CD SWAT

DevTest 9.5.1

How-To

Editing a Virtual Service Image (VSI)

Prepared by: Surya Suravarapu

Date: November 2016

Table of Contents

Purpose

The virtual service image (vsi) file is a critical component that makes up the virtual service. The purpose of this document is to describe the vsi file and talk about different matching mechanisms, match scripts and other concepts like magic strings.

Virtual Service Image

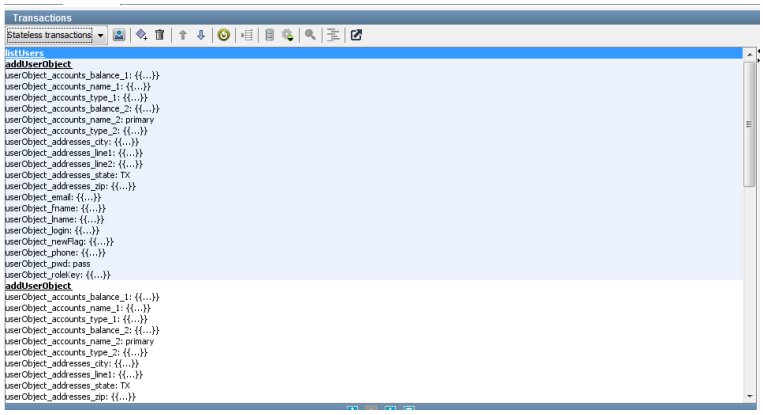
	A Virtual service comprises of two files, the virtual service image (vsi) file and a virtual service model (vsm) file. A vsi file is the file which contains the actual request and response information along with definition of the matching criteria including match scripts. Let's explore the different components of a vsi file below.

Components of a vsi file

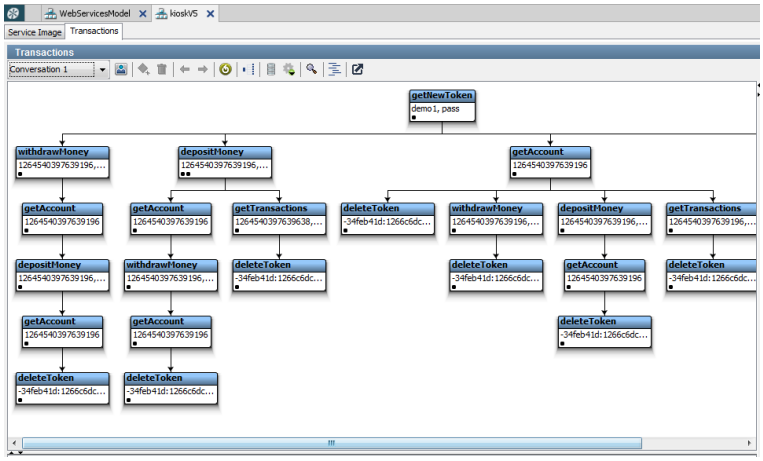
	A vsi file comprises of the below components / sections. Vsi file comprises of two tabs a. Transactions ➢ The default view when we open a vsi, it comprises of the actual request / response information including arguments, attributes, meta data and match criteria including match script. b. Service Image ➢ This tab consists of Image Name field and the response definitions for unknown stateless and conversational requests (response in case of a request mismatch).

Transactions Tab

Stateless Transactions



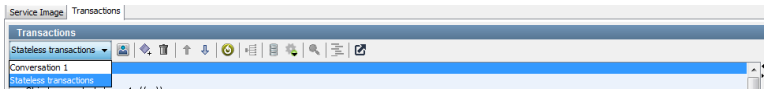
Conversations



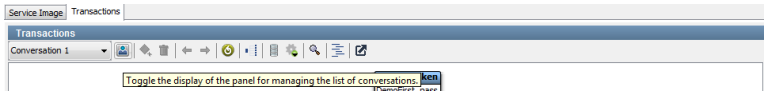
The transactions tab comprises of different transactions that make up the contents of the virtual service. The transactions inside a virtual service can be either stateless transactions or conversations depending on whether a service is stateful or stateless or both.

Some Useful Functions

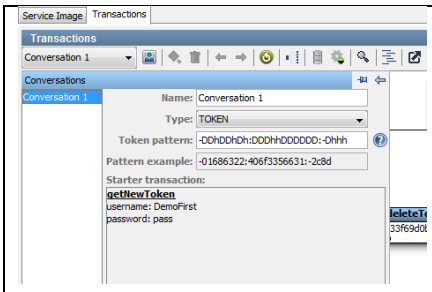
Transactions Selection

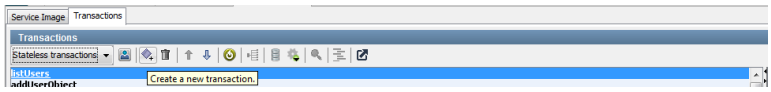


Managing Conversations

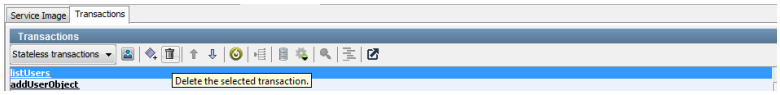


Some of the useful functions in the Transactions tab are as given.

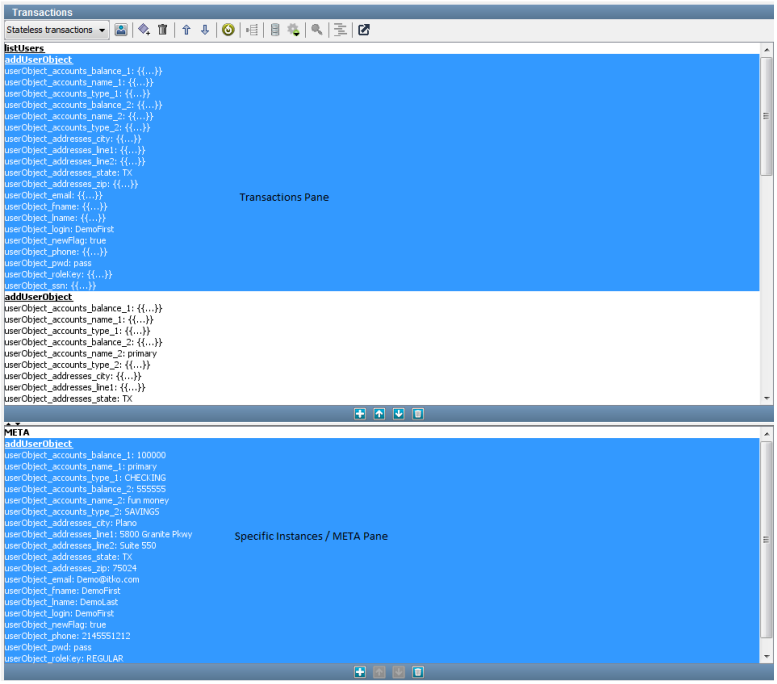




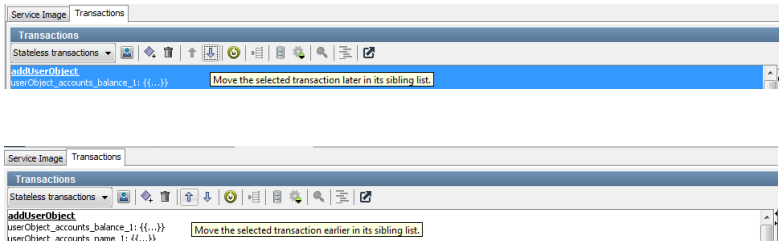
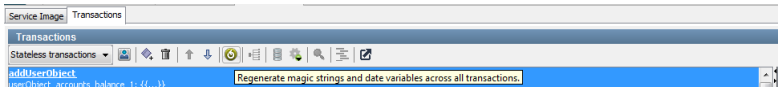
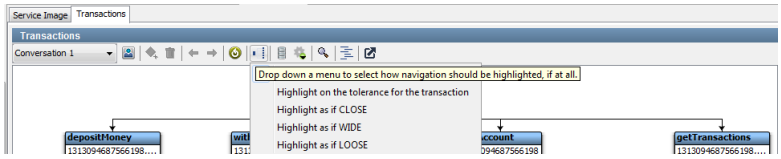
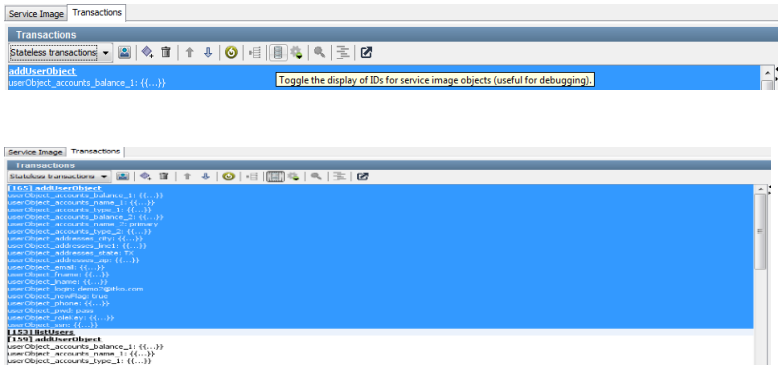
Create new transactions
This will help us add new Transactions to the existing list. This requires more effort as we need to configure everything from operation name to response xml manually. This can also be achieved by clicking on the just below transactions frame.



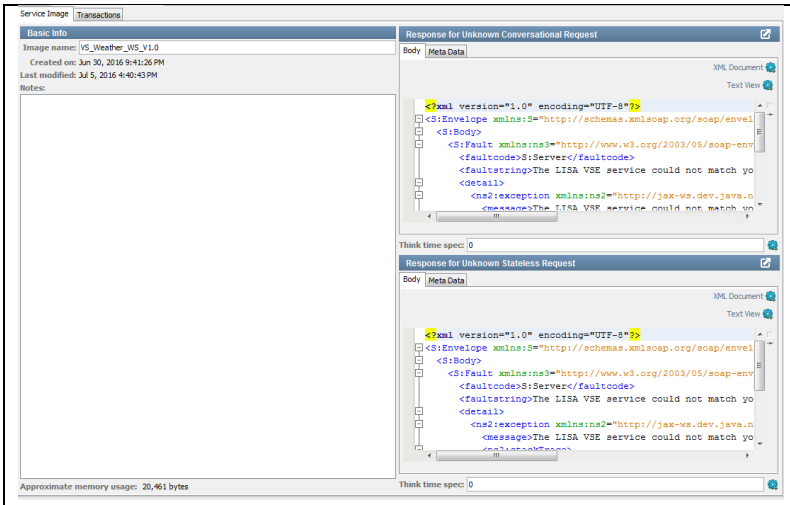
Delete transactions
This will help us with deleting transactions (including META and specific instances) from the existing list. This can also be achieved by clicking on below the transactions frame.



We can also delete specific transaction instances or META instance under a transaction by clicking on below the META / Specific instances frame.

	**Move transactions** This will help us with moving the transactions up and down in the image file. This will help adjust the order in which matching happens for the transactions. The transactions which are earlier in the sibling list get matched first and so on.
	**Regenerate Magic Strings** This will help us with regenerating the magic strings and date variables across all transactions. This is useful if we make changes in the responses / request arguments and would like the magic string to be regenerated accordingly (across all transactions). We can also do it individually at instance level in transaction by selecting Magic String check box under Request Data-> Arguments section.
	**Navigation highlight** This will help us select options for navigation highlight for stateful conversations.
	**Toggle ID Display** This will help us toggle the display of ID for transactions. This ID is very helpful in debugging to identify which instance is getting matched etc.

Service Image Tab

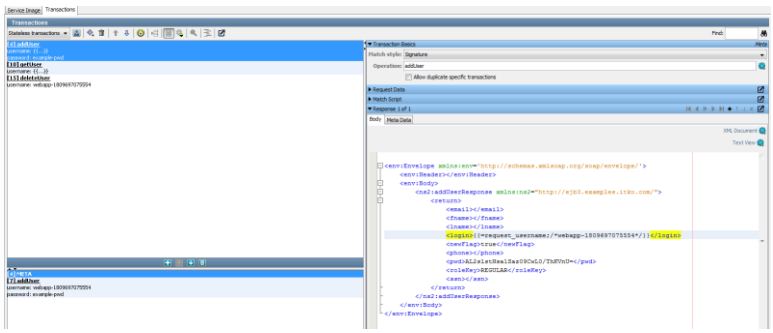
	Service Image tab comprises of the virtual service image file name and the responses to be sent back in case of an unknown request (conversational / stateless). Unknown request is a request that fails to match with any of the recorded / captured transactions. This tab also provides an estimate of the memory usage by the image file.

Magic String and Magic Dates

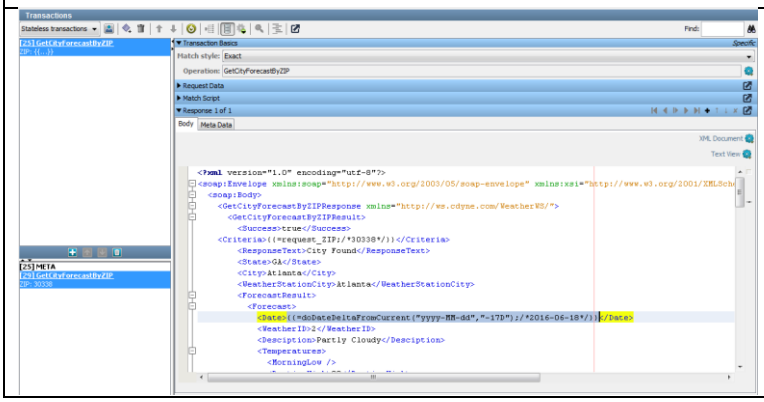
	Magic String is a very useful feature of DevTest, the gives virtual services ability to respond dynamically to requests. Depending on the incoming request, virtual services will be able to provide with appropriate / meaningful response through use of this feature.

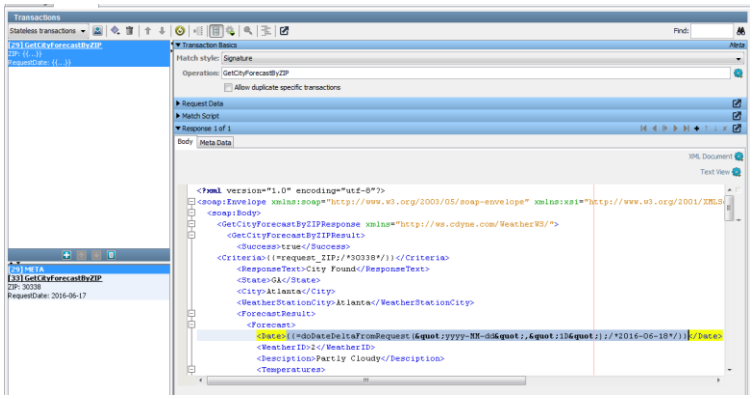
Magic Strings

	Magic String is a form of dynamic parameterization of arguments. VSE examines the request, response and parameterizes the response values that match with the arguments or parameters in the request. For example, consider the addUser operation. When a new user gets added to the application, this service returns the status and new user details. The user id that is present in the response and which is also part of the request is the magic string in this case.

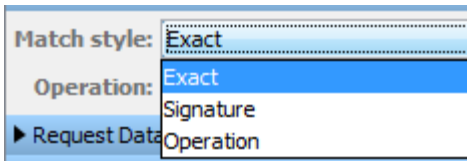
	As shown in the image, the login field matches with the username field of request. Hence, the login field in response is parameterized with the request argument username. The magic string is saved as '=request_username;/*webapp-1809697075554*/'. In this case, even though we recorded addition of only one user to the application, when we send a request with a different user, the virtual service will be able to respond correctly for that user. {{ }} is a standard notation for properties within DevTest. The response will return back 'webapp-1809697075554' as value if there is no value in username ('request_username') argument in the request.

Magic Dates

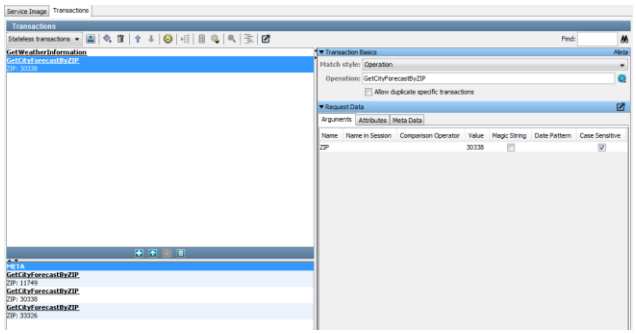
	During recording, the response is parsed and any date values that match a wide range of supported date formats are converted into magic dates. Magic date calculates the delta (difference) of the date value in the response with the current date and populates the value in response. Magic date format is {{=doDateDeltaFromCurrent("Date Format","Delta";/*Default Value*/)}. In the below example, magic date is {{=doDateDeltaFromCurrent("yyyy-MM-dd","-17D");/*2016-06-18*/}}
	In this example, a delta value of '-17D' denotes that the date in response is 17 Days before the current date. The date value will be dynamically computed in response based on the current date time. Some valid parameters for delta are D: Days, H: Hours, M: Minutes, S: Seconds, Ms: Milliseconds.

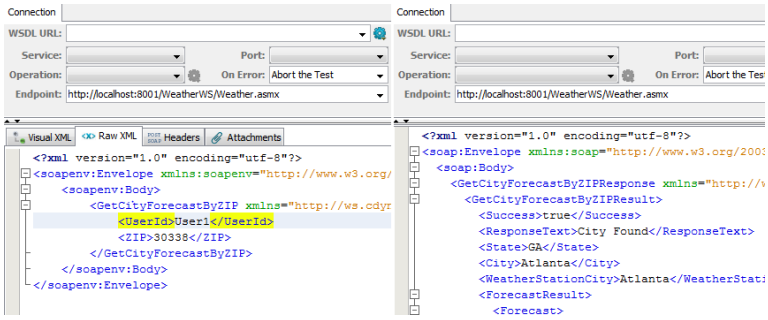
	Another variant of magic date is doDateDeltaFromRequest, which is used when a date is used as a parameter in the request and also seen in the response.
	In this example, we have a date field in the request named RequestDate. We also have a date field in response called Date. The magic string is set as {{=doDateDeltaFromRequest (”yyyy-MM-dd”,”1D”); /*2016-06-18*/}}

Match Styles

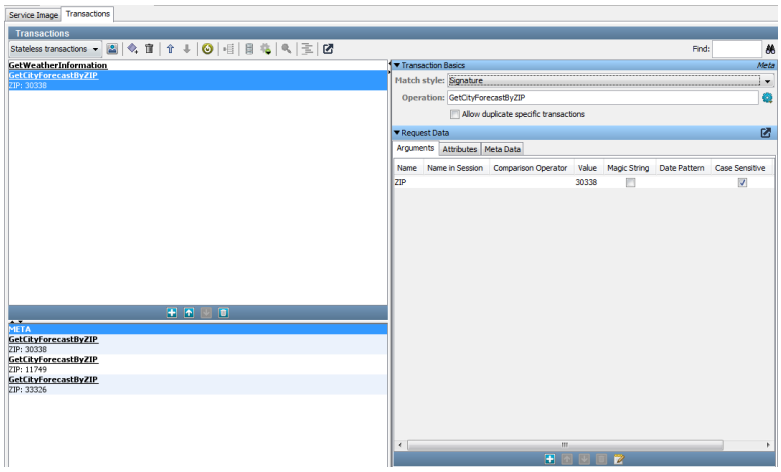
	The different match styles define how matching will be performed between the incoming request and the requests which are part of the captured transactions within the image file. The different match styles within DevTest are as given below. • Operation Match • Signature Match • Exact Match

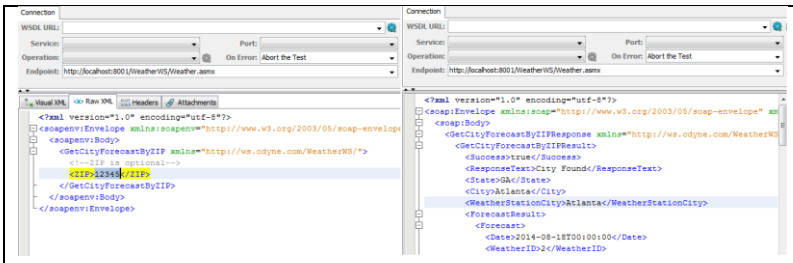
Operation Match

	As the name suggests, in this style of matching, match is performed only on the operation name. This matching style does not consider the signature or the values.
	For example, consider the weather service forecast operation below. The operation has ZIP as the argument in the request signature. If we change the META instance to Operation Match from Signature Match (Default), irrespective of which signature we send, a match will be performed as long as the operation matches.

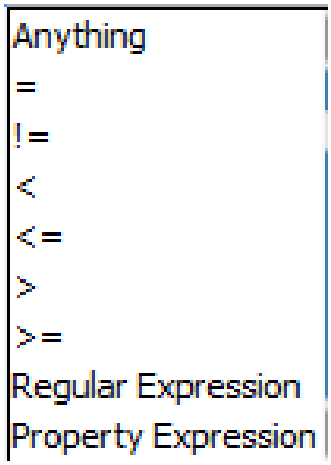
	The request being triggered has a different signature then the captured transaction. Still, in case of operation match, response will be sent back as operation match is successful.

Signature Match

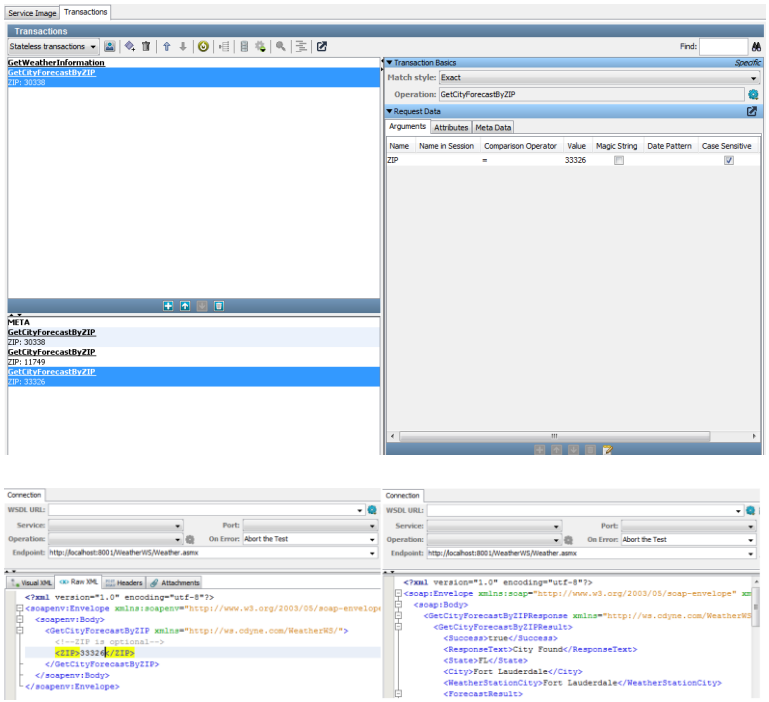
	In this style of matching, match is performed not only on the operation name, but also on the signature of the request. The request argument values are not considered for matching. Meta match is always signature match by default. If the incoming request does not match any of the specific instances that were recorded, as long as the signature matches, it will match the META instance and default response will be sent back.
	As shown in the example, we have recorded the transactions for Weather forecast operation for 3 different locations. If we send a 4th Location which has not been recorded, as long as the layout matches, it will match META instance and send back default response.



Exact Match

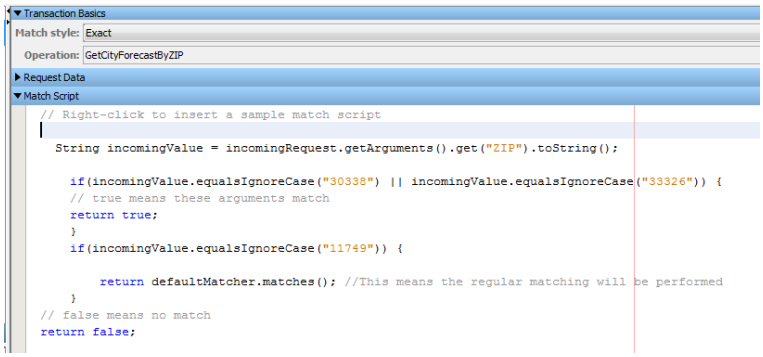


In this style of matching, match should happen on the operation name, signature as well as values being sent as part of request arguments. In case of an exact match, we can define the comparison that needs to be performed with the recorded values through a comparison operator. The different comparison operators are as given below.
Depending on the requirement, one of the comparison operators can be chosen.



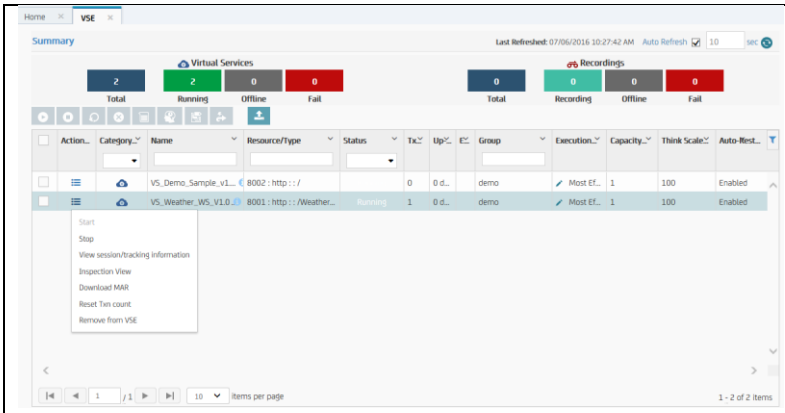
In the example, we are performing match for weather forecast operation. We have a specific instance for ZIP 33326 with forecast for that location as response. When we set the exact match to be equal to '=' that value, when we trigger a request with this zip code, it should match that instance and return weather forecast for Fort Lauderdale.

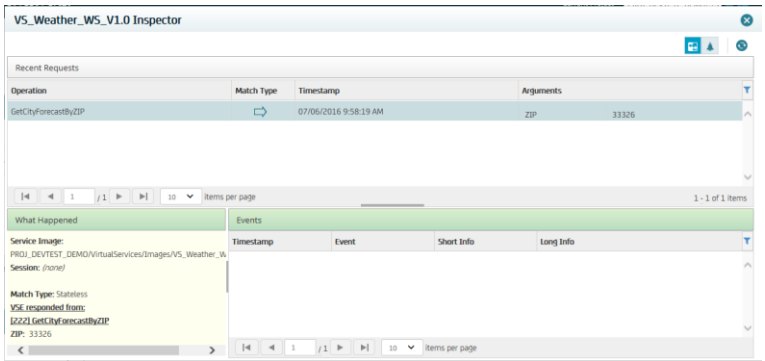
Match Script

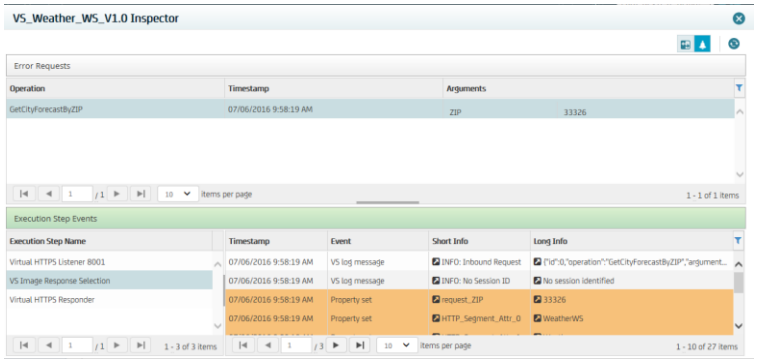
	In some of the scenarios where the existing matching styles do not meet the requirements, we have the option of defining a match script to perform the matching. The match script will over write the normal matching mechanism. We can right click in the Match Script section to insert a sample match script with some commonly used methods, which will be useful in writing a match script to meet the requirements. A match script must return a Boolean value (either true or false). In case a match script returns true, match will be performed on that instance. If a match script returns false, the matcher will proceed to the next instance for matching. One other return value that can be sent by a match script is 'defaultMatcher.matches()'. Returning this value will access the normal matching engine. It evaluates the requests for a match as though there were no match script, and returns a boolean - false if they do not match, true if they do.
	An example match script can be as given. In this example, the match script is defined for the instance of ZIP '11749'. In case of ZIP values 30338 and 33326, response of ZIP '11749' is only sent and also for ZIP 11749, regular matching will be performed ignoring the match script. For all other ZIP values, this instance will not be matched.

	In a real world scenario, when we want a certain response to be sent back for multiple inputs, we can define a match script where we look for those inputs and match that particular instance, instead of defining individual instances for all of those values.

Debugging Matching







We can debug matching by looking at the inspection view in DevTest Portal. In order to debug, we need to enable display of id's (explained earlier in the document) in workstation, which will help us identify different instances, due to a unique ID assignment to the instances.

We can perform match debugging by changing the logging properties in logging.properties file as below.

log4j.logger.VSE can be set to INFO mode to see the matching details in the inspection view. It can also be set to DEBUG or TRACE for more detailed logs in vse_xxx.log file, where xxx is the service image name.

It is recommended to set log4j.logger.VSE property to INFO or WARN for production use.

Logging.properties file

logging.properties

```
log4j.logger.profiler=OFF

log4j.appender.A1=com.itko.util.log4j.TimedRollingFileAppender
log4j.appender.A1.File=${lisa.tmpdir}/${LISA_LOG}
log4j.appender.A1.MaxFileSize=10MB
log4j.appender.A1.MaxBackupIndex=5
log4j.appender.A1.layout=org.apache.log4j.EnhancedPatternLayout
log4j.appender.A1.layout.ConversionPattern=%d{ISO8601}{UTC}Z (%d{HH:mm}) [%t] %-5p %-30c - %m%n



# Keep a separate log for VSE transaction match/no-match events, this makes debugging much easier.
# Change INFO below to WARN for production systems, the logging is expensive and will slow down
# systems with high transaction rates. Do not simply comment out the following line; explicitly
# set the log level to OFF or WARN instead of INFO
log4j.logger.VSE=DEBUG, VSEAPP
# If you want to add VSE logging to other log destinations, comment out the next line
log4j.additivity.VSE=false
log4j.appender.VSEAPP=com.itko.util.log4j.TimedRollingFileAppender
log4j.appender.VSEAPP.File=${lisa.tmpdir}/vse_matches.log
log4j.appender.VSEAPP.MaxFileSize=10MB
log4j.appender.VSEAPP.MaxBackupIndex=20
log4j.appender.VSEAPP.layout=org.apache.log4j.EnhancedPatternLayout
log4j.appender.VSEAPP.layout.ConversionPattern=%d{ISO8601}{UTC}Z (%d{HH:mm})[%t] %-5p - %m%n

# Keep a separate log for advisory events. This logger warns of potential configuration issues, potential
# memory leaks, etc. It is deliberately kept separate to the application log to minimize noise.
```

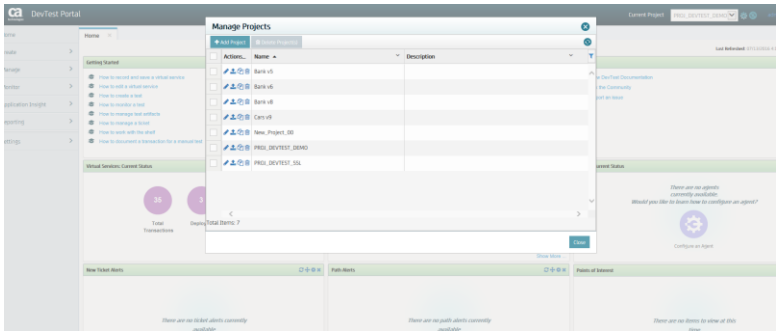
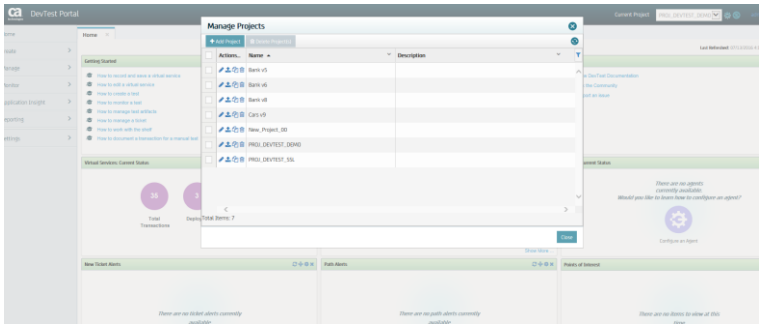
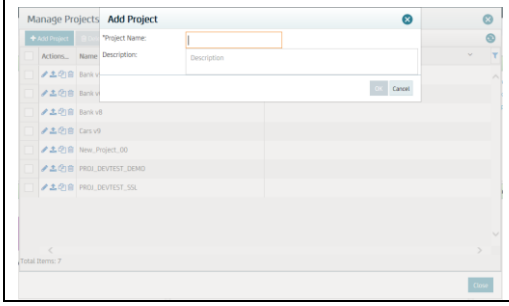
VS_VS_Weather_WS_V1.0.log

```
2016-07-06 14:34:21,334Z (10:34)[VS_Weather_WS_V1.0 [VS_VS_Weather_WS_V1.0_Run]/1] INFO  - No Session ID No session identified
2016-07-06 14:34:21,334Z (10:34)[VS_Weather_WS_V1.0 [VS_VS_Weather_WS_V1.0_Run]/1] INFO  - No Stateless Match Could not match a stateless transaction
2016-07-06 14:34:21,339Z (10:34)[VS_Weather_WS_V1.0 [VS_VS_Weather_WS_V1.0_Run]/1] WARN  - No Match Request <?xml version="1.0" ?>

<request operation="GetCityForecastByZIP" matchTolerance="EXACT">
  <arguments>
    <parameter name="UserId">User1</parameter>
    <parameter name="ZIP">30338</parameter>
  </arguments>
  <attributes>
    <parameter name="HTTP-Segment-Attr-0">WeatherWS</parameter>
    <parameter name="HTTP-Segment-Attr-1">Weather.asmx</parameter>
    <parameter name="xmlns">http://ws.cdyne.com/WeatherWS/</parameter>
    <parameter name="recorded_raw_request">&lt;?xml version=&quot;1.0&quot; encoding=&quot;utf-8&quot;?&gt;&#13;&#10;&lt;soapenv:Envelope xmlns:soapenv="&quot;http://www.w3
  </attributes>
  <metaData>
    <parameter name="HTTP-Method">POST</parameter>
    <parameter name="HTTP-URI">/WeatherWS/Weather.asmx</parameter>
    <parameter name="HTTP-Version">1.1</parameter>
    <parameter name="Content-Type">text/xml; charset=utf-8</parameter>
    <parameter name="SOAPAction">&quot;http://ws.cdyne.com/WeatherWS/GetCityForecastByZIP&quot;</parameter>
    <parameter name="User-Agent">Axis/1.4-LISA</parameter>
    <parameter name="lisaFrameRemoteIP">192.168.1.8</parameter>
    <parameter name="lisaFrameRoot">true</parameter>
    <parameter name="lisaFrameID">0a468730-4386-11e6-acb-a436d9e692a4</parameter>
    <parameter name="Content-Length">390</parameter>
    <parameter name="Host">localhost:8001</parameter>
    <parameter name="Connection">Keep-Alive</parameter>
    <parameter name="lisa.vse.request.client.id">127.0.0.1:59653</parameter>
  </metaData>
  <body charset="UTF-8">&lt;?xml version=&quot;1.0&quot; encoding=&quot;utf-8&quot;?&gt;&#13;&#10;&lt;soapenv:Envelope xmlns:soapenv="&quot;http://www.w3.org/2003/05/soap-e
</request>

2016-07-06 14:34:21,360Z (10:34)[VS_Weather_WS_V1.0 [VS_VS_Weather_WS_V1.0_Run]/1] INFO  - Response Lookup Completed
2016-07-06 14:34:23,065Z (10:34)[VS_Weather_WS_V1.0 [VS_VS_Weather_WS_V1.0_Run]/1] INFO  - Inbound Request {"id":0,"operation":"GetCityForecastByZIP","arguments":{"UserId"
2016-07-06 14:34:23,065Z (10:34)[VS_Weather_WS_V1.0 [VS_VS_Weather_WS_V1.0_Run]/1] INFO  - No Session ID No session identified
2016-07-06 14:34:23,066Z (10:34)[VS_Weather_WS_V1.0 [VS_VS_Weather_WS_V1.0_Run]/1] INFO  - No Stateless Match Could not match a stateless transaction
2016-07-06 14:34:23,066Z (10:34)[VS_Weather_WS_V1.0 [VS_VS_Weather_WS_V1.0_Run]/1] WARN  - No Match Request <?xml version="1.0" ?>

<request operation="GetCityForecastByZIP" matchTolerance="EXACT">
  <arguments>
    <parameter name="UserId">User1</parameter>
    <parameter name="ZIP">30338</parameter>
  </arguments>
  <attributes>
```

Service log file logging after setting log style to DEBUG.

Matching in Portal

DevTest portal is web based app that is planned to be the primary user interface for DevTest. At this time, portal provides a subset of most commonly used DevTest features. With every release, more and more features are getting added to the DevTest portal.We can view virtual services information in DevTest portal. At this time, we can only view the virtual service image related information in the DevTest Portal. In order to view service image, we need to first set the project where the image file is present.

	The project name can be selected from the drop down 'Current Project' on top right of the portal page as shown.
 	If we would like to create a new project, we can do so by clicking on the gear icon  next to the drop down as shown below.