**REST Project**

**Create a new Java project for Integrated Voice and Route, call it “restproject”,**  using the steps as defined previously**.**

## Making the project changes

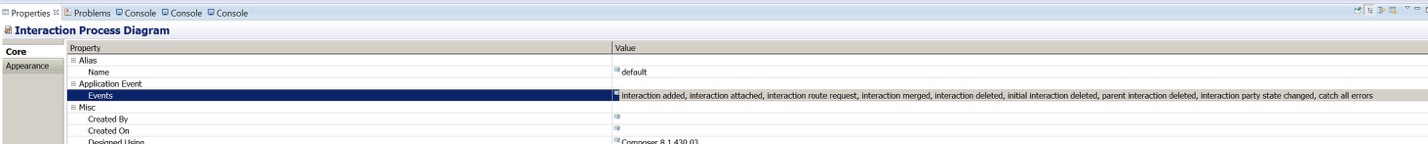
Having created the project, you now have a raw shell of a project. We will be using two key files from here. The Interaction Process Diagram and the default.workflow diagram. By default, when using a blank template these two are configured to point to each other, so we don't have to do any additional work to set that up.

However, since we are not creating an interaction related workflow we will need to modify the IPD to remove the interaction related event that it is waiting for. If we do not do this your state machine will hang waiting for an event that will never occur.

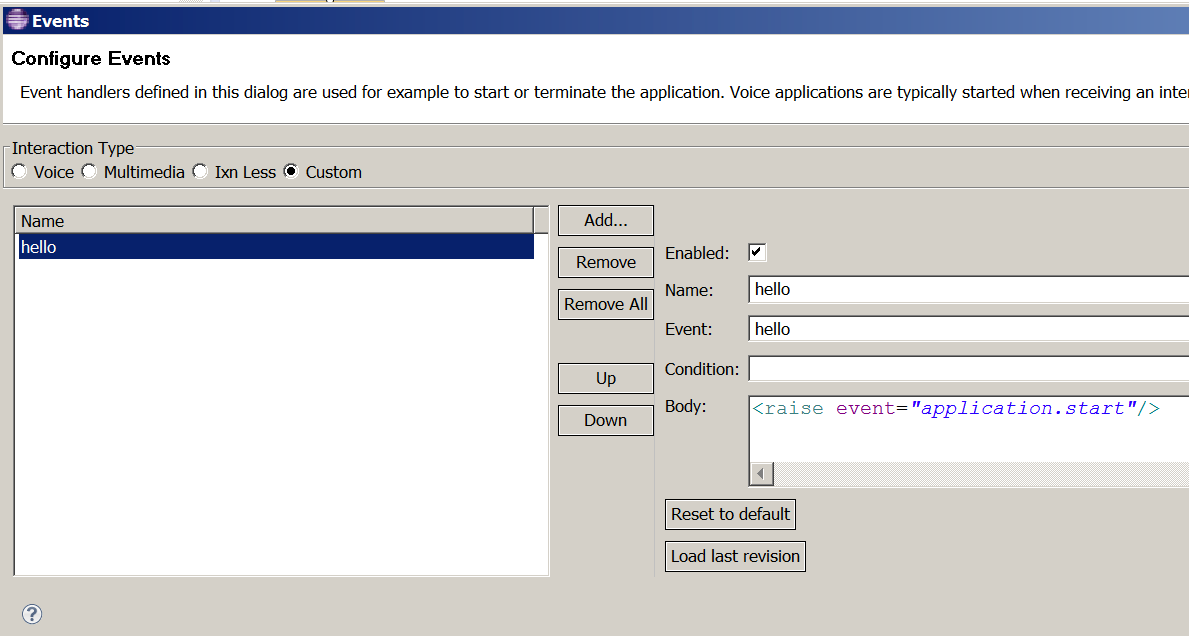
### Clear the Wait for Event within the IPD

Within the Interaction Process Diagram (IPD) we typically wait for an interaction event, since we are not going to interact with the session via an Interaction we need to clear this. We essentially need to have it as blank so that we can just run the application and have it terminate.

Here is how the events structure should look like…



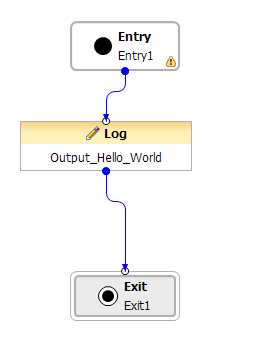
Erase all these events and add a “custom” event…



This now allows us to have an interaction less workflow that will not wait for an initial event. Once loaded it will start the workflow.

## Creating the Hello World Structure

Within the default.workflow we need to create the following simple structure



The Log Block is very simply and just needs to have the following details provide for it in its properties.

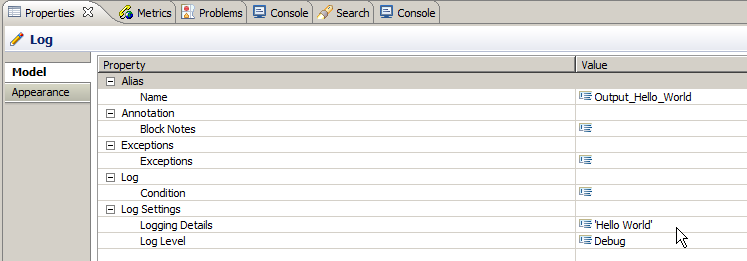


Figure 5: Hello World Block Properties

### Saving and Generating the SCXML Code

When creating diagrams, we need to go through a process of code generation. This typically is initiated manually and is required whenever a change is made. Unsaved changes are highlighted in the tab of each diagram file as followed when open.



Figure 6: Unsaved Changes

The '\*' before the file name is shown to show it needs to be saved, Using ALT+G to save and generate the file. This will first prompt you to save the file. You can also click the check box to remove this dialog box for future times you generate code.

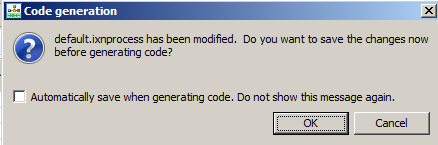


Figure 7: Code Generation Dialog

Once saved the code is then generated and the following is shown.

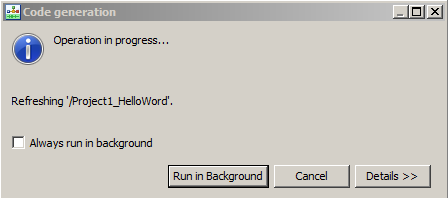
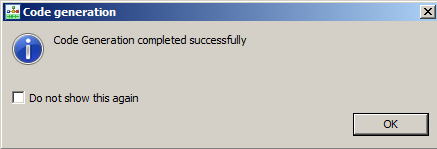
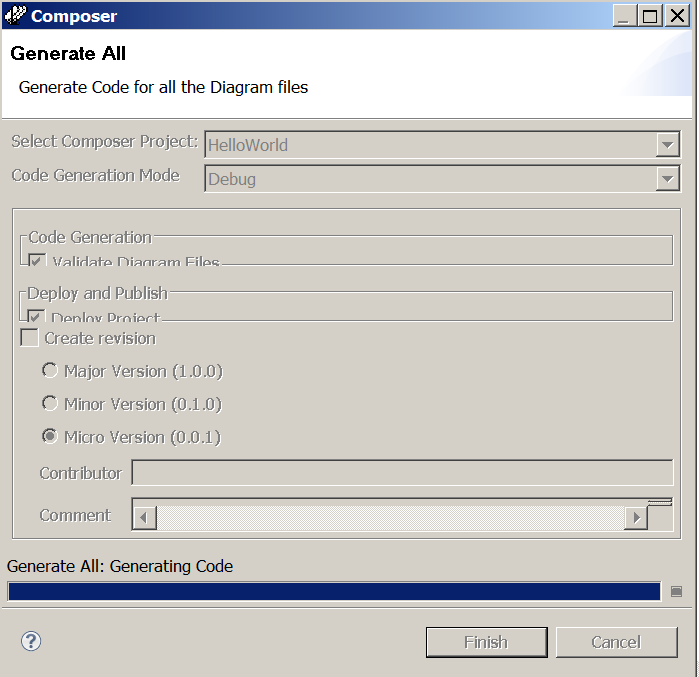


Figure 8: Running Code Generation Dialog

Again, here you can always once you get use to this flow select never to show this dialog again. Upon completion you should see the following.





### Checking that we have the Project Deployed

Now let’s see if it is deployed or not. Right click on project. Then select tomcat properties

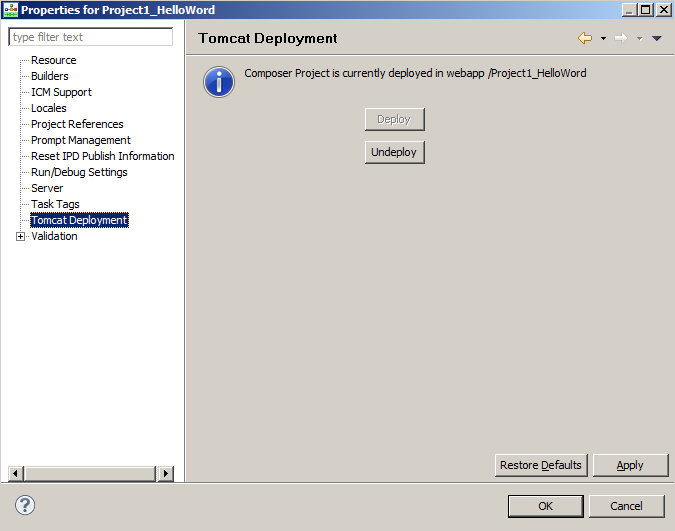


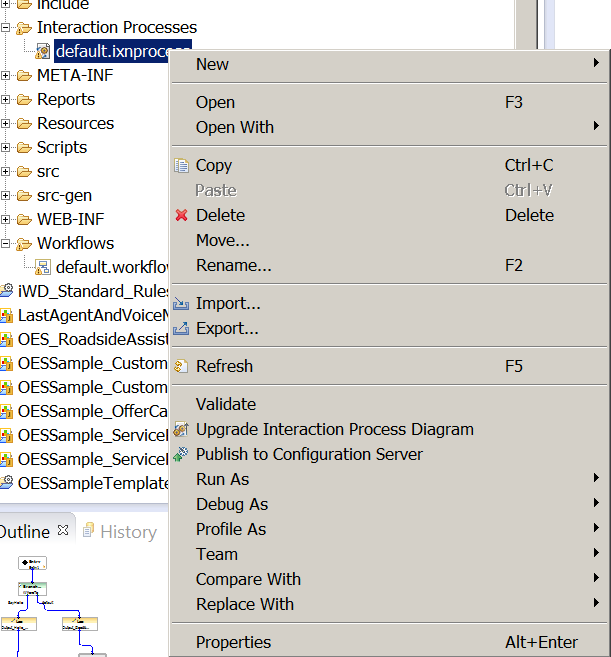
Figure 10: Composer Tomcat Deployment Check

Note: - If Both Deploy and Un-deploy are grayed out, then you will need to ensure that the port, and user name and password have been set in the Tomcat Preferences.

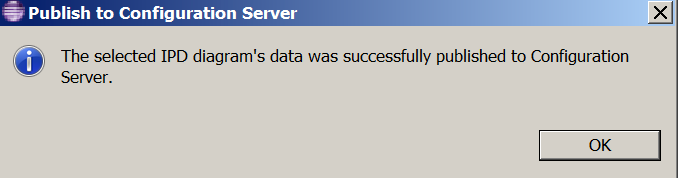
If there is an error, then we might need to revisit the Tomcat configuration. Alternatively, we can check the deployment by looking at the actual instance of Tomcat that is shipped with Composer.

Connect to Configuration Server

Right click on the default.ixnprocess, choose – “ Publish to Configuration Server”



You should get the following response …



Further still, if you want to check the Tomcat deployment, again you can refer to <http://localhost:9001/> through a web page.

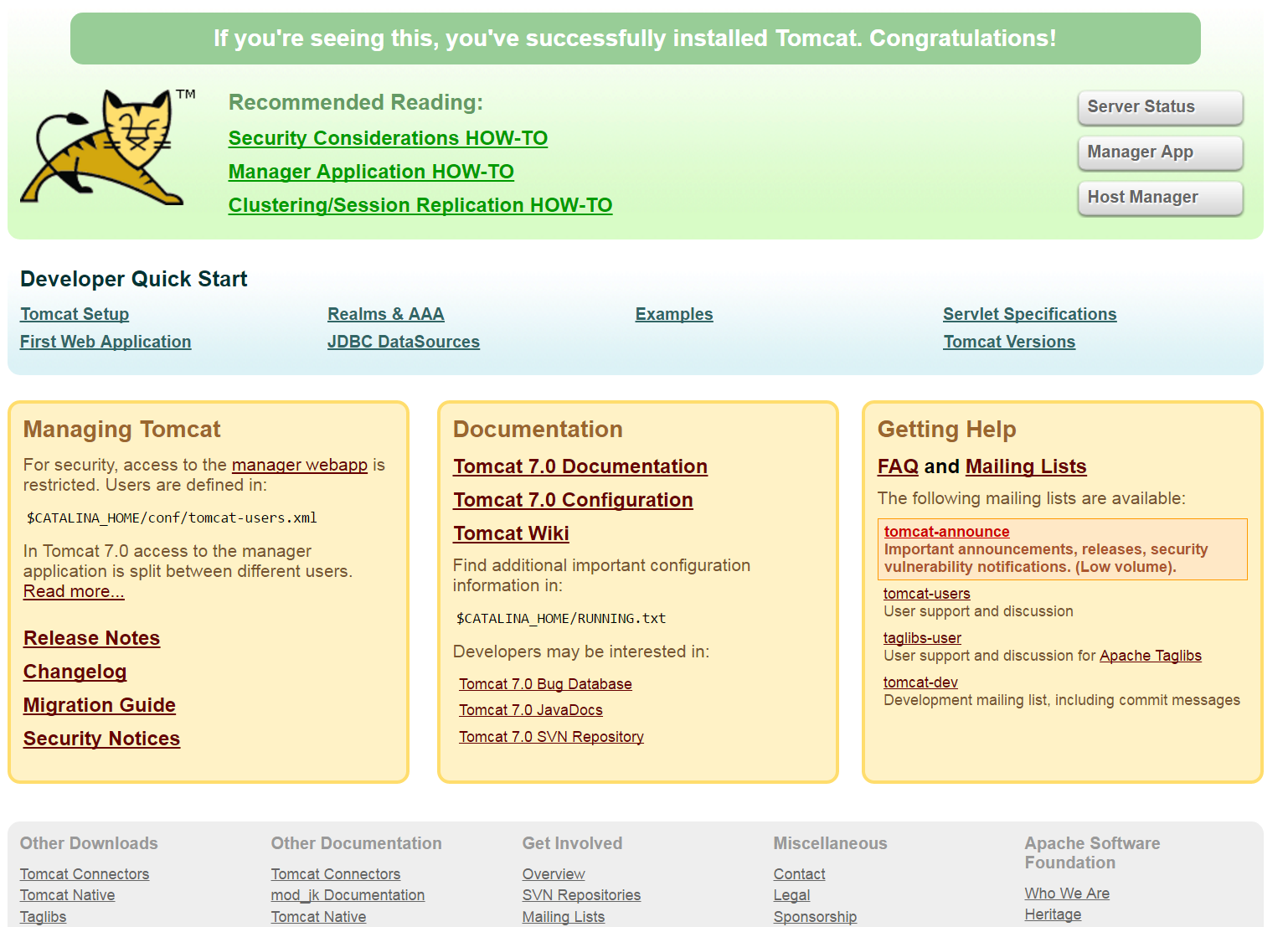


Figure 11: Apache Tomcat Confirmation page

Click on Tomcat Manager App and enter the login details of demo/

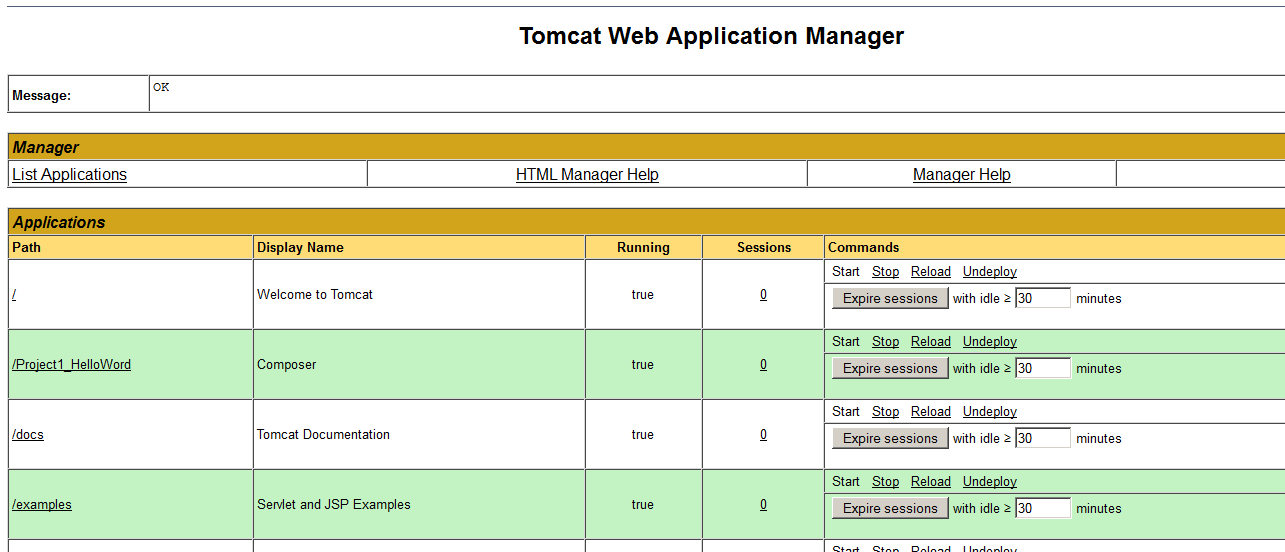


Figure 12: Tomcat Web Application Manager

### Checking the application further

You can also check to see what has been deployed by also requesting the SCXML directly across a browser

<http://192.168.10.98:9001/HelloWorld/src-gen/IPD_default_defaultWorkflow.scxml>

And you should get something like this…

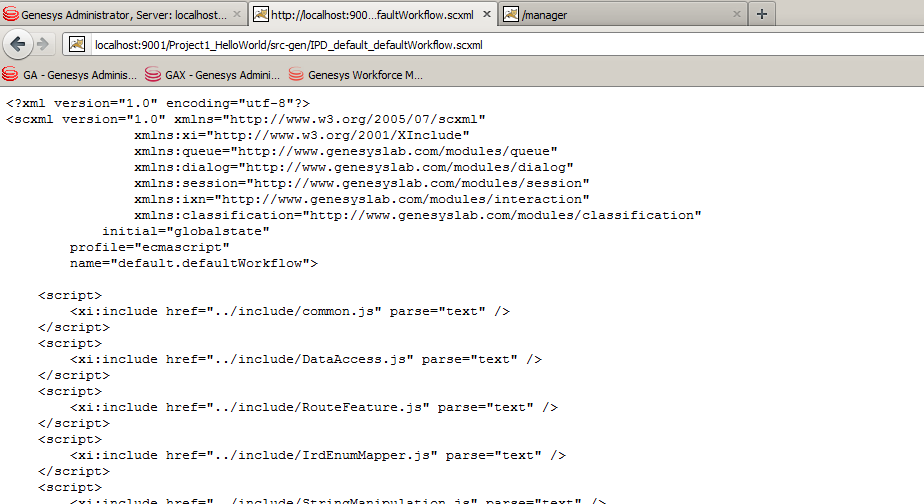


Figure 13: Browser retrieval of SCXML Code

If you get something like the following, then your application is not deployed. Try un-deploy and deploy from composer again.

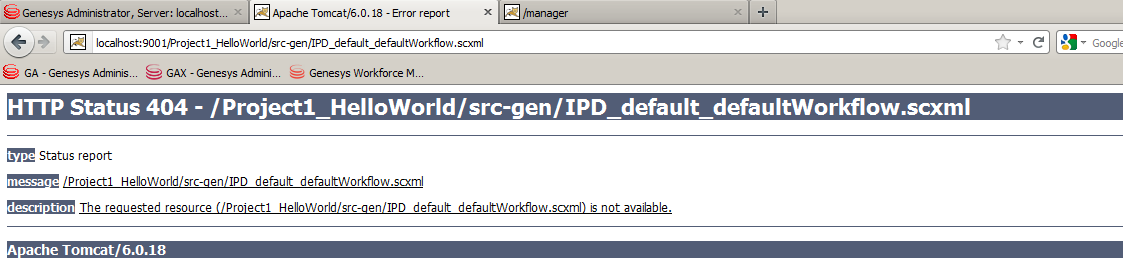


Figure 14: SCXML file not found

You will likely see this in Composer when this happens

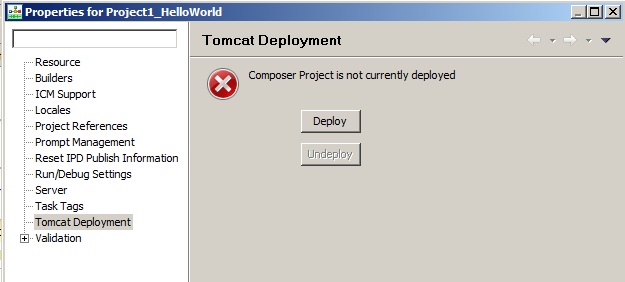


Figure 15: Composer Deploy button

## Starting an SCXML Session

We have the SCXML application deployed and it can now be executed by the ORS web interface.

### ORS Web Services Interface

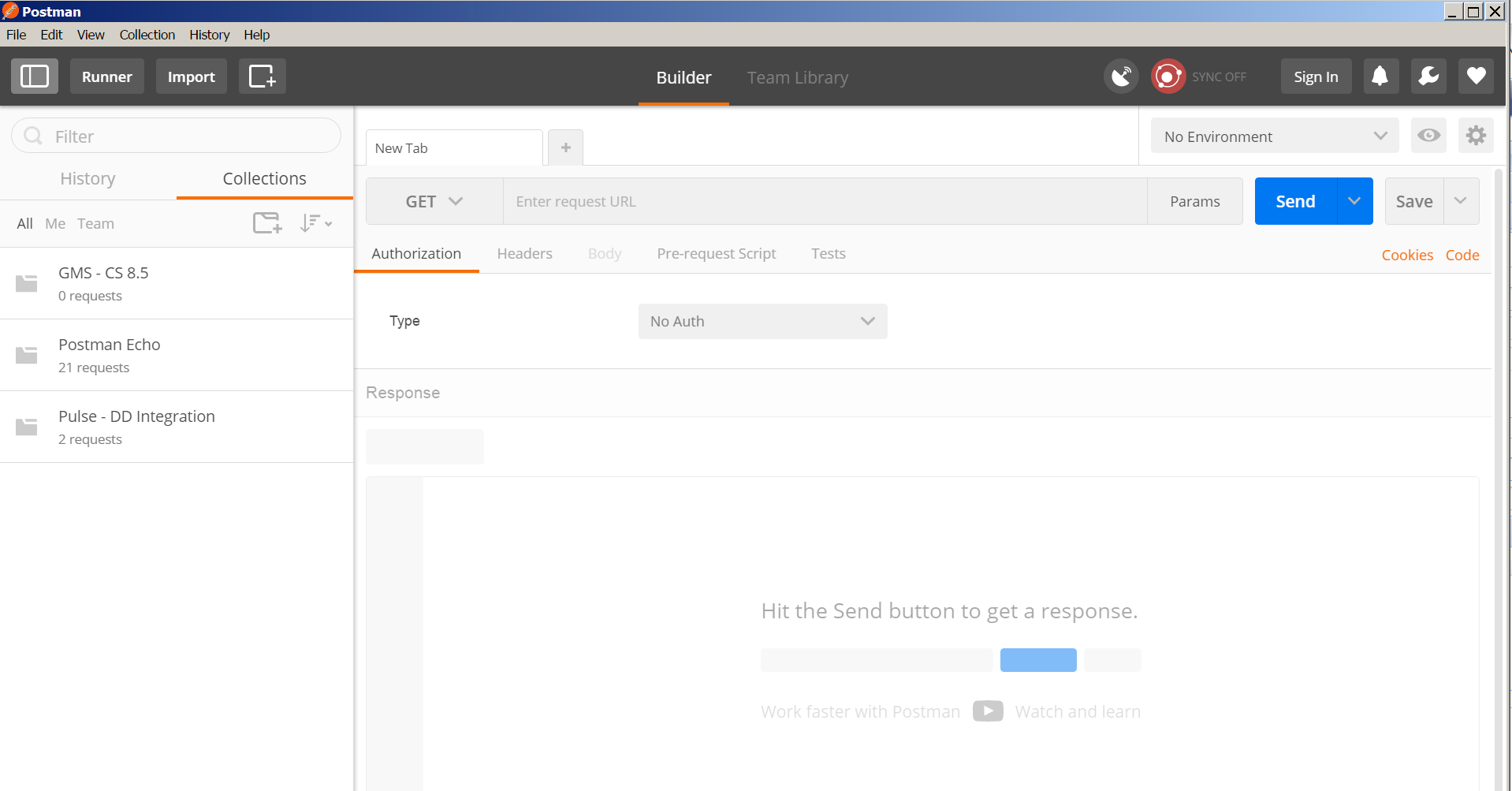
ORS support interaction with external clients across its RESTful Web Services API which are documented in detail here

<http://docs.genesyslab.com/wiki/index.php?title=External_Interfaces>

### WizTools RESTClient

For initiating any request to ORS that is not interaction related, we will be using the REST Client from WizTools this can be downloaded from the following location. But, it’s already installed on your machine. Search for REST client – open Postman REST client.

<http://code.google.com/p/rest-client/>



Use GET and place the following REST request –

<http://demosrv:7210/scxml/session/start?src=http://192.168.10.98:9001/HelloWorld/src-gen/IPD_default_defaultWorkflow.scxml>

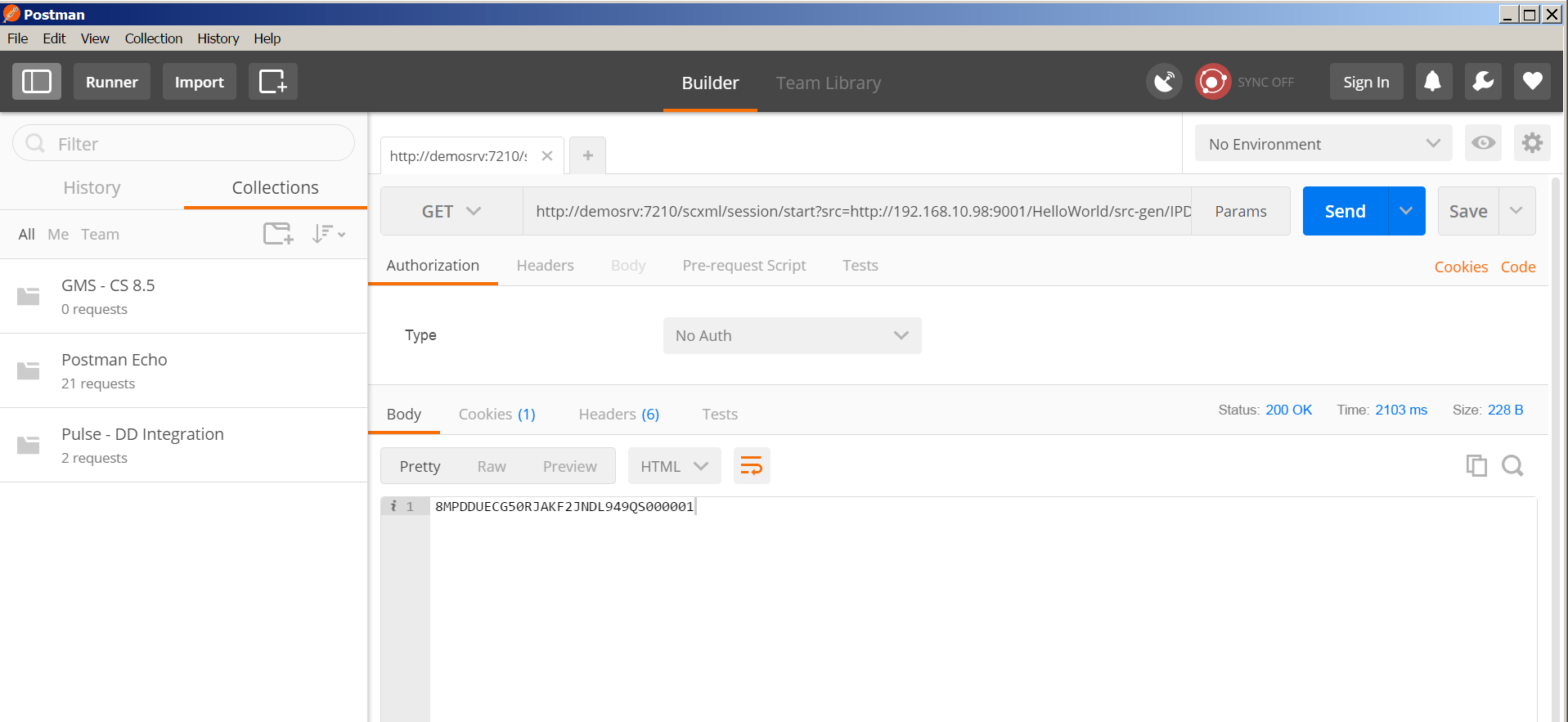


Figure 16: RestClient Interface

Then look at the log files which are on the TM Interface located in C:\logs\Orchestration server

### Querying the running session

As a part of the ORS web interface we have a standard query interface which can be used to enquire on running sessions. This request can be submitted again by the rest client if we know the session ID that we are looking for which we can obtain from the Log File.

Performing the following request

http://demosrv:7210/scxml/session/8MPDDUECG50RJAKF2JNDL949QS000001/query

Returns a JSON payload in the return body as follows

{

"URL":"http://demosrv:9001/Project1\_HelloWorld/src-gen/IPD\_default\_defaultWorkflow.scxml",

"name":"default.defaultWorkflow",

"type":"",

"current states":["globalstate", "defaultWorkflow\_listening"],

"events":[],

"\_genesys":{"session":{"day":{"Friday":5,"Monday":1,"Saturday":6,"Sunday":0,"Thursday":4,"Tuesday":2,"Wednesday":3},"lookupseq":{"StartFromCDN":0,"StartFromRouter":3,"StartFromStrategy":-1,"StartFromTenant":2,"StartFromTserver":1},"server":{"cluster":"wb\_cluster","name":"OrchestrationServer"},"tenant":"Environment","tenantid":1},"statistic":{},"ixn":{"firstixnid":"","interactions":{"0":{}},"mediaType":{"TMediaAlert":16,"TMediaAppSharing":9,"TMediaBusinessEvent":15,"TMediaCallback":12,"TMediaChat":5,"TMediaCobrowsing":7,"TMediaEMail":2,"TMediaFax":13,"TMediaIMChat":14,"TMediaNativeSMS":20,"TMediaOpenMedia":19,"TMediaOutboundPreview":18,"TMediaSMS":17,"TMediaSMail":4,"TMediaVMail":3,"TMediaVideo":6,"TMediaVoIP":1,"TMediaVoice":0,"TMediaWebform":10,"TMediaWhiteboard":8,"TMediaWorkItem":11}},"queue":{"overwriteType":{"UseANI":1,"UseConfig":3,"UseDNIS":2,"UseNone":0,"UseValue":4},"quotaType":{"QuotaMax":2,"QuotaMin":0,"QuotaTarget":1},"rType":{"RouteTypeAgentID":13,"RouteTypeAnnouncement":8,"RouteTypeCallDisconnect":14,"RouteTypeDDD":4,"RouteTypeDefault":1,"RouteTypeDirect":6,"RouteTypeDirectAgent":10,"RouteTypeDirectPriority":12,"RouteTypeGetFromDN":12,"RouteTypeIDDD":5,"RouteTypeLabel":2,"RouteTypeOverwriteDNIS":3,"RouteTypePostFeature":9,"RouteTypePriority":11,"RouteTypeReject":7,"RouteTypeUnknown":0},"statcond":{"ReadyIfGreater":1,"ReadyIfLess":0,"ReadyIfNotGreater":2,"ReadyIfNotLess":3},"usecapcond":{"Never":1,"OnStatError":0,"Only":2}},"resource":{"resourceType":{"CFGACDPosition":2,"CFGACDQueue":3,"CFGCP":10,"CFGCellular":9,"CFGChat":27,"CFGCoBrowse":28,"CFGCommunicationDN":23,"CFGData":12,"CFGDestinationLabel":20,"CFGEAPort":7,"CFGEmail":24,"CFGExtRoutingPoint":19,"CFGExtension":1,"CFGFAX":11,"CFGMixed":18,"CFGMusic":13,"CFGNoDN":0,"CFGRoutingPoint":4,"CFGRoutingQueue":22,"CFGServiceNumber":21,"CFGTieLine":16,"CFGTieLineGroup":17,"CFGTrunk":14,"CFGTrunkGroup":15,"CFGVideo":26,"CFGVirtACDQueue":5,"CFGVirtRoutingPoint":6,"CFGVoIP":25,"CFGVoIPService":29,"CFGVoiceMail":8,"CFGWorkflow":30,"any":1000}}},

"\_data":{"provision\_type":"Web","server":"OrchestrationServer"}

}

Query provides a snapshot of where the session is currently. It should be used sparingly in production but is a good tool to check just what is happening. And may be where but not why you are apparently stuck.

The returned data is in JSON and can be viewed in various tools in a more readable manner. I personally prefer the online tool located here

http://jsonviewer.stack.hu

This allows you to view the JSON text data in a more structured manner if you copy it into the tool. Notepad++ also has a similar feature too. The best tool probably would be to download JSONFormat.



Figure 12: Pretty print of JSON from jsonviewer.stack.hu

### Sending in event we are waiting on

To move the session along we need to send it the 'hello' event. Here we use another web API call called event

Let’s first call it with upper case

<http://demosrv:7210/scxml/session/8MPDDUECG50RJAKF2JNDL949QS000001/event/HELLO>

Using the logs, we see the following



Figure 13: Event Receipt in Logs

Note that there was NO transition taken. We simply do not match the event as it is the wrong case. Submit now the correct event 'hello'. Note RestClient may try and still use the previous URL, so they may need to restart the rest client.

<http://demosrv:7210/scxml/session/8MPDDUECG50RJAKF2JNDL949QS000001/event/hello>

Once this is submitted we will now see the following

METRIC <event\_processed sid='PGPBM6D23H1534J6PJA7S66FIG000002' name='hello' disposition='transition selected' />

Metrics are used to track what is happening within the SCXML engine. Here we can see that a transition was selected. i.e. the event was not ignored. Metrics will be seen more in the debugger later.

### Multiple Events

Currently multiple events can be added however they are applied in a logic OR manner. So if the following is defined.

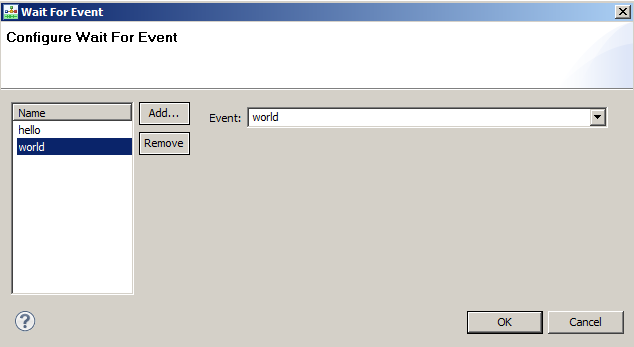


Figure 14: Configuring Multiple Events

It implies that the workflow will be started if either the 'hello' event is sent OR the 'world' event is sent. If you need to do additional conditions for events and conditions, you are best to use no 'wait for event' and place your starting logic in the application.

The current version of Composer comes with a set of Events. Please refer to documentation to for more information.

## Introducing the debugger

The debugger just makes life a lot easier than looking at the log files, but understanding the logs and the manual way of debugging allows you to understand more of what the debugger is showing you.

### Configuring the debugger

1. Need to ensure that ORS has been configured for the debugger



Figure 15: Debugger ORS Configuration

1. Need to configure Composer to access the debugger

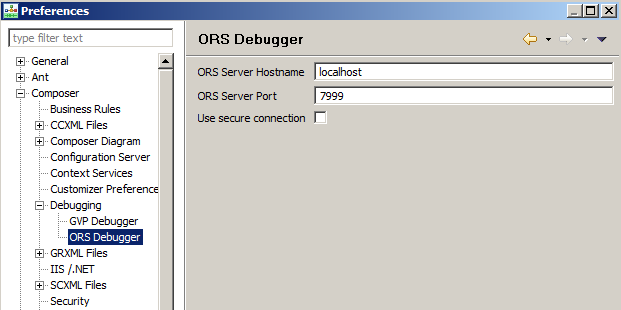


Figure 16: Composer ORS Debugger Configuration

### Start a debug session - Creating the debug configuration

- Select the workflow in the project window. right click and select debug configuration

- Put a name in like project 1

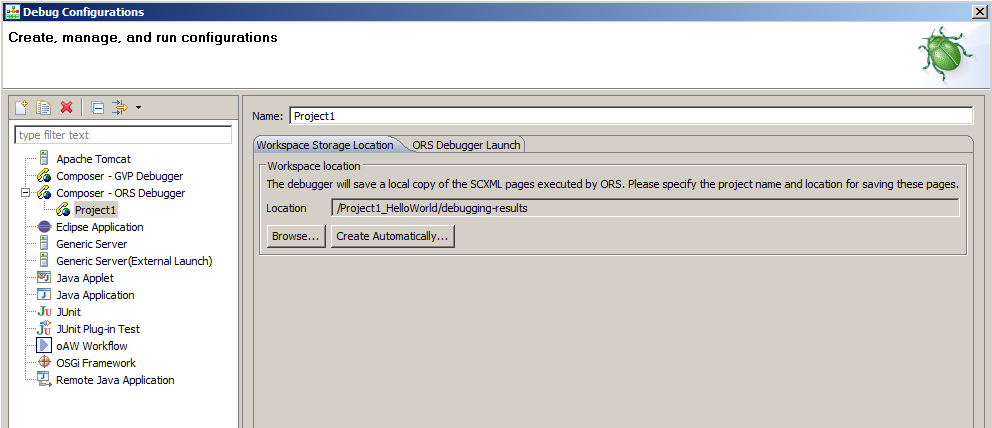


Figure 17: Debug Configuration for Workflow

Select that we have the debug result to capture debug session information (This is the fully rendered SCXML document that ORS sees it as). This can be in the workspace 'create automatically' or within the project if you browse to a suitable directory.

On the ORS Debugger Launch make sure the following is entered

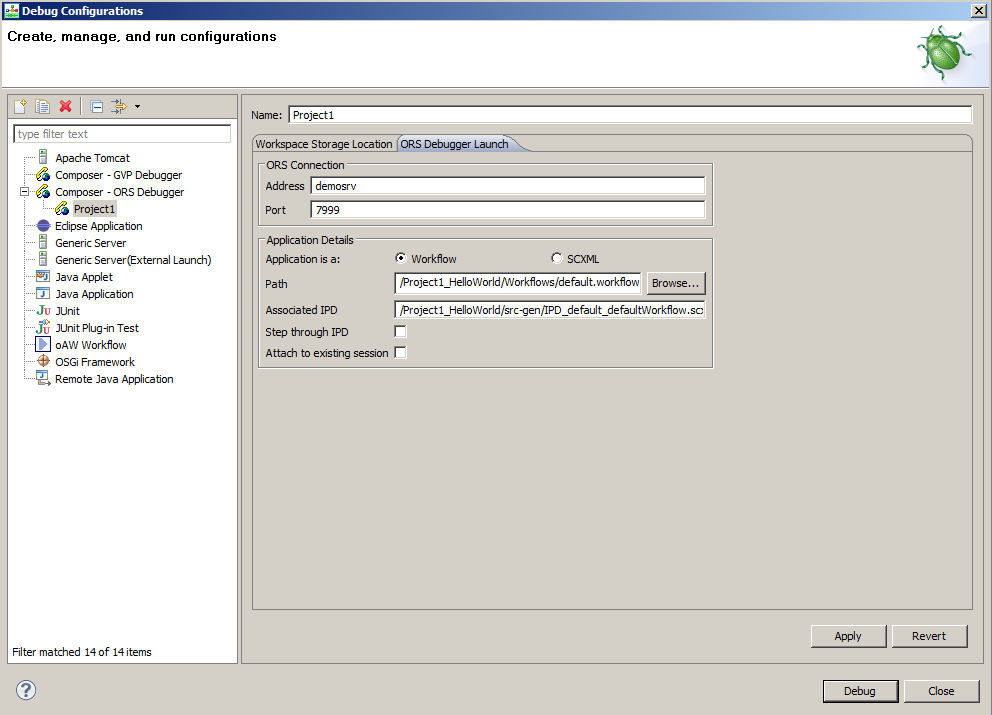


Figure 18: Debug Launch Configuration

Leave the Step through IPD and Attach to existing session blank. Click Debug.

You are now waiting for a session to start. This is referred to as camping on a URL. In the latest release of ORS, you need to provide existing session information for interaction-less interactions. So, choose “Attach to existing session” option before you start debugging.

### Starting the session

Use the rest client to start the session

http://localhost:7210/scxml/session/start?src=http://demosrv:9001/Project1\_HelloWorld/src-gen/IPD\_default\_defaultWorkflow.scxml

We now are able to get back a session ID such as the following PGPBM6D23H1534J6PJA7S66FIG000003

There is a flurry of activity in the debugger. We don’t filter everything the debugger shows for now. Show them how to use the filter within the debugger. Double click the Metrics tab to maximize it.

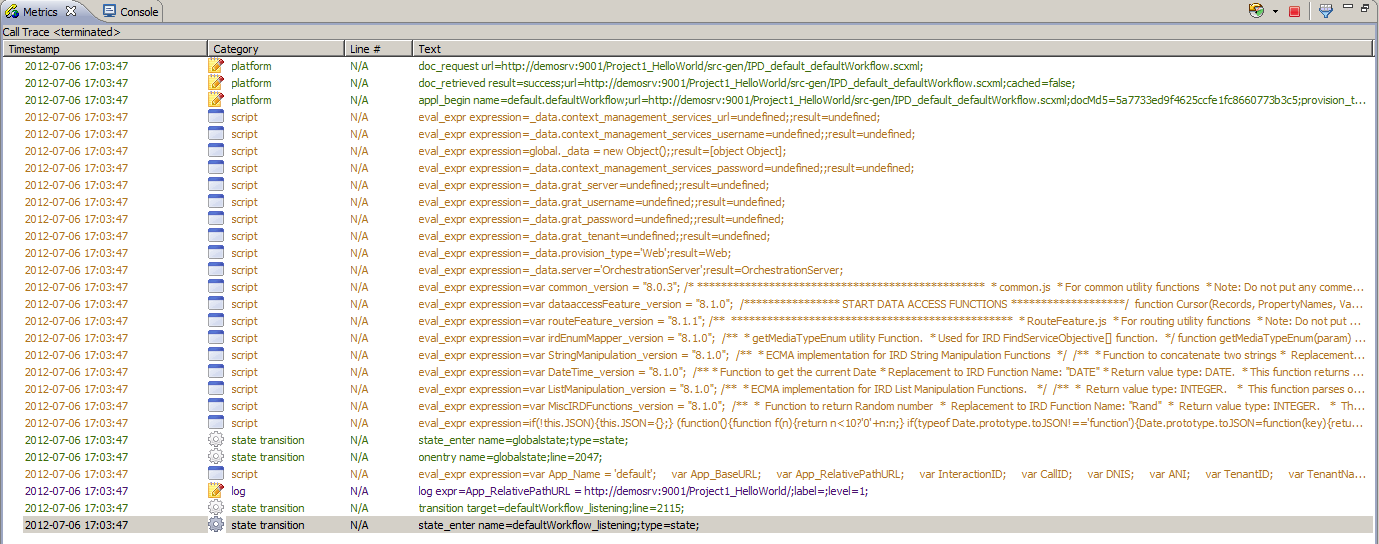


Figure 19: Composer Metric View

We are not yet stopped since the debugger has not hit any break point. Well, we have not set any.

Place a break point on the on Entry block of the default.workflow

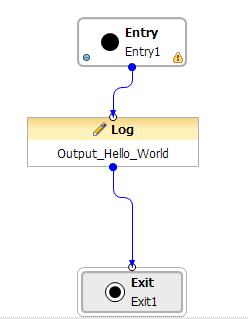


Figure 20: Break point on Entry

Now submit the event, again more information will appear in the debugger and we are now stopped at the break point.

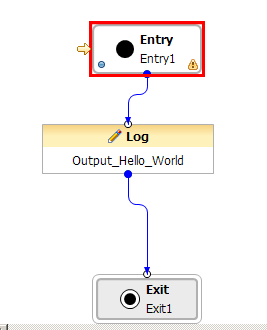


Figure 21: Composer signifies our position

Then show the variables, and ability to step over the application. Next we will modify the behavior slightly.

### Modify the debugged flow

First add a new workflow/diagram variable called 'path'

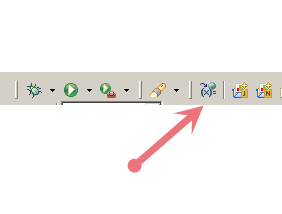


Figure 22: Workflow Variables access from Toolbar

Then in the following window add a new user variable called path. Set the default value to be default

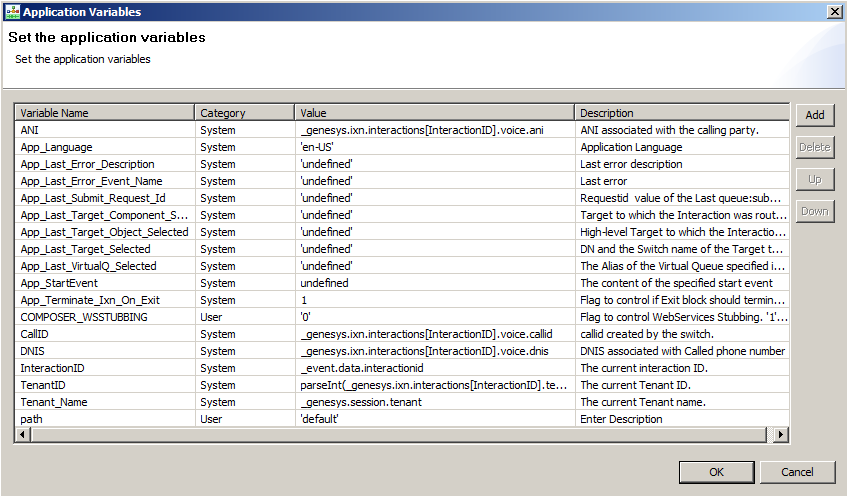


Figure 23: Application Variables

We then modify the flow to look like the following

- Mention about the CTRL click to duplicate blocks and also that we can move labels, add comments and multiple exits to make the diagram more readable.

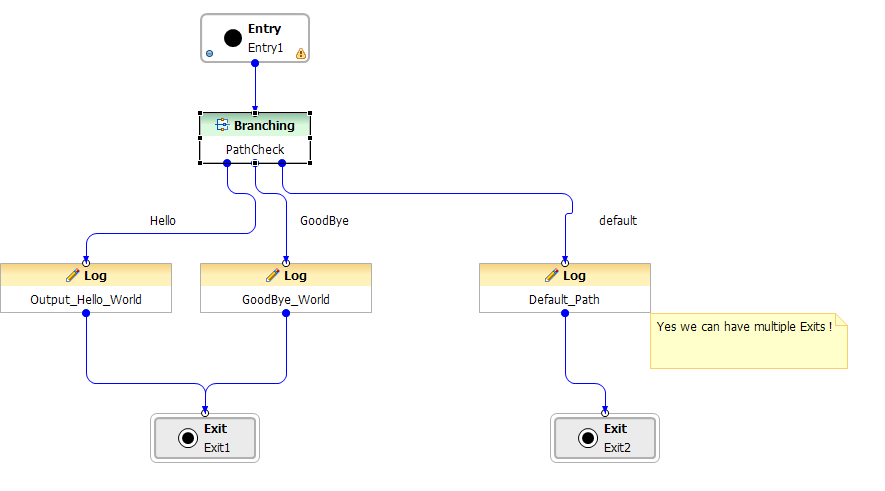


Figure 24: Advanced Hello World

Within the Branching block let’s make a beginners JavaScript mistake, using the equals -

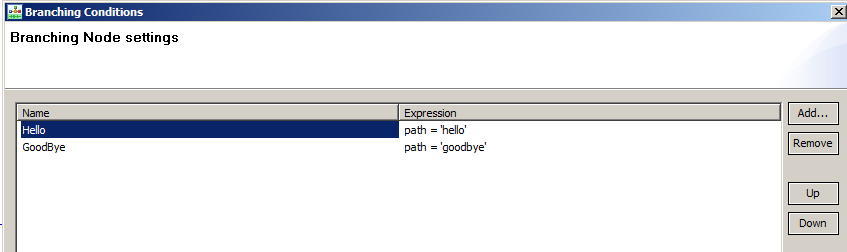


Figure 25: Branching Conditions

Generate the diagram and place break points on the 3 log statements.

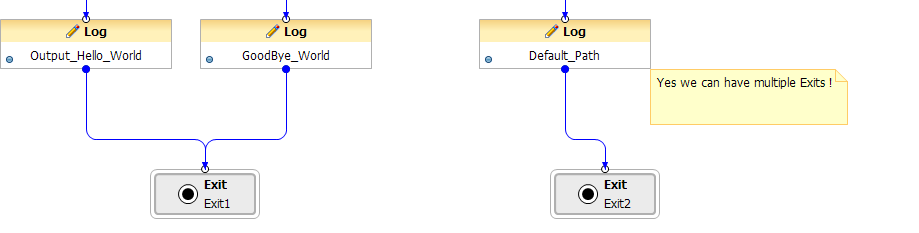


Figure 26: Multiple Exits in one diagram

Start a camp on debug session

Start the session via the rest client and send the initial hello event. We break on the on entry of the workflow. At this point out workflow variable has not been created so let’s hit Step Over.

Use the console and type path to display the current value of it

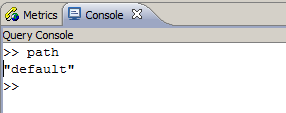


Figure 27: Debugger Console

Next use Step Over we should expect it to go to the default branch but what happens?

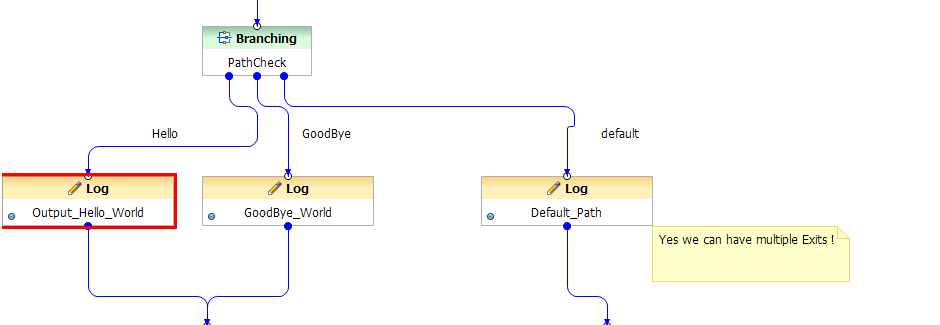


Figure 28: Wrong Path taken

We don’t go where we expect to go. Why?

Let’s check what path is set to now -

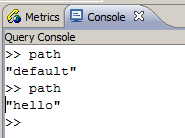


Figure 29: Using Console to inspect variables

Path was assigned a new value. In an equality test this will always result in true which forces the first branch to be taken.

path = "hello" resolves to be true.

We can see this in the metrics.

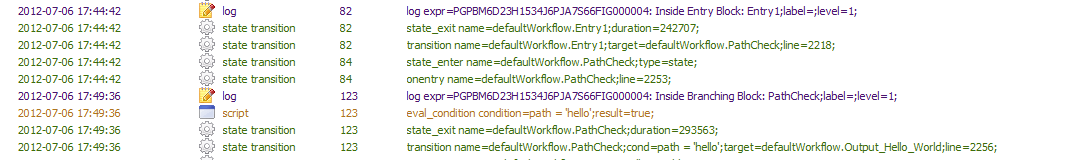


Figure 30: Viewing Metrics

Now, let’s let the session run to completion and start it again. This time we will apply a watch on the path value and also correct the logical error in our code. Use the re-launch from the terminated session to repeat the debug session.

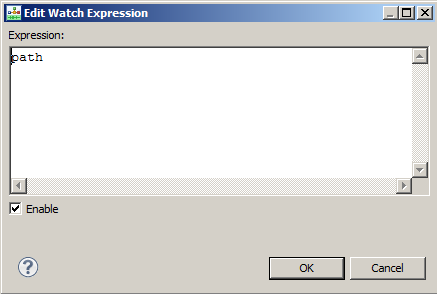


Figure 31: Add watch expression

Start the session, submit the event and see what the watch expression states.

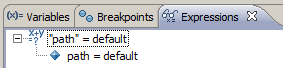
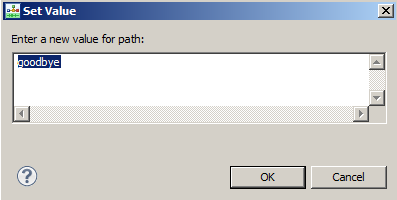


Figure 32: Current state of watch expression

Now let’s change the value to goodbye



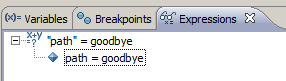


Figure 33: Modifying value from watch expression

Now let’s step. We are now able to control where the application logic branches

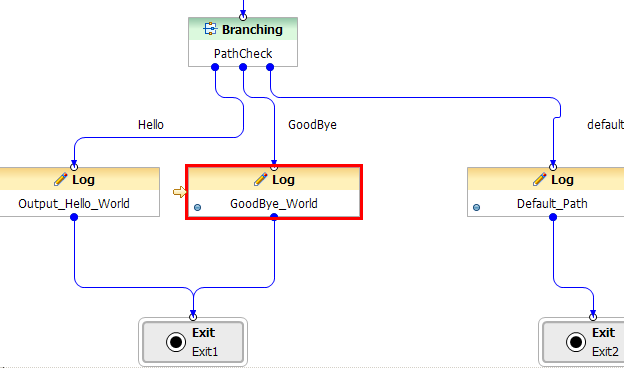


Figure 34: Observing the modified behavior

Note, other possible extensions to this can be (We will address these likely later)

- Obtain the path value from a Transaction List Object, OPM, config variable or Genesys Rules. Show that we can obtain multiple values for a transaction list object.

- Obtain the data from the START request