software License, Version 2.0	https://hc.apache.org/httpcomponents-client-ga/
httpcore-4.3.2.jar	Apache Software License, Version 2.0	https://hc.apache.org/httpcomponents-core-ga/
jaxen-1.1.6.jar	BSD style license	http://jaxen.org/
jdom-1.1.3.jar	Apache-style	http://jdom.org/docs/faq.html#a0030
jdom2-2.0.5.jar	Apache-style	http://jdom.org/docs/faq.html#a0030
jetty-all-9.3.7.RC1-uber.jar	Apache Software License, Version 2.0 Eclipse Public License 1.0	https://hc.apache.org/httpcomponents-client-ga/ https://projects.eclipse.org/projects/technology.collections
jjwt 0.9.0	Apache Software License, Version 2.0	https://github.com/jwtk/jjwt
kotlin-runtime-1.0.2.jar	Apache Software License, Version 2.0	https://hc.apache.org/httpcomponents-client-ga/
kotlin-stdlib-1.0.2.jar	Apache Software License, Version 2.0	https://hc.apache.org/httpcomponents-client-ga/
lz4-1.3.0.jar	Apache Software License, Version 2.0	https://hc.apache.org/httpcomponents-client-ga/
mapdb-3.0.1.jar	Apache Software License, Version 2.0	https://hc.apache.org/httpcomponents-client-ga/
slf4j-api-1.7.5.jar	MIT license.	https://www.slf4j.org/license.html
velocity-1.7.jar	Apache Software License, Version 2.0	https://hc.apache.org/httpcomponents-client-ga/
velocity-tools-2.0.jar	Apache Software License, Version 2.0	https://hc.apache.org/httpcomponents-client-ga/